Scaling Wearable Cognitive Assistance

Junjue Wang

CMU-CS-20-107
April 2020

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

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

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

This research was supported by the National Science Foundation (NSF) under grant number CNS-1518865. Additional support was provided by Intel, Vodafone, Deutsche Telekom, Verizon, Crown Castle, Seagate, VMware, MobiledgeX, InterDigital, and the Conklin Kistler family fund.

Keywords: Wearable Cognitive Assistance, Scalability, Adaptation, Wearable Computing, Augmented Reality, Edge Computing, Cloudlet

To the future of machines.

Abstract

It has been a long endeavour to augment human cognition with machine intelligence. Recently, a new genre of applications, named Wearable Cognitive Assistance, has advanced 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. While previous research has demonstrated the technical feasibility of wearable cognitive assistants, this dissertation addresses the problem of *scalability*. We identify two critical challenges to the widespread deployment of these applications to be 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. To address these challenges, we first design and evaluate adaptation-centric optimizations that reduce resource consumption and improve resource management in contentious systems while maintaining acceptable end-to-end latency. We then propose and implement a new prototyping methodology and a suite of development tools to lower the barrier of application development.

Acknowledgments

I have been extremely fortunate to be inspired and supported by a group of talented and caring individuals throughout my PhD research. This dissertation would not be possible without them.

I am deeply indebted to my advisor Mahadev Satyanarayanan (Satya), who played a paramount role in guiding me through this dissertation research. Satya taught me how to design, implement, and evaluate mobile and distributed systems. His knowledge and encouragement have been a constant source of inspiration for me to work harder and produce more impactful system research. In addition to an excellent academic advisor, Satya has also been a visionary mentor whom I look up to. It has been invaluable for me to observe and learn from his leadership in shaping new technologies and bringing people together for a shared vision. Another person who has inspired and helped me tremendously is Padmanabhan (Babu) Pillai. He is an exemplary researcher with broad and deep knowledge, kindness, and patience. In addition, I would like to thank my thesis committee members, Daniel Siewiorek, Martial Hebert, and Roberta (Bobby) Klatzky. As this dissertation is at the intersection of multiple fields, they have given me many insightful feedback and guidance on human-computer interaction and computer vision. Their comments from our weekly meetings have been vital for me to stay on the correct course.

Brilliant and kind individuals in Satya's research group have provided many rapport and joy during my PhD study. In addition, many ideas and experimental results presented in this dissertation come from close collaborative work with them. They have also provided many system infrastructure support to perform the experiments. I cannot dream of a better team and environment to be in for a PhD student. My day-to-day interactions with them helped me grow as a better researcher and a critical thinker. Especially, I would like to give my appreciation to, in an alphabetical order, Yoshihisa Abe, Brandon Amos, Jim Blakley, Zhuo Chen, Tom Eiszler, Ziqiang (Edmond) Feng, Shilpa George, Kiryong Ha, Jan Harkes, Wenlu Hu, Roger Iyengar, Natalie Janosik, and Haithem Turki. I also want to thank Chase Klingensmith for his outstanding administrative assistance for the group.

My PhD research has also benefited significantly from the larger research community. I was fortunate enough to collaborate with many exceptional academic researchers and industry professionals in the past few years. Thank you all for the inspiration you have given me. In particular, thank you, Pauline Anthonysamy, Mihir Bala, Richard Buchler, Kevin Christensen, Anupam Das, Nigel Davies, Debidatta Dwibedi, Khalid Elgazzar, Wei Gao, Ying Gao, James Gross, Alex Guerrero, Guenter Klas, Zico Kolter, Michael Kozuch, Hongkun Leng, Grace Lewis, Tan Li,

Haodong Liu, Yuqi Liu, Lin Ma, Mateusz Mikusz, Manuel Olguín, Sai Teja Peddinti, Truong An Pham, Norman Sadeh, Rolf Schuster, Asim Smailagic, Nihar B. Shah, Nina Taft, Joseph Wang, Guanhang Wu, Yu Xiao, Shao-Wen Yang, Canbo (Albert) Ye, and Siyan Zhao.

My beloved family and friends have provided enormous understanding, sympathy, and support throughout my PhD study. I could not have finished this thesis research without knowing they are on my back. In particular, I have always felt extremely lucky and gracious to be born to the most attentive, considerate, and wise parents in the world. They instilled the value of perseverance and compassion early in my life through examples. Without their trust and encouragement, I would never dream of completing a doctorate degree halfway around the globe at one of the finest computer science institutions in the world. Furthermore, as lucky as a man could be, I met my dearest partner, Han, while pursuing my PhD. Since then, she has been my closest friend and most cherished counselor who has provided an immeasurable amount of love, strength, and comfort. I am able to become a better version of myself because of her. As much as this dissertation is written for the future of machines, it is the unwritten words about people that will be treasured for the rest of my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

It has been a long endeavour to augment human cognition with machine intelligence. As early as in 1945, Vannevar Bush envisioned a machine Memex that provides "enlarged intimate supplement to one's memory" and can be "consulted with exceeding speed and flexibility" in the seminal article *As We May Think* [14]. This vision has been brought closer to reality by years of research in computing hardware, artificial intelligence, and human-computer interaction. In late 90s to early 2000s, Smailagic et al. [102, 103, 104] created prototypes of wearable computers to assist cognitive tasks. For example, they displayed inspection manuals in a head-up screen to facilitate aircraft maintenance. Around the same time, Loomis et al. [63, 64] explored using computers carried in a backpack to provide auditory cues in order to help the blind navigate. Davis et al. [18, 23] developed a context-sensitive intelligent visitor guide leveraging hand-portable multimedia systems. While these research works pioneered cognitive assistance and its related fields, their robustness and functionality were limited by the supporting technologies of their time.

More recently, as the underlying technologies experience significant advancement, a new genre of applications, Wearable Cognitive Assistance (WCA) [16, 35], has emerged that pushes the boundaries of augmented cognition. WCA 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 [16] 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.

Since its conceptualization in 2004 [92], WCA has attracted much research interest from both academia and industry. The building blocks for its vision came into place by 2014, enabling the first implementation of this concept in *Gabriel* [35]. In 2017, Chen et al [17] described a number of applications of this genre, quantified their latency requirements, and profiled the end-to-end latencies of their implementations. In late 2017, SEMATECH and DARPA jointly funded $27.5 million of research on such applications [77, 108]. At the Mobile World Congress in February 2018, wearable cognitive assistance was the focus of an entire session [84]. For AI-based military use cases, this class of applications is the centerpiece of "Battlefield 2.0" [26]. By 2019, WCA

was being viewed as a prime source of "killer apps" for edge computing [93, 98].

Different from previous research efforts, the design goals of WCA 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 [96] 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 [16, 17, 35]. On top of Gabriel, several prototype applications have been built, such as PINGPONG Assistance, LEGO Assistance, COOKING Assistance, and IKEA LAMP Assembly Assistance. Using these applications as benchmarks, Chen et al. [17] presented 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 precludes 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 trials and errors. Development tools that alleviate the time and the expertise needed can greatly facilitate the creation of these applications.

1.1 Thesis Statement

In this dissertation, we address the problem of *scaling wearable cognitive assistance*. Scalability here has a two-fold meaning. First, a scalable system supports a large number of associated clients with fixed amount of infrastructure, and is able to serve more clients as resources increase. Second, we want to enable a small software team to quickly create, deploy, and manage these applications. 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.

We validate this thesis in this dissertation. The main contributions of the dissertation are as follows:

1. We propose application-agnostic and application-aware techniques to reduce bandwidth consumption and offered load when the cloudlet is oversubscribed.
2. We provide a profiling-based cloudlet resource allocation mechanism that takes account of diverse application adaptation characteristics.
3. We propose a new prototyping methodology and create a suite of development tools to reduce the time and lower the barrier of entry for WCA creation.

1.2 Thesis Overview

The remainder of this dissertation is organized as follows.

- In Chapter 2, we introduce prior work in wearable cognitive assistance.
- In Chapter 3, we describe and evaluate application-agnostic techniques to reduce bandwidth consumption when offloading computation.
- In Chapter 4, we propose and evaluate application-specific techniques to reduce offered load. We demonstrate their effectiveness with minimal impact on result latency.
- In Chapter 5, we present a resource management mechanisms that takes application adaptation characteristics into account to optimize system-wide metrics.
- In Chapter 6, we introduce a methodology and development tools for quick prototyping.
- In Chapter 7, we conclude this dissertation and discuss future directions.

Chapter 2

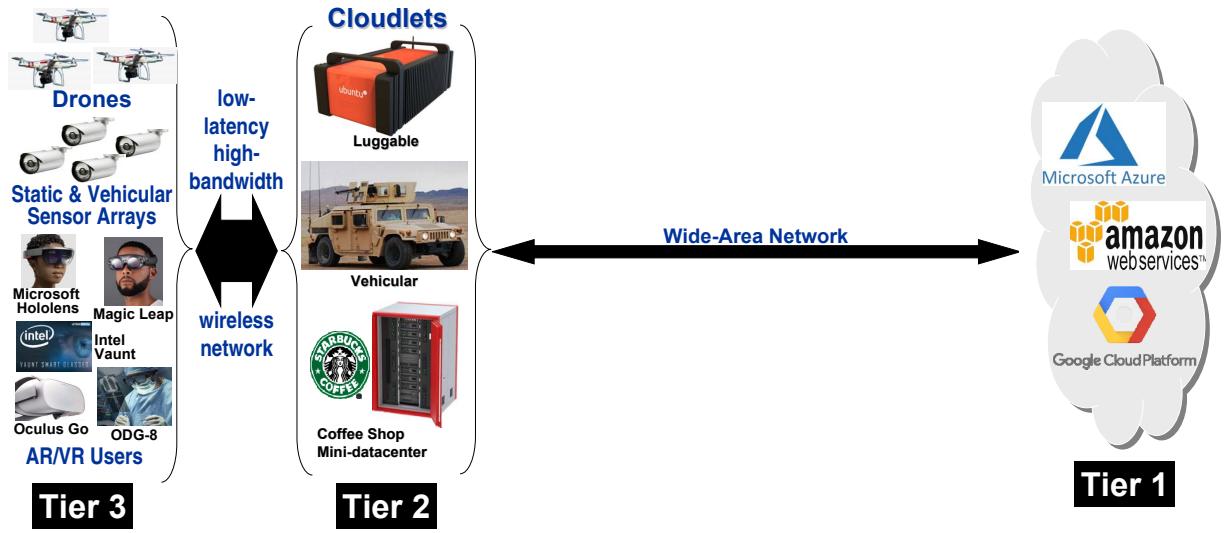
Background

2.1 Edge Computing

Edge computing is a nascent computing paradigm that has gained considerable traction over the past few years. It champions the idea of placing substantial compute and storage resources at the edge of the Internet, in close proximity to mobile devices or sensors. Terms such as "cloudlets" [95], "micro data centers (MDCs)" [7], "fog" [11], and "mobile edge computing (MEC)" [13] are used to refer to these small, edge-located computing nodes. We use these terms interchangably in the rest of this dissertation. Edge computing is motivated by its potential to improve latency, bandwidth, and scalability over a cloud-only model. More practically, some efforts stem from the drive towards software-defined networking (SDN) and network function virtualization (NFV), and the fact that the same hardware can provide SDN, NFV, and edge computing services. This suggests that infrastructure providing edge computing services may soon become ubiquitous, and may be deployed at greater densities than content delivery network (CDN) nodes today.

Satya et al. [97] best describes the modern computing landscape with edge computing using a tiered model, shown in Figure 2.1. Tiers are separated by distinct yet stable sets of design constraints. From left to right, this tiered model represents a hierarchy of increasing physical size, compute power, energy usage, and elasticity. Tier-1 represents today's large-scale and heavily consolidated data-centers. Compute elasticity and storage permanence are two dominating themes here. Tier-3 represents IoT and mobile devices, which are constrained by their physical size, weight, and heat dissipation. Sensing is the key functionality of Tier-3 devices. For example, today's smartphones are already rich in sensors, including camera, microphone, accelerometers, gyroscopes and GPS. In addition, an increasing amount of IoT devices with specific sensing modalities are getting adopted, e.g. smart speakers, security cameras, and smart thermostats.

With the large-scale deployment of Tier-3 devices, there exists a tension between the gigantic amount of data collected and generated by them and their limited capabilities to process these data on-board. For example, most surveillance cameras are limited in computation to run

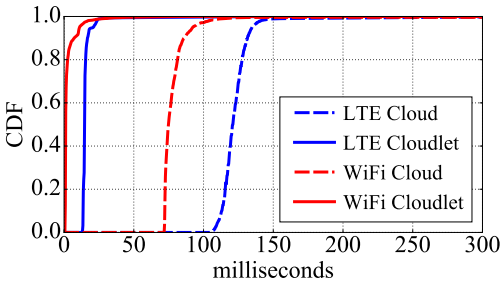


(Source: Satya et al [97])

Figure 2.1: Tiered Model of Computing

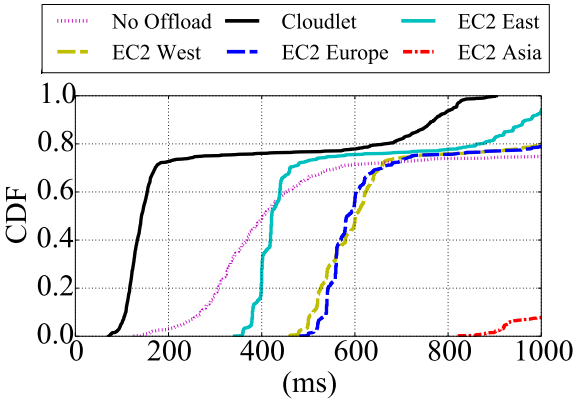
state-of-art computer vision algorithms to analyze the videos they capture. To overcome this tension, a Tier-3 device could offload computation over network to Tier-1. This capability was first demonstrated in 1997 by Noble et al. [75], who used it to implement speech recognition with acceptable performance on a resource-limited mobile device. In 1999, Flinn et al. [30] extended this approach to improve battery life. These concepts were generalized in a 2001 paper that introduced the term *cyber foraging* for the amplification of a mobile device's data or compute capabilities by leveraging nearby infrastructure [91]. Thanks to these research efforts, computation offloading is widely used by IoT devices today. For example, when a user asks an Amazon Echo smart speaker "Alexa, what is the weather today?", the user's audio stream is captured by the smart speaker and transmitted to the cloud for speech recognition, text understanding, and question answering.

However, offloading computation to the cloud has its own downside. Because of the consolidation needed to achieve the economy of scale, today's data centers are "far" from Tier-3 devices. The latency, throughput, and cost of wide-area network (WAN) significantly limit the amount of applications that can benefit from computation offloading. Even worse, it is the logical distance in the network that matters rather than the physical distance. Routing decisions in today's Internet are made locally and are based on business agreements, resulting in suboptimal solutions. For example, using traceroute, we determine that a LTE packet originating from a smartphone on the campus of Carnegie Mellon University (CMU) in Pittsburgh to a nearby server actually traverses to Philedelphia, a city several hundreds miles away. This is because Philidephia has the nearest peering point of the particular commercial LTE network in use to the public Internet. In 2010, Li et al. [59] report that the average round trip time (RTT) from 260 global vantage points to their optimal Amazon EC2 instances is 74 ms. In addition to long network delay, the high network fan-in of data centers means its aggregation network needs to carry significant amount of traffic. As the number of Tier-3 devices is expected to grow exponentially, these network links



(Source: Hu et al [44])

Figure 2.2: CDF of pinging RTTs



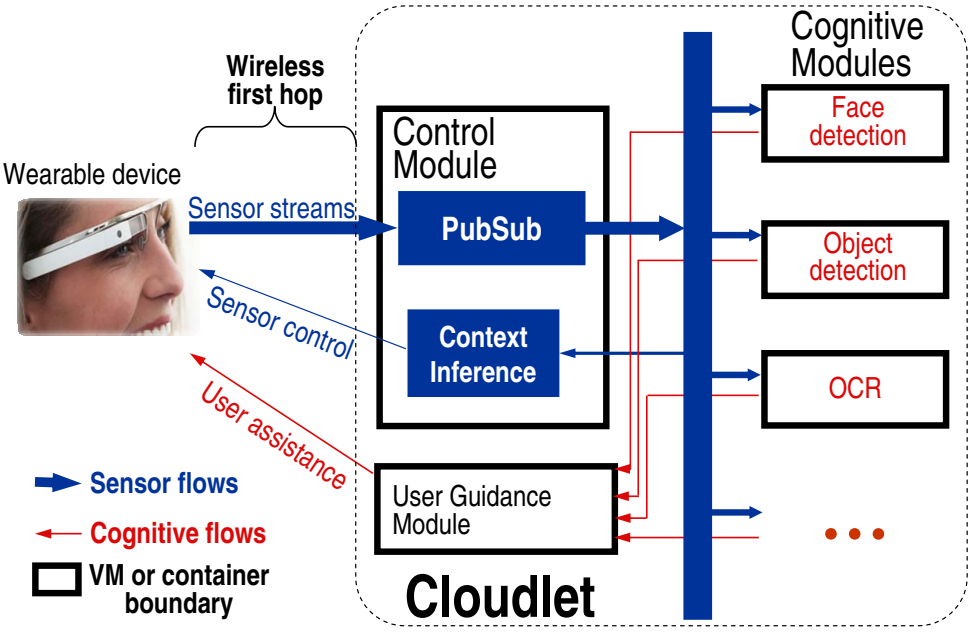
(Source: Hu et al [44])

Figure 2.3: FACE Response Time over LTE

face significant challenges to handle the ever-increasing volume of ingress traffic.

To counter these problems, edge computing, shown as the Tier-2 in Figure 2.1, is proposed. Cloudlets at Tier-2 creates the illusion of bringing Tier-1 "closer". They are featured by their network proximity to Tier-3 devices and their significantly larger compute and storage compared to Tier-3 devices. While Tier-3 devices typically run on battery and are optimized for low energy consumption, saving energy is not a major concern for Tier-2 as they are either plugged into the electric grid or powered by other sources of energy (e.g. fuels in a vehicle). Cloudlets serve two purposes in the tiered model. First, they provide infrastructure for compute-intensive and latency-sensitive applications for Tier-3. Wearable cognitive assistance is an examplar of these applications. Second, by processing data closer to the source of the content, it reduces the excessive traffic going into Tier-1 data centers. Figure 2.2 shows the RTT comparison of PING to the cloud and the cloudlet over WiFi and LTE. Cloudlet's RTT is on average 80 to 100ms shorter than its counterpart to the cloud. Figure 2.3 shows the impact of network latency on an application that recognizes faces. Three offloading scenarios are considered: offloading to the cloud, offloading to the cloudlet, and no offloading. The data transmitted are images captured by a smartphone. As we can see, the limited bandwidth of the cellular network further worsen the response time when offloading to the cloud. In fact, for this particular application, even local execution outperforms offloading to the nearest cloud due to network delay. The optimal computational offload location is cloudlet, whose median response time is more than 200ms faster than local execution and about 250 ms faster than the nearest cloud.

The low-latency and high-bandwidth compute infrastructure provided by cloudlets is an indispensible foundation for latency-sensitive and compute-intensive wearable cognitive assistance. Cloudlets also pose unique challenges for scalability as resources are a lot more limited compared to data centers. How to scale WCAs to many users using cloudlets is a key question we set out to investigate in this dissertation.



(Source: Chen et al [17])

Figure 2.4: Gabriel Platform

2.2 Gabriel Platform

The Gabriel platform [17, 35], shown in Figure 2.4, is the first application framework for wearable cognitive assistance. It consists of a front-end running on wearable devices and a back-end running on cloudlets. The Gabriel front-end performs preprocessing of sensor data (e.g., compression and encoding), which it then streams over a wireless network to a cloudlet. The Gabriel back-end on the cloudlet has a modular structure. The *control module* is the focal point for all interactions with the wearable device and can be thought as an agent for a particular client on the cloudlet. A publish-subscribe (PubSub) mechanism distributes the incoming sensor streams to multiple *cognitive modules* (e.g., task-specific computer vision algorithms) for concurrent processing. Cognitive module outputs are integrated by a task-specific *user guidance module* that performs higher-level cognitive processing such as inferring task state, detecting errors, and generating guidance in one or more modalities (e.g., audio, video, text, etc.). The Gabriel platform automatically discovers cognitive engines on the local network via a universal plug-and-play (UPnP) protocol. The platform is designed to run on a small cluster of machines with each module capable of being separated or co-located with other modules via process, container, or virtual machine virtualization.

The original Gabriel platform was built with a single user in mind, and did not have mechanisms to share cloudlet resources in a controlled manner. It did, however, have a token-based transmission mechanism. This limited a client to only a small number of outstanding operations, thereby offering a simple form of rate adaptation to processing or network bottlenecks. We have retained this token mechanism in our system, described in the rest of this dissertation.

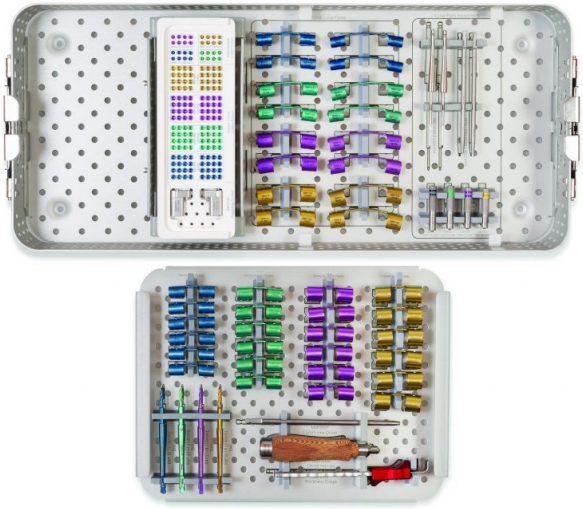


Figure 2.5: RibLoc Kit

In addition, we have extended Gabriel with new mechanisms to handle multi-tenancy, perform resource allocation, and support application-aware adaptation. We refer to the two versions of the platform as "Original Gabriel" and "Scalable Gabriel."

2.3 Example Gabriel Applications

Many applications have been built on top of the Gabriel platform. Table 2.1 provides a summary of applications built by Chen et al [16]. These applications run on multiple wearable devices such as Google Glass, Microsoft HoloLens, Vuzix Glass, and ODG R7. The cloudlet processing of these applications consists of two major phases. The first phase uses computer vision to extract a symbolic, idealized representation of the state of the task, accounting for real-world variations in lighting, viewpoint, etc. The second phase operates on the symbolic representation, implements the logic of the task at hand, and occasionally generates guidance for the user. In most WCA applications, the first phase is far more compute intensive than the second phase. While visual data is the focus of the analysis, other types of sensor data, e.g. audio, are also used to help infer user states.

Building on lessons learned by Chen et al [16], we create and implement several real-life WCA applications whose tasks are more complex. They also pose more challenges to the computer vision processing as the parts are not designed for machine recognition. In particular, we focus on assembly tasks. Many assembly tasks, including manufacturing assembly and medical procedures, are tedious and error-prone. WCAs can help reduce errors, provide training, and assist human operators by continuously monitoring their actions and offering feedback. We describe two of these applications below in detail.

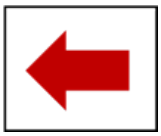
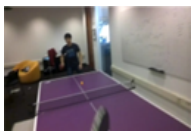
App Name	Example Input Video Frame	Description	Symbolic Representation	Example Guidance
Pool		Helps a novice pool player aim correctly. Gives continuous visual feedback (left arrow, right arrow, or thumbs up) as the user turns his cue stick. Correct shot angle is calculated based on fractional aiming system. Color, line, contour, and shape detection are used. The symbolic representation describes the positions of the balls, target pocket, and the top and bottom of cue stick.	<Pocket, object ball, cue ball, cue top, cue bottom>	
Ping-pong		Tells novice to hit ball to the left or right, depending on which is more likely to beat opponent. Uses color, line and optical-flow based motion detection to detect ball, table, and opponent. The symbolic representation is a 3-tuple: in rally or not, opponent position, ball position. Whispers "left" or "right" or offers spatial audio guidance using. Video URL: *https://youtu.be/_lp32sowyUA*	<InRally, ball position, opponent position>	Whispers "Left!"
Workout		Guides correct user form in exercise actions like sit-ups and push-ups, and counts out repetitions. Uses Volumetric Template Matching on a 10-15 frame video segment to classify the exercise. Uses smart phone on the floor for third-person viewpoint.	<Action, count>	Says "8 "
Face		Jogs your memory on a familiar face whose name you cannot recall. Detects and extracts a tightly-cropped image of each face, and then applies a state-of-art face recognizer using deep residual network. Whispers the name of a person. Can be used in combination with Expression to offer conversational hints.	ASCII text of name	Whispers "Barack Obama"
Lego		Guides a user in assembling 2D Lego models. Each video frame is analyzed in three steps: (i) finding the board using its distinctive color and black dot pattern; (ii) locating the Lego bricks on the board using edge and color detection; (iii) assigning brick color using weighted majority voting within each block. Color normalization is needed. The symbolic representation is a matrix representing color for each brick. Video URL: *https://youtu.be/7L9U-n29abg*	[[0, 2, 1, 1], [0, 2, 1, 6], [2, 2, 2, 2]]	 Says "Put a 1x3 green piece on top"
Draw		Helps a user to sketch better. Builds on third-party app that was originally designed to take input sketches from pen-tablets and to output feedback on a desktop screen. Our implementation preserves the back-end logic. A new Glass-based front-end allows a user to use any drawing surface and instrument. Displays the error alignment in sketch on Glass. Video URL: *https://youtu.be/nuQpPtVJC6o*		
Sandwich		Helps a cooking novice prepare sandwiches according to a recipe. Since real food is perishable, we use a food toy with plastic ingredients. Object detection follows the state-of-art faster-RCNN deep neural net approach. Implementation is on top of Caffe and Dlib. Transfer learning helped us save time in labeling data and in training. Video URL: *https://youtu.be/USakPP45WvM*	Object: "E.g. Lettuce on top of ham and bread"	 Says "Put a piece of bread on the lettuce"

Table 2.1: Prototype Wearable Cognitive Assistance Applications

(Adapted from Chen et al [16])

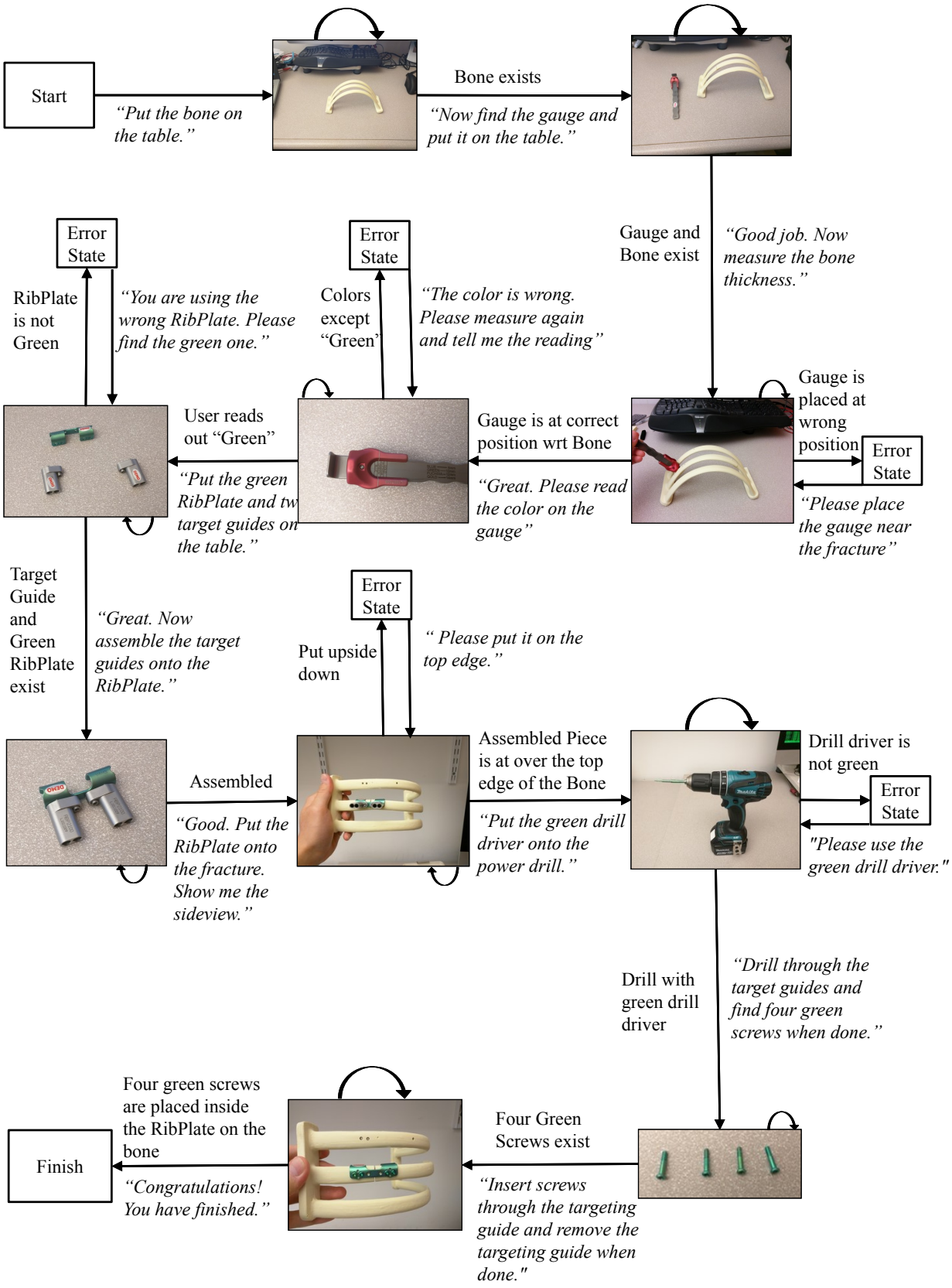


Figure 2.6: RibLoc Assistant Workflow

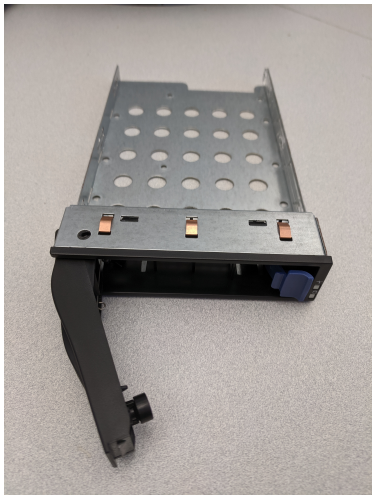


Figure 2.7: Assembled DiskTray

2.3.1 RibLoc Application

RibLoc Fracture Plating System [3], shown in Figure 2.5 is used by surgeons to stabilize broken ribs for fracture repair. The surgery consists of five major steps: measure ribs thickness, prepare the plate, drill bone for screw insertion, insert screws, and tighten screws. To train doctors how to use this kit, Acute Innovations, the manufacturer of RibLoc, currently send sales personnel to doctors' office, often for extended period of time. To reduce the cost of training, we develop a WCA for RibLoc that guides a surgeon to learn RibLoc step-by-step on fake bones. Figure 2.6 shows the task steps of the application as a finite state machine (FSM). Conditions of state transitions are shown above the transition arrows and instructions given to users are quoted in italics. Most of user state recognition is achieved by verifying the existence and locations of key objects. One exception is rib thickness measurement. The application asks the user to read out some text from the gauge. Automatic speech recognition (ASR) is used to detect the user's read-out. ASR is used because the text on the gauge has a low contrast with the background that they are too hard to optical character recognition (OCR). A demo video of RibLoc WCA is at `https://youtu.be/YRTXUty2P1U`.

2.3.2 DiskTray Application

In collaboration with InWin [2], a computer hardware manufacturer, we created a cognitive assistant to train operators how to assemble their disk tray product. Figure 2.7 shows the assembled tray, which is used to host hard drives that go into a server chassis. As with all the other WCAs, we do not modify the original parts.

We face two challenges when building this application. First, some parts are tiny. In one step, the application needs to check if a pin is placed correctly into a slot. As shown in Figure 2.8, both the slot and the pin are tiny and hard to see. To facilitate the detection of these two parts, we ask the user to bring the parts close to the camera in addition to zooming the camera and turning on

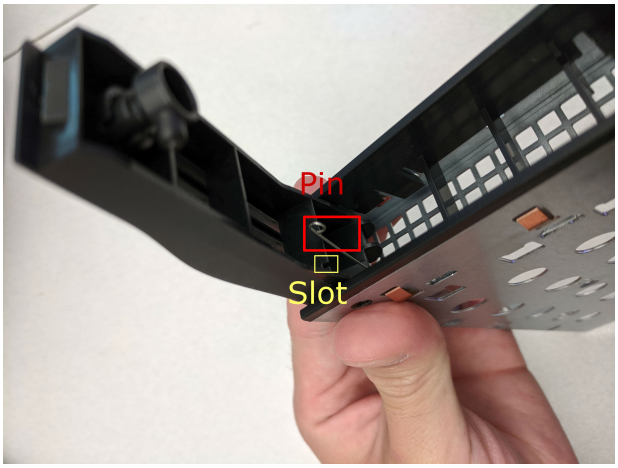


Figure 2.8: Small Parts in DiskTray WCA

	POOL	WORK OUT	PING PONG	FACE	Assembly Tasks (e.g. RibLoc)
Bound Range (tight-loose)	95-105	300-500	150-230	370-1000	600-2700

Table 2.2: Application Latency Bounds (in milliseconds)

(Adapted from Chen et al [17])

the camera flashlight. Second, there is a plastic strip that is translucent. Transparent objects are extremely challenging to detect using 2D RGB cameras, because their appearance depends on the background and lighting [68]. Instead of detecting the placement of the transparent strip by computer vision, we leverage the human operator to signal to the system when the operation is completed. InWin showcased this application at 2018 Computex Technology Show [1]. A demo video of the Computex Demo can be viewed at `https://www.youtube.com/watch?v=AwWZcL9XGI0`.

2.4 Application Latency Bounds

Not only are the accuracy of the instructions important to WCAs, but the speed at which these instructions are delivered is also critical. With a human in the loop, the latency requirements of WCAs are closely related to the task-specific human speed. For example, for assembly tasks, an instruction delivered tens of seconds after a user has finished a step can cause user annoyance and frustration. For PING PONG assistant, a task that is even more fast-paced, an instruction on where to hit the ball becomes useless if it is delivered after the user has hit the ball.

Previous work [17] quantifies task-dependent application latency bounds by answering the question *How fast does an instruction need to be delivered?* Three different approaches are used. For tasks that have published human speed, numbers from the literature are used as the upper bound of the end-to-end system response time. For example, it takes about 1000 ms for a

human to recognize the identity of a face. Therefore, a face recognition assistant should deliver a person's name faster than 1000ms to exceed human speed. For tasks in which users interact with physical systems, the latency bounds can be derived directly from first principles of physics. For instance, the latency bounds for PINGPONG assistant are calculated by subtracting audio perception time, motion initiation time, and swinging time from the average time between opponents hitting the ball. In addition, an user study of LEGO assembly assistant is also conducted to deduct latency bounds for assembly tasks.

Table 2.2 shows a summary of latency bounds calculated from the previous work [17]. Each application is assigned both a tight and a loose bound from the application perspective. The tight bound represents an ideal target, below which the user is insensitive to improvements. Above the loose bound, the user becomes aware of slowness, and user experience and performance is significantly impacted. Latency improvements between the two limits may be useful in reducing user fatigue.

These application latency bounds can be considered as application quality of service (QoS) metrics. Similar to bitrate and startup time in video streaming, these metrics serve as measurable proxies to user experience. When the system delay increases, we can compare the delay with these latency bounds to estimate how much a user is suffering. In this dissertation, we use these bounds to formulate application utility functions, which quantify user experience when a system response is delayed due to contention. These application utility functions are the foundation for our adaptation-centered approach to scalability.

Chapter 3

Application-Agnostic Techniques to Reduce Network Transmission

WCAs continuously stream sensor data to a cloudlet. The richer a sensing modality is, the more information can be extracted. The core sensing modality of WCAs is visual data, e.g. egocentric images and videos from wearable cameras. Compared to other sensors, e.g. microphones and inertial measurement units (IMUs), cameras capture visual information with rich semantics. As commercial camera hardware become more affordable, they have become increasingly pervasive. In 2013, it is estimated that there is one security camera for every eleven citizens in the UK [10]. In the meantime, deep neural networks (DNNs) have driven significant advancement in computer vision and have achieved human-level accuracy in several previously intractable perception problems (e.g. face recognition, image classification) [57, 99]. The richness and the open-endness of visual data makes camera the ideal sensor for WCAs.

However, continuous video transmission from many Tier-3 devices places severe stress on the wireless spectrum. Hulu estimates that its video streams require 13 Mbps for 4K resolution and 6 Mbps for HD resolution using highly optimized offline encoding [46]. Live streaming is less bandwidth-efficient, as confirmed by our measured bandwidth of 10 Mbps for HD feed at 25 FPS. Just 50 users transmitting HD video streams continuously can saturate the theoretical uplink capacity of 500 Mbps in a 4G LTE cell that covers a large rural area [67]. This is clearly not scalable.

In this chapter, we show how per-user bandwidth demand in WCA-like live video analytics can be significantly reduced using an application-agnostic approach. We aim to reduce bandwidth demand without compromising the timeliness or accuracy of results. In contrast to previous works [116, 117, 123], we leverage state-of-the-art deep neural networks (DNNs) to selectively transmit interesting data from a video stream and explore environment-specific optimizations. The accuracy of the data selection is important, as fewer false positives result in lower network bandwidth and cloudlet computing cycle consumption.

This chapter is organized as follows. We first discuss the challenges of running DNNs for visual perception solely on Tier-3 devices in Section 3.1. Next, we propose and compare two

application-agnostic techniques to reduce network transmission. We present our results first in the context of live video analytics for small autonomous drones. Both as emerging Tier-3 devices, drones and wearable devices face similar challenges in live video analysis. Finally, we showcase how these techniques can be applied to WCAs in Section 3.5.

3.1 Video Processing on Mobile Devices

In the context of real-time video analytics, Tier-3 devices represent fundamental mobile computing challenges that were identified two decades ago [90]. Two challenges have specific relevance here. First, mobile elements are resource-poor relative to static elements. Second, mobile connectivity is highly variable in performance and reliability. We discuss their implications below.

3.1.1 Computation Power on Tier-3 Devices

Unfortunately, the hardware needed for deep video stream processing in real time is larger and heavier than what can fit on a typical Tier-3 device. State-of-art techniques in image processing use DNNs that are compute- and memory-intensive. Table 3.1 presents experimental results on two fundamental computer vision tasks, image classification and object detection, on four different devices. In the figure, MobileNet V1 and ResNet101 V1 are image classification DNNs. Others are object detection DNNs. We use publicly available pretrained DNN models for measurements [109]. We carefully choose hardware platforms to represent a range of computation capabilities of Tier-3 devices. To anticipate future improvements in smartphone technology, our experiments also consider more powerful devices such as the Intel® Joule [37] and the NVIDIA Jetson [76] that are physically compact and light enough to be credible as wearable device platforms in the future.

Note that the absolute speed of DNN inference depends on a wide range of factors, including the optimizations used in DNN model implementation, the choice of linear algebra libraries, the presence of vectorized instructions and hardware accelerators. In Table 3.1, we present the best results we could obtain on each platform. This is not intended to directly compare frameworks and platforms (as others have been doing [124]), but rather to illustrate the differences between wearable platforms and fixed infrastructure servers.

Image classification is a computer vision task that maps an image into categories, with each category indicating whether one or many particular objects (e.g., a human survivor, a specific animal, or a car) exist in the image. Two widely used multiclass classification DNNs are tested. Their prediction speed are shown in column "Image Classification". One of the DNNs, MobileNet V1 [41], is designed for mobile devices from the ground-up by reducing the number of parameters and simplifying the layer computation using depth-wise separable convolution. On the other hand, ResNet101 V1 [39] is a more accurate but also more resource-hungry DNN that won the highly recognized ImageNet classification challenge in 2015 [88].

Also shown is another harder computer vision task, object detection. Object detection is a

M: MobileNet V1; R: ResNet101 V1; S-M: SSD MobileNet V1; S-I: SSD Inception V2;
F-I: Faster R-CNN Inception V2; F-R: Faster R-CNN ResNet101 V1

	Weight (g)	CPU	GPU	Image Classification		Object Detection			
				M (ms)	R (ms)	S-M (ms)	S-I (ms)	F-I (ms)	F-R (ms)
Nexus 6	184	4-core 2.7 GHZ Krait 450, 3GB Mem	Adreno 420	353 (67)	983 (141)	441 (60)	794 (44)	ENOMEM	ENOMEM
Intel® Joule 570x	25	4-core 1.7 GHz Intel Atom® T5700, 4GB Mem	Intel® HD Graphics (gen 9)	37 (1)‡	183 (2)†‡	73 (2)‡	442 (29)	5125 (750)	9810 (1100)
NVIDIA Jetson TX2	85	2-Core 2.0 GHz Denver2 + 4-Core 2.0 GHz Cortex-A57, 8GB Mem	256 cuda core 1.3 GHz NVIDIA Pascal	13 (0)†	92 (2)†	192 (18)	285 (7)†	ENOMEM	ENOMEM
Rack-mounted Server		2x 36-core 2.3 GHz Intel® Xeon® E5-2699v3 Processors, 128GB Mem	2880 cuda core 875MHz NVIDIA Tesla K40, 12GB GPU Mem	4 (0)‡	33 (0)†	12 (2)‡	70 (6)	229 (4)†	438 (5)†

Figures above are means of 3 runs across 100 random images. The time shown includes only the forward pass time using batch size of 1. ENOMEM indicates failure due to insufficient memory. Figures in parentheses are standard deviations. The weight figures for Joule and Jetson include only the modules without breakout boards. Weight for Nexus 6 includes the complete phone with battery and screen. Numbers are obtained with TensorFlow (TensorFlow Lite for Nexus 6) unless indicated otherwise.
† indicates GPU is used. ‡ indicates Intel® Computer Vision SDK beta 3 is used.

Table 3.1: Deep Neural Network Inference Speed on Tier-3 Devices

more difficult perception problem, because it not only requires categorization, but also prediction of bounding boxes around the specific areas of an image that contains a object. Object detection DNNs are built on top of image classification DNNs by using image classification DNNs as low-level feature extractors. Since feature extractors in object detection DNNs can be changed, the DNN structures excluding feature detectors are referred as object detection meta-architectures. We benchmarked two object detection DNN meta-architectures: Single Shot Multibox Detector (SSD) [62] and Faster R-CNN [85]. We used multiple feature extractors for each meta-architecture. The meta-architecture SSD uses simpler methods to identify potential regions for objects and therefore requires less computation and runs faster. On the other hand, Faster R-CNN [85] uses a separate region proposal neural network to predict regions of interest and has been shown to achieve higher accuracy [45] for difficult tasks. Table 3.1 presents results in four columns: SSD combined with MobileNet V1 or Inception V2, and Faster R-CNN combined with Inception V2 or ResNet101 V1 [39]. The combination of Faster R-CNN and ResNet101 V1 is one of the most accurate object detectors available today [88]. The entries marked "ENOMEM" correspond to experiments that were aborted because of insufficient memory.

These results demonstrates the computation gap between mobile and static elements. While the most accurate object detection model Faster R-CNN Resnet101 V1 can achieve more than two FPS on a server GPU, it either takes several seconds on Tier-3 devices or fails to execute due to insufficient memory. In addition, the figure also confirms that sustaining open-ended real-time video analytics on smartphone form factor computing devices is well beyond the state of the art today and may remain so in the near future. This constrains what is achievable with Tier-3 devices alone.

3.1.2 Result Latency, Offloading and Scalability

Result latency is the delay between first capture of a video frame in which a particular result is present, and report of its discovery or feedback based on the discovery after video processing. Operating totally disconnected, a Tier-3 device can capture and store video, but defer its processing until the mission is complete. At that point, the data can be uploaded from the device to the cloud and processed there. This approach completely eliminates the need for real-time video processing, obviating the challenges of Tier-3 computation power mentioned previously. Unfortunately, this approach delays the discovery and use of knowledge in the captured data by a substantial amount (e.g., many tens of minutes to a few hours). Such delay may be unacceptable in use cases such as search-and-rescue using drones, or real-time step-by-step instruction feedback in WCAs. In this chapter, we focus on approaches that aim for much smaller, real-time result latency.

A different approach is to offload video processing in real-time over a wireless link to an edge computing node. With this approach, even a weak Tier-3 device can leverage the substantial processing capability of a cloudlet, without concern for its weight, size, heat dissipation or energy usage. Much lower result latency is now possible. However even if cloudlet resources are viewed as "free" from the viewpoint of mobile computing, the Tier-3 device consumes wireless bandwidth in transmitting video.

Today, 4G LTE offers the most plausible wide-area connectivity from a Tier-3 device to its associated cloudlet. The much higher bandwidths of 5G are still several years away, especially at global scale. More specialized wireless technologies, such as Lightbridge 2 [25], could also be used. Regardless of specific wireless technology, the principles and techniques described in this chapter apply.

When offloading, scalability in terms of maximum number of concurrently operating Tier-3 devices within a 4G LTE cell becomes an important metric. In this chapter, we explore how the limited processing capability on a Tier-3 device can be used to greatly decrease the volume of data transmitted, thus improving scalability while minimally impacting result accuracy and result latency.

Note that the uplink capacity of 500 Mbps per 4G LTE cell assumes standard cellular infrastructure that is undamaged. In natural disasters and military combat, this infrastructure may be destroyed. Emergency substitute infrastructure, such as Google and AT&T's partnership on balloon-based 4G LTE infrastructure for Puerto Rico after hurricane Maria [70], can only sustain much lower uplink bandwidth per cell, e.g. 10Mbps for the balloon-based LTE [89]. Conserving wireless bandwidth from Tier-3 video transmission then becomes even more important, and the techniques described here will be even more valuable.

3.2 Baseline Strategy

3.2.1 Description

We first establish and evaluate the baseline case of no image processing performed at the Tier-3 device. Instead, all captured video is immediately transmitted to the cloudlet. Result latency is very low, merely the sum of transmission delay and cloudlet processing delay. We use drones as the example of Tier-3 devices and drone video search as the scenario of video analytics first. We later demonstrate how to apply the techniques developed to WCAs.

3.2.2 Experimental Setup

To ensure experimental reproducibility, our evaluation is based on replay of a benchmark suite of pre-captured videos rather than on measurements from live drone flights. In practice, live results may diverge slightly from trace replay because of non-reproducible phenomena. These can arise, for example, from wireless propagation effects caused by varying weather conditions, or by seasonal changes in the environment such as the presence or absence of leaves on trees. In addition, variability can arise in a drone's pre-programmed flight path due to collision avoidance with moving obstacles such as birds, other drones, or aircraft.

All of the pre-captured videos in the benchmark suite are publicly accessible, and have been captured from aerial viewpoints. They characterize drone-relevant scenarios such as surveillance, search-and-rescue, and wildlife conservation. Table 3.2 presents this benchmark suite of videos,

Task	Detection Goal	Data Source	Data Attributes	Training Subset	Testing Subset
T1	People in scenes of daily life	Okutama Action Dataset [9]	33 videos 59842 fr 4K@30 fps	9 videos 17763 fr	6 videos 20751 fr
T2	Moving cars	Stanford Drone Dataset [87]	60 videos 522497 fr 1080p@30 fps	16 videos 179992 fr	14 videos 92378 fr Combination of test videos from each dataset.
T3	Raft in flooding scene	YouTube collection [4]	11 videos 54395 fr 720p@25 fps	8 videos 43017 fr	
T4	Elephants in natural habitat	YouTube collection [5]	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

Table 3.2: Benchmark Suite of Drone Video Traces

Task	Total Bytes (MB)	Avg BW (Mbps)	Recall	Precision
T1	924	10.7	74%	92%
T2	2704	7.0	66%	90%

Peak bandwidth demand is same as average since video is transmitted continuously. Precision and recall are at the maximum F1 score.

Table 3.3: Baseline Object Detection Metrics

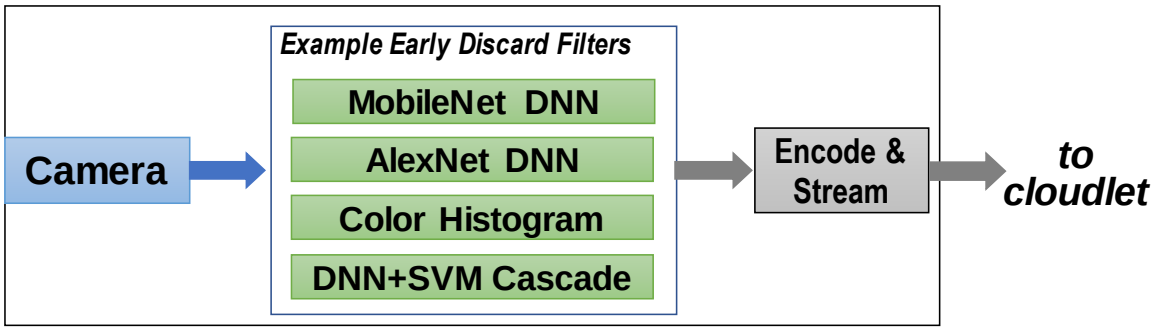


Figure 3.1: Early Discard on Tier-3 Devices

organized into four tasks. All the tasks involve detection of tiny objects on individual frames. Although T2 is also nominally about action detection (moving cars), it is implemented using object detection on individual frames and then comparing the pixel coordinates of vehicles in successive frames.

3.2.3 Evaluation

Table 3.3 presents the key performance indicators on the object detection tasks T1 and T2. We use the well-labeled dataset to train and evaluate Faster-RCNN with ResNet 101. We report the precision and recall at maximum F1 score. Peak bandwidth is not shown since it is identical to average bandwidth demand for continuous video transmission. As shown earlier in Table 3.1, the accuracy of this algorithm comes at the price of very high resource demand. This can only be met today by server-class hardware that is available in a cloudlet. Even on a cloudlet, the figure of 438 milliseconds of processing time per frame indicates that only a rate of two frames per second is achievable. Sustaining a higher frame rate will require striping the frames across cloudlet resources, thereby increasing resource demand considerably. Note that the results in Table 3.1 were based on 1080p frames, while tasks T1 uses the higher resolution of 4K. This will further increase demand on cloudlet resources.

Clearly, the strategy of blindly shipping all video to the cloudlet and processing every frame is resource-intensive to the point of being impractical today. It may be acceptable as an offline processing approach in the cloud, but is unrealistic for real-time processing on cloudlets. We therefore explore an approach in which a modest amount of computation on the Tier-3 is able, with high confidence, to avoid transmitting many video frames and thereby saving wireless bandwidth as well as cloudlet processing resources. This leads us to the EarlyDiscard strategy of the next section.

3.3 EarlyDiscard Strategy

3.3.1 Description

EarlyDiscard is based on the idea of using on-board processing to filter and transmit only interesting frames in order to save bandwidth when offloading computation. Frames are considered to be interesting if they capture objects or events valuable for processing, for instance, survivors for a search task. Previous work [43, 71] leveraged pixel-level features and multiple sensing modalities to select interesting frames from hand-held or body-worn cameras. In this section, we explore the use of DNNs to filter frames from aerial views. The benefits of using DNNs are as follows. First, DNNs, even shallow ones, are capable of understanding some semantically meaningful visual information. Their decisions of what to send are based on the reasoning of image content in addition to pixel-level characteristics. Next, DNNs are trained and specialized for each task, resulting in their high accuracy and robustness for that particular task. Finally, compared to a sensor fusing approach that requires other sensing modalities to be present on Tier-3 devices, no additional hardware is added to the existing platforms.

Although smartphone-class hardware is incapable of supporting the most accurate object detection algorithms at full frame rate today, it is typically powerful enough to support less accurate algorithms. These *weak detectors*, for instance, MobileNet in Table 3.1, are typically designed for mobile platforms or were the state of the art just a few years ago. In addition, they can be biased towards high recall with only modest loss of precision. In other words, many clearly irrelevant frames can be discarded by a weak detector, without unacceptably increasing the number of relevant frames that are erroneously discarded. This asymmetry is the basis of the early discard strategy.

As shown in Figure 3.1, we envision a choice of weak detectors being available as early discard filters on Tier-3 devices with the specific choice of filter being task-specific. Based on the measurements presented in Table 3.1, we choose cheap DNNs that can run in real-time as EarlyDiscard filters on Tier-3 devices. Note that both object detection and image classification algorithms can yield meaningful early discard results, as it is not necessary to know exactly where in the frame relevant objects occur — just an estimate of key object presence is good enough. This suggests that MobileNet would be a good choice as a weak detector. For a given image or partial of an image, it can predict whether the input contains objects of interests. More importantly, MobileNet's speed of 13 ms per frame on the Tier-3 platform Jetson yields more than 75 fps. We therefore use MobileNet for early discard in our experiments.

Pre-trained classifiers for MobileNet are available today for generic objects such as cars, animals, human faces, human bodies, watercraft, and so on. However, these DNN classifiers have typically been trained on images that were captured from a human perspective — often by a camera held or worn by a person. These images typically have the objects at the center of the image and occupy the majority of the image. Many Tier-3 devices, however, capture images from different viewpoints (e.g. aerial views) and need to recognize rare task-specific objects different from generic categories. To improve the classification accuracy for custom objects from different viewpoints, we used *transfer learning* [120] to finetune the pre-trained classifiers

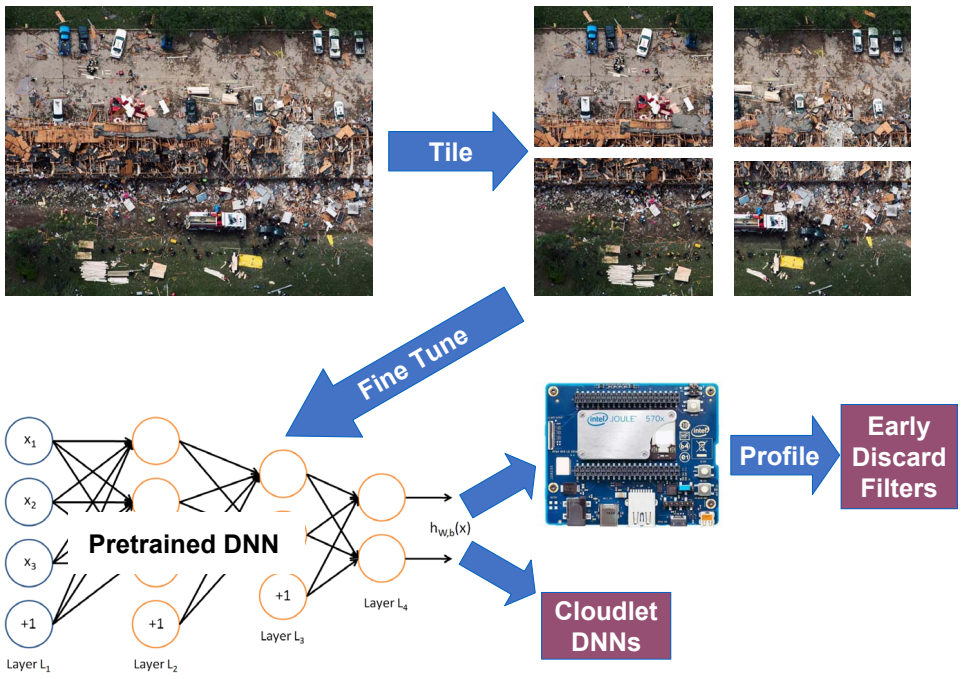


Figure 3.2: Tiling and DNN Fine Tuning

on small training sets of images that were captured from correct viewpoint. The process of fine-tuning involves initial re-training of the last DNN layer, followed by re-training of the entire network until convergence. Transfer learning enables accuracy to be improved significantly for custom objects without incurring the full cost of creating a large training set.

For live drone video analytics, images are typically captured from a significant height, and hence objects in such an image are small. This interacts negatively with the design of many DNNs, which first transform an input image to a fixed low resolution — for example, 224x224 pixels in MobileNet. Many important but small objects in the original image become less recognizable. It has been shown that small object size correlates with poor accuracy in DNNs [45]. To address this problem, we *tile* high resolution frames into multiple sub-frames and then perform recognition on the sub-frames as a batch. This is done offline for training, as shown in Figure 3.2, and also for online inference on the drone and on the cloudlet. The lowering of resolution of a sub-frame by a DNN is less harmful, since the scaling factor is smaller. Objects are represented by many more pixels in a transformed sub-frame than if the entire frame had been transformed. The price paid for tiling is increased computational demand. For example, tiling a frame into four sub-frames results in four times the classification workload. Note that this increase in workload typically does not translates into the same increase in inference time, as workloads can be batched together to leverage hardware parallelism for a reduced total inference time.

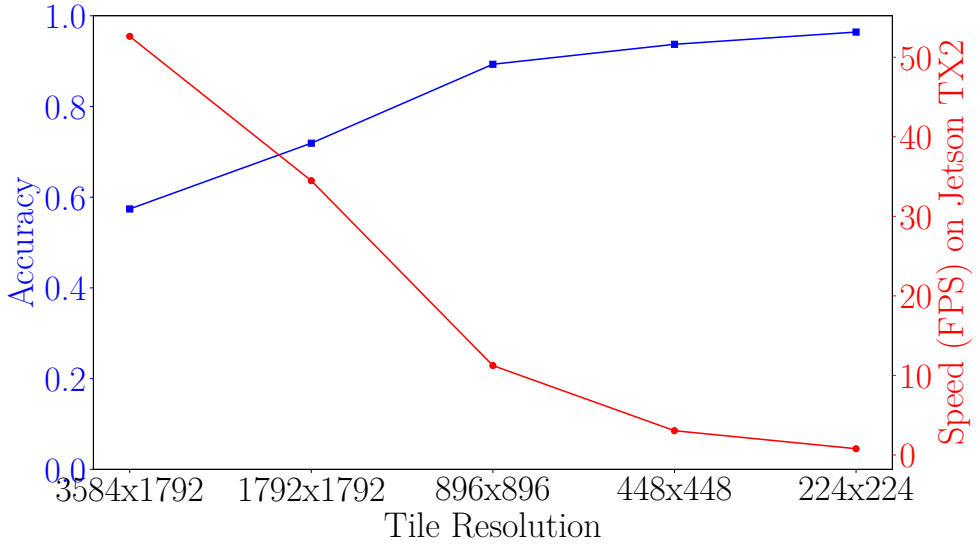


Figure 3.3: Speed-Accuracy Trade-off of Tiling

3.3.2 Experimental Setup

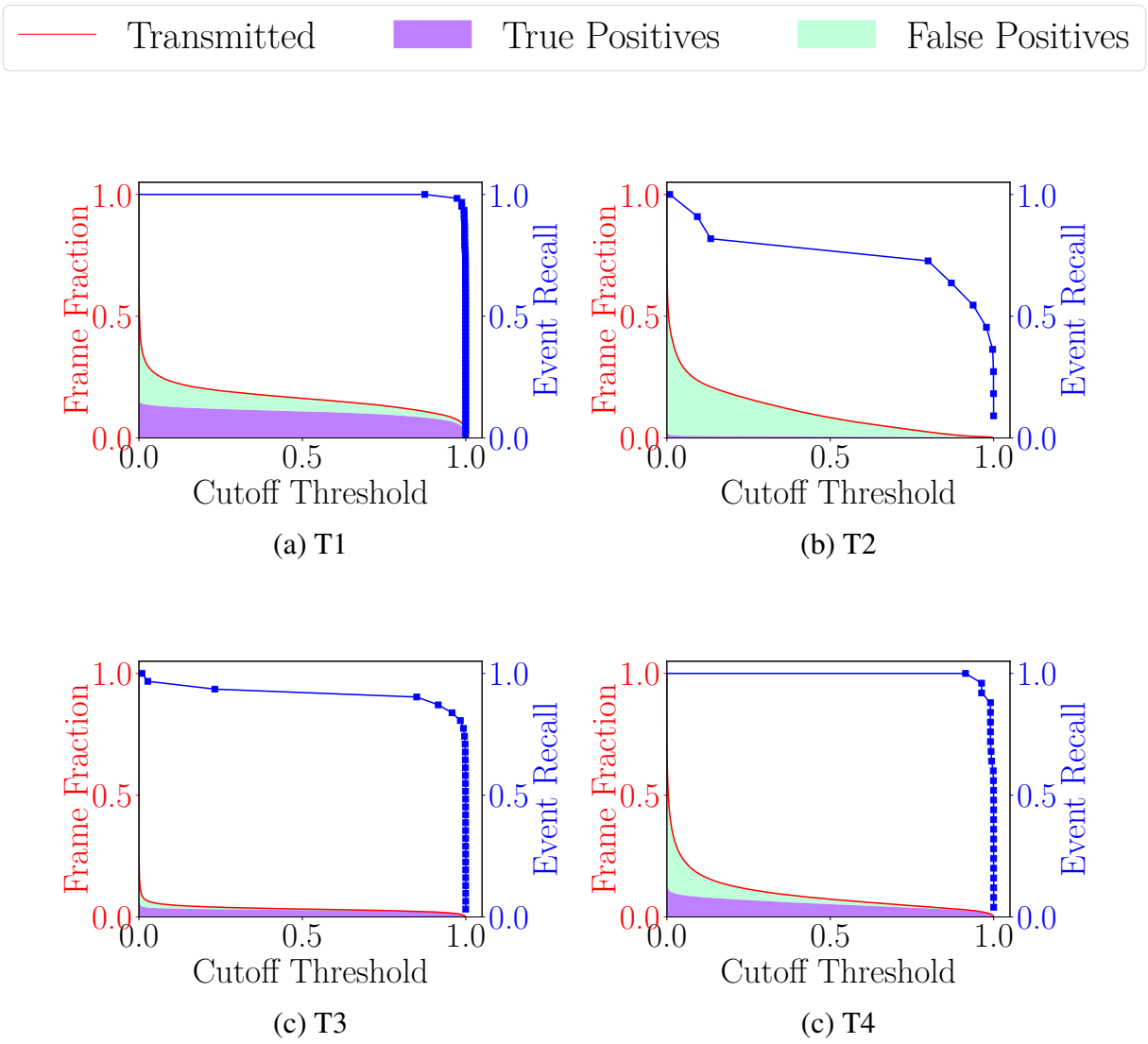
Our experiments on the EarlyDiscard strategy used the same benchmark suite described in Section 3.2.2. We used Jetson TX2 as the Tier-3 device platform. We run MobileNet filters to get predictions on whether sub-frames contain objects of interests. We compare the predictions with ground truths (e.g. whether a sub-frame is indeed interesting) to evaluate the effectiveness of EarlyDiscard. Both frame-based and event-based metrics are used in the evaluation.

3.3.3 Evaluation

EarlyDiscard is able to significantly reduce the bandwidth consumed while maintaining high result accuracy and low average delay. For three out of four tasks, the average bandwidth is reduced by a factor of ten. Below we present our results in detail.

Effects of Tiling

Tiling is used to improve the accuracy for high resolution aerial images. We used the Okutama Action Dataset, whose attributes are shown in row T1 of Table 3.2, to explore the effects of tiling. For this dataset, Figure 3.3 shows how speed and accuracy change with tile size. Accuracy improves as tiles become smaller, but the sustainable frame rate drops. We group all tiles from the same frame in a single batch to leverage parallelism, so the processing does not change linearly with the number of tiles. The choice of an operating point will need to strike a balance between the speed and accuracy. In the rest of the chapter, we use two tiles per frame by default.



(a) T1

(b) T2

(c) T3

(c) T4

Figure 3.4: Bandwidth Breakdown

	Task 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

Table 3.4: Recall, Event Latency and Bandwidth at Cutoff Threshold 0.5

EarlyDiscard Filter Accuracy

The output of a Tier-3 filter is the probability of the current tile being "interesting." A tunable *cutoff threshold* parameter specifies the threshold for transmission to the cloudlet. All tiles, whether deemed interesting or not, are still stored in the Tier-3 storage for offline processing.

Since objects have temporal locality in videos, we define an event (of an object) in a video to be consecutive frames containing the same object of interests. For example, the appearance of the same red raft in T3 in consecutive 45 frames constitutes a single event. A correct detection of an event is defined as at least one of the consecutive frames being transmitted to the cloudlet.

Figure 3.4 shows our results on all four tasks. Blue lines show how the event recalls of EarlyDiscard filters for different tasks change as a function of cutoff threshold. The MobileNet DNN filter we used is able to detect all the events for T1 and T4 even at a high cutoff threshold. For T2 and T3, the majority of the events are detected. Achieving high recall on T2 and T3 (on 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 Tier-3 device drops a frame containing an important event, improved cloudlet processing cannot help. The results in the third column of Table 3.4 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 Table 3.4 confirm that early discard contributes little to result latency. The amounts range from 0.1 s for T1 to 12.7 s for T3.

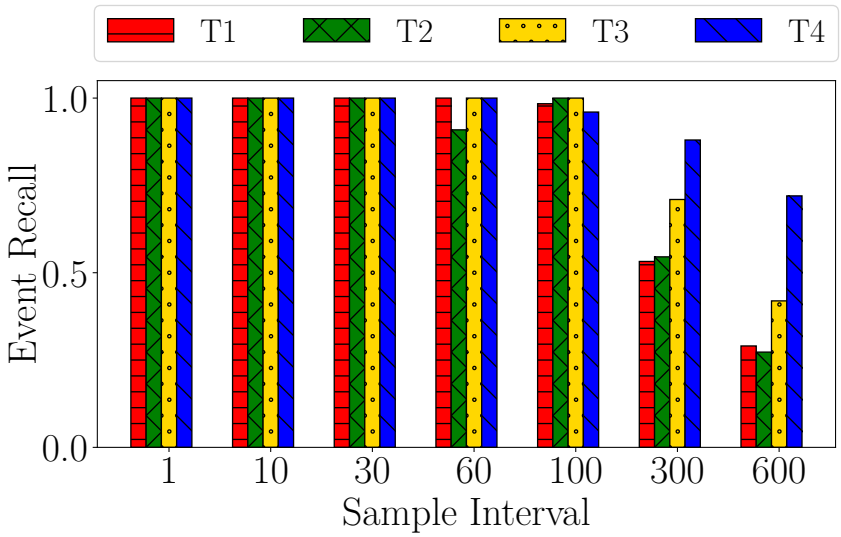


Figure 3.5: Event Recall at Different Sampling Intervals

Bandwidth

Columns 5–7 of Table 3.4 pertain to wireless bandwidth demand for the benchmark suite with early discard. The figures shown are based on H.264 encoding of each individual frames in the video transmission. Average bandwidth is calculated as the total data transmitted divided by mission duration. Comparing column 5 of Table 3.4 with column 2 of Table 3.3, we see that all videos in the benchmark suite are benefited by early discard (Note T3 and T4 have the same test dataset as T2). For T2, T3, and T4, the bandwidth is reduced by more than 10x. The amount of benefit is greatest for rare events (T2 and T3). When events are rare, the Tier-3 device can drop many frames.

Figure 3.4 provides deeper insight into the effectiveness of cutoff-threshold on event recall. It also shows how many true positives (violet) and false positives (aqua) are transmitted. Ideally, the aqua section should be zero. However for T2, most frames transmitted are false positives, indicating the early discard filter has low precision. The other tasks exhibit far fewer false positives. This suggests that the opportunity exists for significant bandwidth savings if precision could be further improved, without hurting recall.

3.3.4 Use of Sampling

Given the relatively low precision of the weak detectors, a significant number of false positives are transmitted. Furthermore, the occurrence of an object will likely last through many frames, so true positives are also often redundant for simple detection tasks. Both of these result in excessive consumption of precious bandwidth. This suggests that simply restricting the number of transmitted frames by sampling may help reduce bandwidth consumption.

Figure 3.5 shows the effects of sending a sample of frames from Tier-3, without any content-

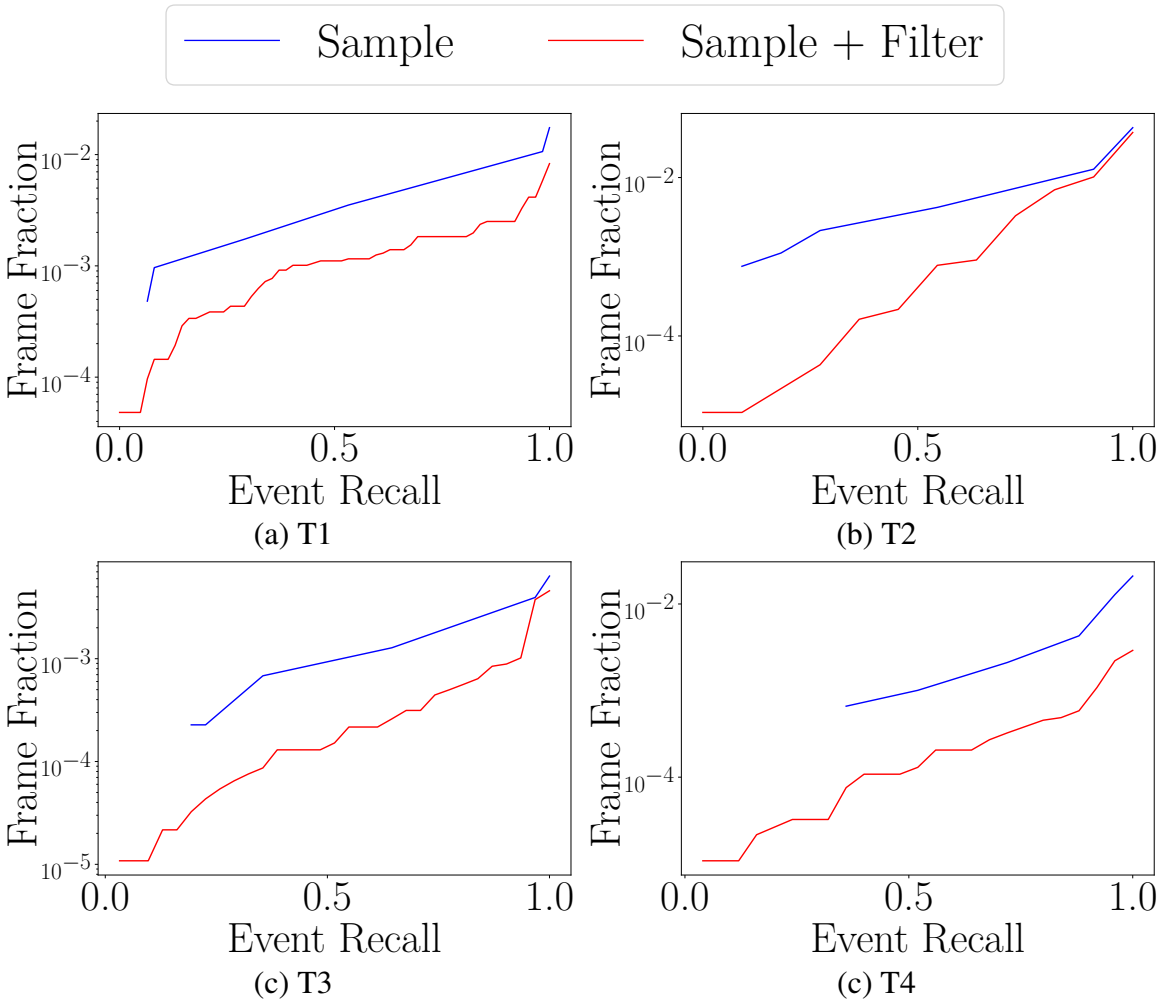


Figure 3.6: Sample with Early Discard. Note the log scale on y-axis.

JPEG Frame Sequence (MB)	H264 High Quality (MB)	H264 Medium Quality (MB)	H264 Low Quality (MB)
5823	3549	1833	147

H264 high quality uses Constant Rate Factor (CRF) 23. Medium uses CRF 28 and low uses CRF 40 [69].

Table 3.5: Test Dataset Size With Different Encoding Settings

based filtering. Based on these results, we can reduce the frames sent as little as one per second and still get adequate recall at the cloudlet. Note that this result is very sensitive to the actual duration of the events in the videos. For the detection tasks outlined here, most of the events (e.g., presences of a particular elephant) last for many seconds (100's of frames), so such sparse sampling does not hurt recall. However, if the events were of short duration, e.g., just a few frames long, then this method would be less effective, as sampling may lead to many missed events (false negatives).

Can we use content-based filtering along with sampling to further reduce bandwidth consumption? Figure 3.6 shows results when running early discard on a sample of the frames. This shows that for the same recall, we can reduce the bandwidth consumed by another factor of 5 on average over sampling alone. This effective combination can reduce the average bandwidth consumed for our test videos to just a few hundred kilobits per second. Furthermore, more processing time is available per processed frame, allowing more sophisticated algorithms to be employed, or to allow smaller tiles to be used, improving accuracy of early discard.

One case where sampling is not an effective solution is when all frames containing an object need to be sent to the cloudlet for some form of activity or behavior analysis from a complete video sequence. In this case, bandwidth will not reduce much, as all frames in the event sequence must be sent. However, the processing time benefits of sampling may still be exploited, provided all frames in a sample interval are transmitted on a match.

3.3.5 Effects of Video Encoding

One advantage of the Dumb strategy is that since all frames are transmitted, one can use a modern video encoding to reduce transmission bandwidth. With early discard, only a subset of disparate frames are sent. These will likely need to be individually compressed images, rather than a video stream. How much does the switch from video to individual frames affect bandwidth?

In theory, this can be a significant impact. Video encoders leverage the similarity between consecutive frames, and model motion to efficiently encode the information across a set of frames. Image compression can only exploit similarity within a frame, and cannot efficiently reduce number of bits needed to encode redundant content across frames. To evaluate this difference, we start with extracted JPEG frame sequences of our video data set. We encode the frame sequence with different H.264 settings. Table 3.5 compares the size of frame sequences in JPEG and the encoded video file sizes. We see only about 3x difference in the data size for the medium quality. We can increase the compression (at the expense of quality) very easily, and are able to reduce the video data rate by another order of magnitude before quality degrades catastrophically.

However, this compression does affect analytics. Even at medium quality level, visible compression artifacts, blurring, and motion distortions begin to appear. Initial experiments analyzing compressed videos show that these distortions do have a negative impact on accuracy of analytics. Using average precision analysis, a standard method to evaluate accuracy, we estimate that the most accurate model (Faster-RCNN ResNet101) on low quality videos performs similarly to

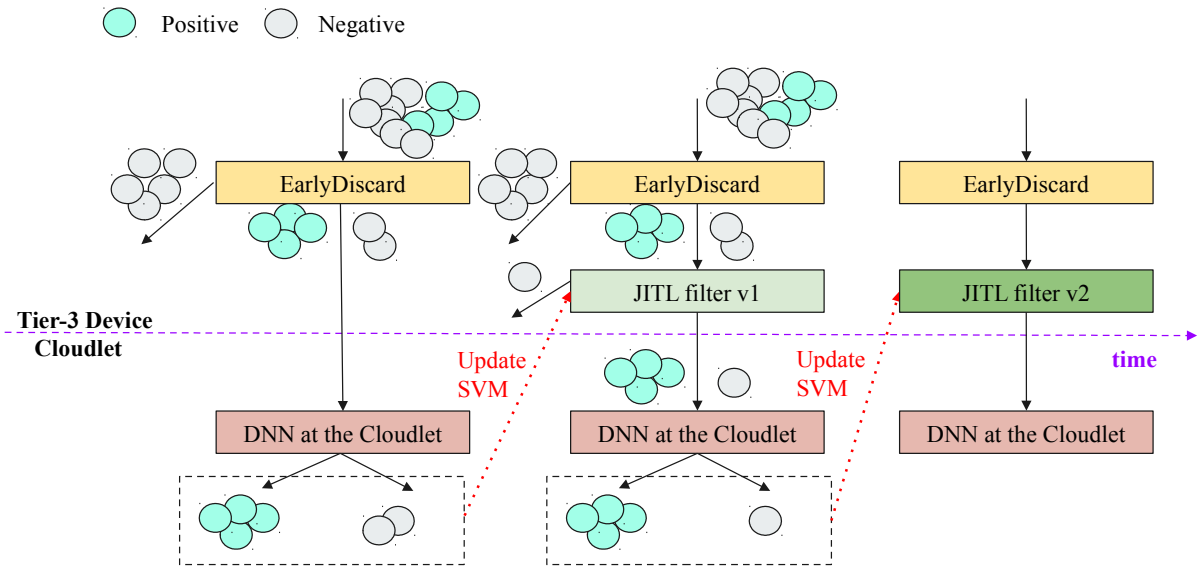


Figure 3.7: JITL Pipeline

the less accurate model (Faster-RCNN InceptionV2) on high quality JPEG images. This negates the benefits of using the state-of-art models.

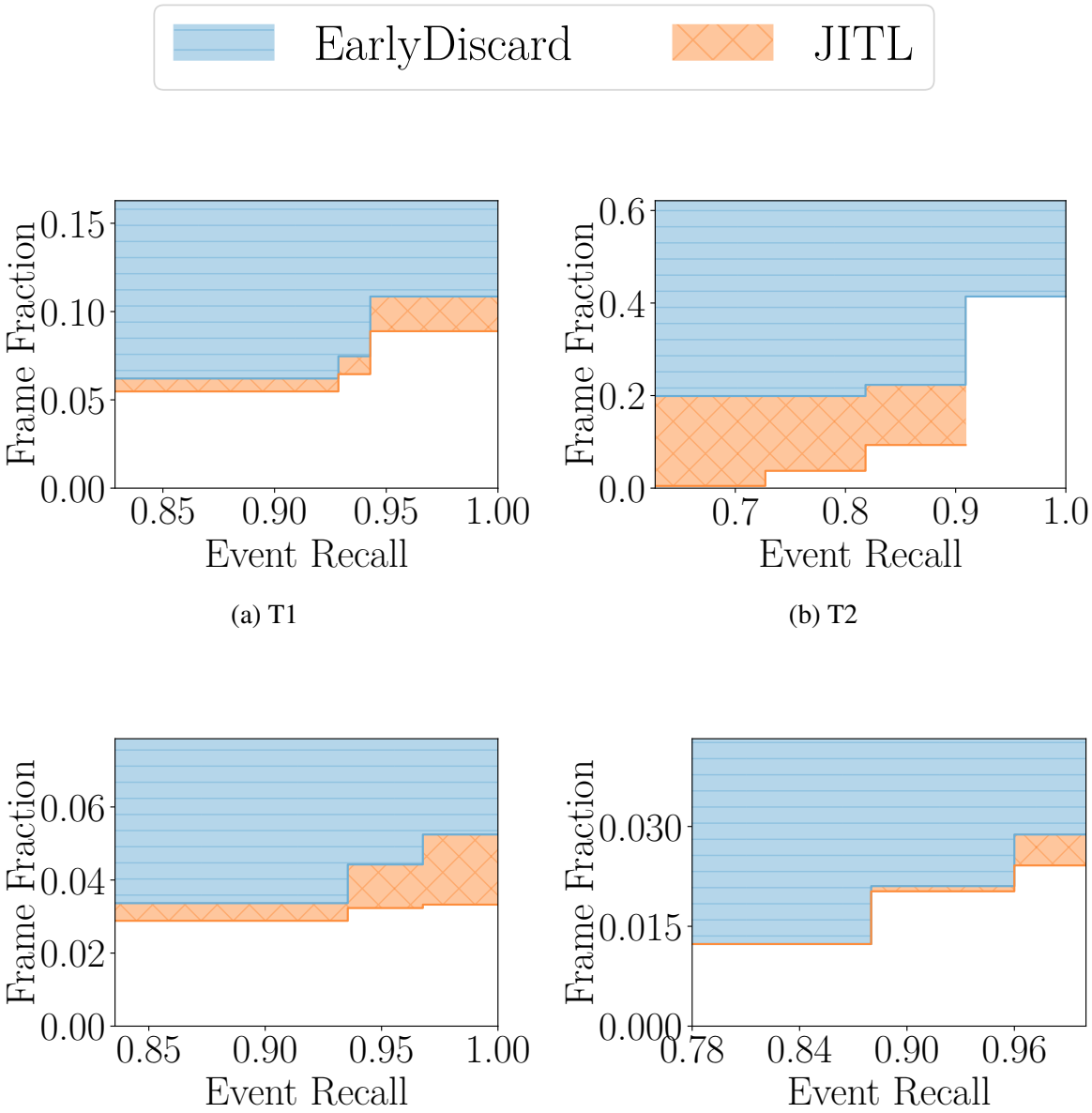
In our EarlyDiscard design, we pay a penalty of sending frames instead of a compressed low quality video stream. This overhead (approximately 30x) is compensated by the 100x reduction in frames transmitted due to sampling with early discard. In addition, the selective frame transmission preserves the accuracy of the state-of-art detection techniques.

Finally, one other option is to treat the set of disparate frames as a sequence and employ video encoding at high quality. This can ultimately eliminate the per frame overhead while maintaining accuracy. However, this will require a complex setup with both low-latency encoders and decoders, which can generate output data corresponding to a frame as soon as input data is ingested, with no buffering, and can wait arbitrarily long for additional frame data to arrive.

For the experiments in the rest of this chapter, we only account for the fraction of frames transmitted, rather than the choice of specific encoding methods used for those frames.

3.4 Just-In-Time-Learning (JITL) Strategy To Improve Early Discard

While EarlyDiscard filters are customized and optimized for specific tasks (e.g. detecting a human with red life jacket), we observe that EarlyDiscard filters do not leverage context information



(a) T1

(b) T2

(c) T3

(d) T4

Figure 3.8: JITL Fraction of Frames under Different Event Recall

within a specific video stream. Opportunities exist if we could further specialize the computer vision processing to the characteristics of video streams.

We propose Just-in-time-learning (JITL), which tunes the Tier-3 processing pipeline to the characteristics of the current task in order to reduce transmitted false positives from the Tier-3 device, and thereby reducing wasted bandwidth. Intuitively, JITL leverages temporal locality in video streams to quickly adapt processing outcomes based on recent feedback.

It is inspired by the ideas of cascade architecture from the computer vision community [114], but is different in construction. A JITL filter is a cheap cascade filter that distinguishes between the EarlyDiscard DNN's *true positives* (frames that are actually interesting) and *false positives* (frames that are wrongly considered interesting). Specifically, when a frame is reported as positive by EarlyDiscard, it is then passed through a JITL filter. If the JITL filter reports negative, the frame is regarded as a false positive and will not be sent. Ideally, all *true positives* from EarlyDiscard are marked *positive* by the JITL filter, and all *false positives* from EarlyDiscard are marked *negative*. Frames dropped by EarlyDiscard are not processed by the JITL filter, so this approach can only serve to improve precision, but not recall.

As shown in Figure 3.7 during task execution, a JITL filter is trained on the cloudlet using the frames transmitted from the Tier-3 device. The frames received on the cloudlet are predicted positive by the EarlyDiscard filter. The cloudlet, with more processing power, is able to run more accurate DNNs to identify true positives and false positives. Using this information as a feedback on how well current Tier-3 processing pipeline is doing, a small and lightweight JITL filter is trained to distinguish true positives and false positives of EarlyDiscard filters. These JITL filters are then pushed to the Tier-3 device to run as a cascade filter after the EarlyDiscard DNN.

True or false positive frames have high temporal locality throughout a task. The JITL filter is expected to pick up the features that confused the EarlyDiscard DNN in the immediate past and improve the pipeline's accuracy in the near future. These features are usually specific to the current task execution, and may be affected by terrain, shades, object colors, and particular shapes or background textures.

JITL can be used with EarlyDiscard DNNs of different cutoff probabilities to strike different trade-offs. In a bandwidth-favored setting, JITL can work with an aggressively selective Early-Discard DNN to further reduce wasted bandwidth. In a recall-favored setting, JITL can be used with a lower-cutoff DNN to preserve recall.

In our implementation, we use a linear support vector machine (SVM) [31] as the JITL filter. Linear SVM has several advantages: 1) short training time in the order of seconds; 2) fast inference; 3) only requires a few training examples; 3) small in size to transmit, usually on the order of 50KB in our experiments. The input features to the JITL SVM filter are the image features extracted by the EarlyDiscard DNN filter. In our case, since we are using MobileNet as our EarlyDiscard filter, they are the 1024-dimensional vector elements from the second last layer of MobileNet. This vector, also called "bottleneck values" or "transfer values" captures high-level features that represents the content of an image. Note that the availability of such image feature vector is not tied to a particular image classification DNN nor unique to MobileNet. Most image classification DNNs can be used as a feature extractor in this way.

3.4.1 JITL Experimental Setup

We used Jetson TX2 as our Tier-3 device platform and evaluated the JITL strategy on four tasks, T1 to T4. For the test videos in each task, we began with the EarlyDiscard filter alone and gradually trained and deployed JITL filters. Specifically, every ten seconds, we trained an SVM using the frames transmitted from the Tier-3 device and the ground-truth labels for these frames. In a real deployment, the frames would be marked as true positives or false positives by an accurate DNN running on the cloudlet since ground-truth labels are not available. In our experiments, we used ground-truth labels to control variables and remove the effect of imperfect prediction of DNN models running on the cloudlet.

In addition, we used the true and false positives from all previous intervals, not just the last ten seconds when training the SVM. The SVM, once trained, is used as a cascade filter running after the EarlyDiscard filter on the Tier-3 device to predict whether the output of the EarlyDiscard filter is correct or not. For a frame, if the EarlyDiscard filter predicts it to be interesting, but the JITL filter predicts the EarlyDiscard filter is wrong, it would not be transmitted to the cloudlet. In other words, following two criteria need to be satisfied for a frame to be transmitted to the cloudlet: 1) EarlyDiscard filter predicts it to be interesting 2) JITL filter predicts the EarlyDiscard filter is correct on this frame.

3.4.2 Evaluation

From our experiments, JITL is able to filter out more than 15% of remaining frames after EarlyDiscard without loss of event recall for three of four tasks. Figure 3.8 details the fraction of frames saved by JITL. The X-axis presents event recall. Y-axis represents the fraction of total frames. The blue region presents the achievable fraction of frames by EarlyDiscard. The orange region shows the additional savings using JITL. For T1, T3, and T4, at the highest event recall, JITL filters out more than 15% of remaining frames. This shows that JITL is effective at reducing the false positives thus improving the precision of the pipeline. However, occasionally, JITL predicts wrongly and removes true positives. For example, for T2, JITL does not achieve a perfect event recall. This is due to shorter event duration in T2, which results in fewer positive training examples to learn from. Depending on tasks, getting enough positive training examples for JITL could be difficult, especially when events are short or occurrences are few. To overcome this problem in practice, techniques such as synthetic data generation [27] could be explored to synthesize true positives from the background of the current task.

3.5 Applying EarlyDiscard and JITL to Wearable Cognitive Assistants

While the experiments in previous sections (3.3 3.4) are performed in a drone video analytics context, EarlyDiscard and JITL approaches can be applied more generally to live video analytics

(a) Searching for Lego Blocks (b) Assembling Lego Pieces

Figure 3.9: Example Images from a Lego Assembly Video

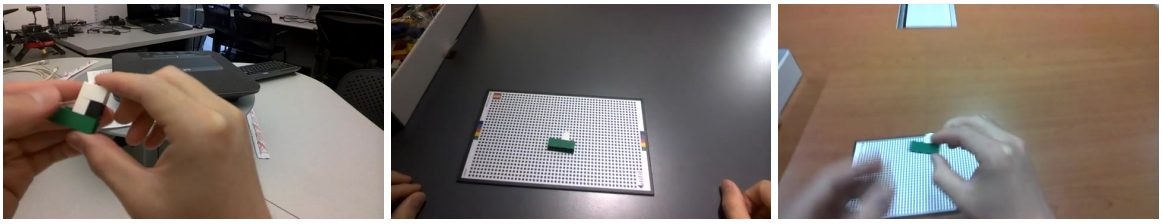


Figure 3.10: Example Images from LEGO Dataset

offloading from Tier-3 devices to Tier-2 edge data-centers. In this section, we use the LEGO application [16] to showcase how to apply these bandwidth saving approaches to WCAs.

The LEGO wearable cognitive assistant helps a user put together a specific Lego pattern by providing step-by-step audiovisual instructions. The application works as follows. The assistant first prompts a user an animated image showing the Lego block to use and asks the user to put it on the Lego board or assemble it with previous pieces. Following the guidance, the user searches for the particular Lego block, assemble it, and put the assembled piece on the Lego board for the next instruction. Figure 3.9 shows the first-person view images captured from the wearable device during this process. The assistant analyzes the assembled Lego piece on the Lego board by identifying its shape and color using computer vision and provides the suitable instruction.

Intuitively, to the assistant, frames capturing the assembled piece on the Lego board, (for example Figure 3.9 (b)) are the crucial frames to process, as they reflect the user's working progress. Figure 3.9 (a), on the other hand, is less interesting as it does not contain information on user progress. If some cheap processing on the wearable device could distinguish (a) from (b), bandwidth consumption can be reduced as we can discard Figure 3.9 (a) early on the wearable device without transmitting the frame to the cloudlet for processing. This provides opportunities to apply EarlyDiscard and JITL.

We collect a LEGO dataset of twelve videos, in which users assemble Lego pieces in three environments with different background, lighting, and viewpoints. Figure 3.10 shows example images from the dataset. We run the LEGO WCA on these videos to get pseudo ground truth labels. Specifically, for each frame, based on the outputs of the LEGO WCA vision processing, we categorize the frame to be either "interesting" or "not interesting". A frame is considered to

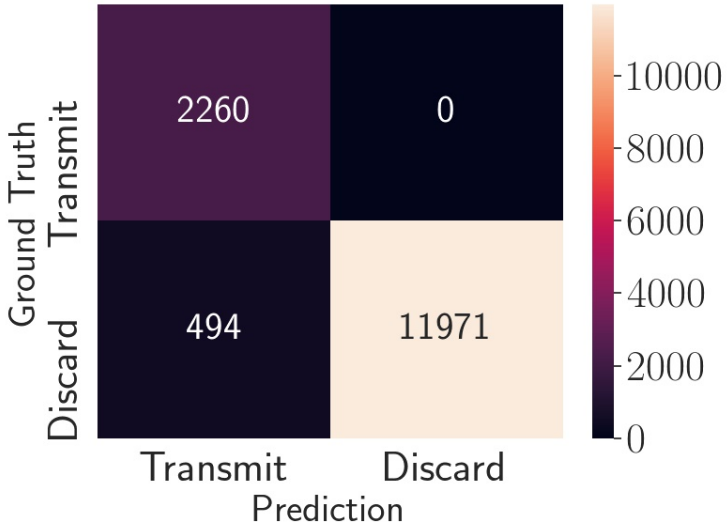


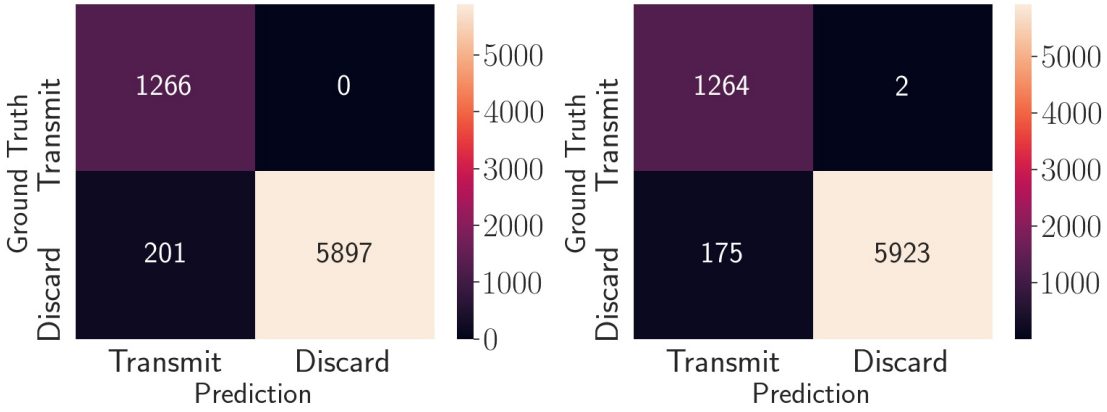
Figure 3.11: EarlyDiscard Filter Confusion Matrix

be interesting if a LEGO board is found in the frame, otherwise considered not interesting.

We use this dataset to finetune a MobileNet DNN in order to automatically distinguish interesting frames from the boring ones for EarlyDiscard. For each of the three environments, we randomly select two videos for training, one video for validation, and one video for testing. We randomly sample 2000 interesting images and 2000 boring images from the six training videos as the training data. Similarly, we random sample 200 interesting images and 200 boring images from the three validation videos as the validation data. We implement MobileNet transfer learning using the PyTorch framework [80]. We train the model for 20 epochs and select the model weights that give the highest accuracy on the validation set as the model for inference.

Our test sets have in total 14725 frames. Figure 3.11 shows the confusion matrix of our trained EarlyDiscard classifier. X-axis represents the predicted results: "Transmit" means the frame is predicted to be interesting and should be transmitted to cloudlet for processing while "Discard" means the frame is predicted to be boring and should not be transmitted. Similarly, Y-axis represents the ground truth results. As we can see, the classifier correctly predicts 2260 out of 14725 frames to be interesting and correctly suppresses 11971 frames. With EarlyDiscard in place, only 19% of all the frames are transmitted. Meanwhile, the false negative is 0 frame, meaning no "interesting" frame is wrongly discarded. This is the result of biasing the classifier towards recall instead of precision.

Among all the frames that are transmitted, 18% of them are false positives. These 494 false positives suggest that there are room to improve using JITL. For each of the test videos, we use the first half of the video as training examples for JITL to train a SVM that produces a confidence score for EarlyDiscard prediction. Figure 3.12 compares the confusion matrix of using EarlyDiscard alone with EarlyDiscard + JITL. As we can see, JITL reduces 13% of the false positives at the cost of 2 false negatives. Note that these 2 false negative frames do not result in missing instructions as adjacent interesting frames are still transmitted.



(a) EarlyDiscard (b) EarlyDiscard + JITL

Figure 3.12: JITL Confusion Matrix

3.6 Related Work

In the context of drone video analytics, Wang et al. [117] shares our concern for wireless bandwidth, but focuses on coordinating a network of drones to capture and broadcast live sport event. In addition, Wang et al [116] explored adaptive video streaming with drones using content-based compression and video rate adaptation. While we share their inspiration, our work leverages characteristics of DNNs to enable mission-specific optimization strategies.

Much previous work on static camera networks explored efficient use of compute and network resources at scale. Zhang et al. [122] studied resource-quality trade-off under result latency constraints in video analytics systems. Kang et al. [51] worked on optimizing DNN queries over videos at scale. While they focus on supporting a large number of computer vision workload, our work optimizes for the first hop wireless bandwidth. In addition, Zhang et al. [123] designed a wireless distributed surveillance system that supports a large geographical area through frame selection and content-aware traffic scheduling. In contrast, our work does not assume static cameras. We explore techniques that tolerate changing scenes in video feeds and strategies that work for moving cameras.

Some previous work on computer vision in mobile settings has relevance to aspects of our system design. Chen et al. [15] explore how continuous real-time object recognition can be done on mobile devices. They meet their design goals by combining expensive object detection with computationally cheap object tracking. Although we do not use object tracking in our work, we share the resource concerns that motivate that work. Naderiparizi et al. [72] describe a programmable early-discard camera architecture for continuous mobile vision. Our work shares their emphasis on early discard, but differs in all other aspects. In fact, our work can be viewed as complementing that work: their programmable early-discard camera would be an excellent choice for Tier-3 devices. Lastly, Hu et al [43] have investigated the approach of using lightweight computation on a mobile device to improve the overall bandwidth efficiency of a computer vision pipeline that offloads computation to the edge. We share their concern for

wireless bandwidth, and their use of early discard using inexpensive algorithms on the mobile device.

3.7 Chapter Summary and Discussion

In this chapter, we address the bandwidth challenge of running many WCAs at scale. We propose two application-agnostic methods to reduce bandwidth consumption when offloading computation to edge servers.

The EarlyDiscard technique employs on-board filters to select interesting frames and suppress the transmission of mundane frames to save bandwidth. In particular, cheap yet effective DNN filters are trained offline to fully leverage the large quantity of training data and the high learning capacities of DNNs. Building on top of EarlyDiscard, JITL adapts an EarlyDiscard filter to a specific environment online. While a WCA is running, JITL continuously evaluates the EarlyDiscard filter and reduces the number of false positives by predicting whether an Eary-Discard decision is made correctly. These two techniques together reduce the total number of unnecessary frames transmitted.

We evaluate these two strategies first in the drone live video analytics context for search tasks in domains such as search-and-rescue, surveillance, and wildlife conservation, and then for WCAs. Our experimental results show that this judicious combination of Tier-3 processing and edge-based processing can save substantial wireless bandwidth and thus improve scalability, without compromising result accuracy or result latency.

Chapter 4

Application-Aware Techniques to Reduce Offered Load

Elasticity is a key attribute of cloud computing. When load rises, new servers can be rapidly spun up. When load subsides, idle servers can be quiesced to save energy. Elasticity is vital to scalability, because it ensures acceptable response times under a wide range of operating conditions. To benefit, cloud services need to be architected to easily scale out to more servers. Such a design is said to be "cloud-native."

In contrast, edge computing has limited elasticity. As its name implies, a cloudlet is designed for much smaller physical space and electrical power than a cloud data center. Hence, the sudden arrival of an unexpected flash crowd can overwhelm a cloudlet. Since low end-to-end latency is a prime reason for edge computing, shifting load elsewhere (e.g., the cloud) is not an attractive solution. *How do we build multi-user edge computing systems that preserve low latency even as load increases?* That is the focus of the next two chapters.

Our approach to scalability is driven by the following observation. Since compute resources at the edge cannot be increased on demand, the only paths to scalability are (a) to reduce offered load, as discussed in this chapter, or (b) to reduce queueing delays through improved end-to-end scheduling, as discussed in Chapter 5. Otherwise, the mismatch between resource availability and offered load will lead to increased queueing delays and hence increased end-to-end latency. Both paths require the average burden placed by each user on the cloudlet to fall as the number of users increases. This, in turn, implies *adaptive application behavior* based on guidance received from the cloudlet or inferred by the user's mobile device. In the context of Figure 2.1, scalability at the left is achieved very differently from scalability at the right. The relationship between Tier-3 and Tier-2 is *non-workload-conserving*, while that between Tier-1 and other tiers is workload-conserving.

While we demonstrated application-agnostic techniques to reduce network transmission between Tier-3 and Tier-2 in Chapter 3, offered load can be further reduced with application assistance. We claim that scalability at the edge can be better achieved for applications that have been designed with this goal in mind. We refer to applications that are specifically written to lever-

age edge infrastructure as *edge-native applications.* These applications are deeply dependent on the services that are only available at the edge (such as low-latency offloading of compute, or real-time access to video streams from edge-located cameras), and are written to adapt to scalability-relevant guidance. For example, an application at Tier-3 may be written to offload object recognition in a video frame to Tier-2, but it may also be prepared for the return code to indicate that a less accurate (and hence less compute-intensive) algorithm than normal was used because Tier-2 is heavily loaded. Alternatively, Tier-2 or Tier-3 may determine that the wireless channel is congested; based on this guidance, Tier-3 may reduce offered load by preprocessing a video frame and using the result to decide whether it is worthwhile to offload further processing of that frame to the cloudlet. Several earlier works [19, 43] have shown that even modest computation, such as color filtering and shallow DNN processing, at Tier-3 can make surprisingly good predictions about whether a specific use of Tier-2 is likely to be worthwhile.

Edge-native applications may also use *cross-layer adaptation strategies,* by which knowledge from Tier-3 or Tier-2 is used in the management of the wireless channel between them. For example, an assistive augmented reality (AR) application that verbally guides a visually-impaired person may be competing for the wireless channel and cloudlet resources with a group of AR gamers. In an overload situation, one may wish to favor the assistive application over the gamers. This knowledge can be used by the cloudlet operating system to preferentially schedule the more important workload. It can also be used for prioritizing network traffic by using *fine-grain network slicing,* as envisioned in 5G [20].

Wearable cognitive assistance, perceived to be "killer apps" for edge computing, are perfect exemplars of edge-native applications. In the rest of this chapter, we showcase how we can leverage unique application characteristics of WCAs to adapt application behavior and reduce offered load. Our work is built on the Gabriel platform [17, 35], shown in Figure 2.4. The Gabriel front-end on a wearable device performs preprocessing of sensor data (e.g., compression and encoding), which it streams over a wireless network to a cloudlet. We refer to the Gabriel platform with new mechanisms that handle multitenancy, perform resource allocation, and support application-aware adaptation as "Scalable Gabriel" and the single-user baseline platform as "Original Gabriel".

4.1 Adaptation Architecture and Strategy

The original Gabriel platform has been validated in meeting the latency bounds of WCA applications in single-user settings [17]. Scalable Gabriel aims to meet these latency bounds in multi-user settings, and to ensure performant multitenancy even in the face of overload. We take two complementary approaches to scalability. The first is for applications to reduce their offered load to the wireless network and the cloudlet through adaptation. The second uses end-to-end scheduling of cloudlet resources to minimize queueing and impacts of overload (See Chapter 5 for more details). We both approaches, and combine them using the system architecture shown in Figure 4.1. We assume benevolent and collaborative clients in the system.

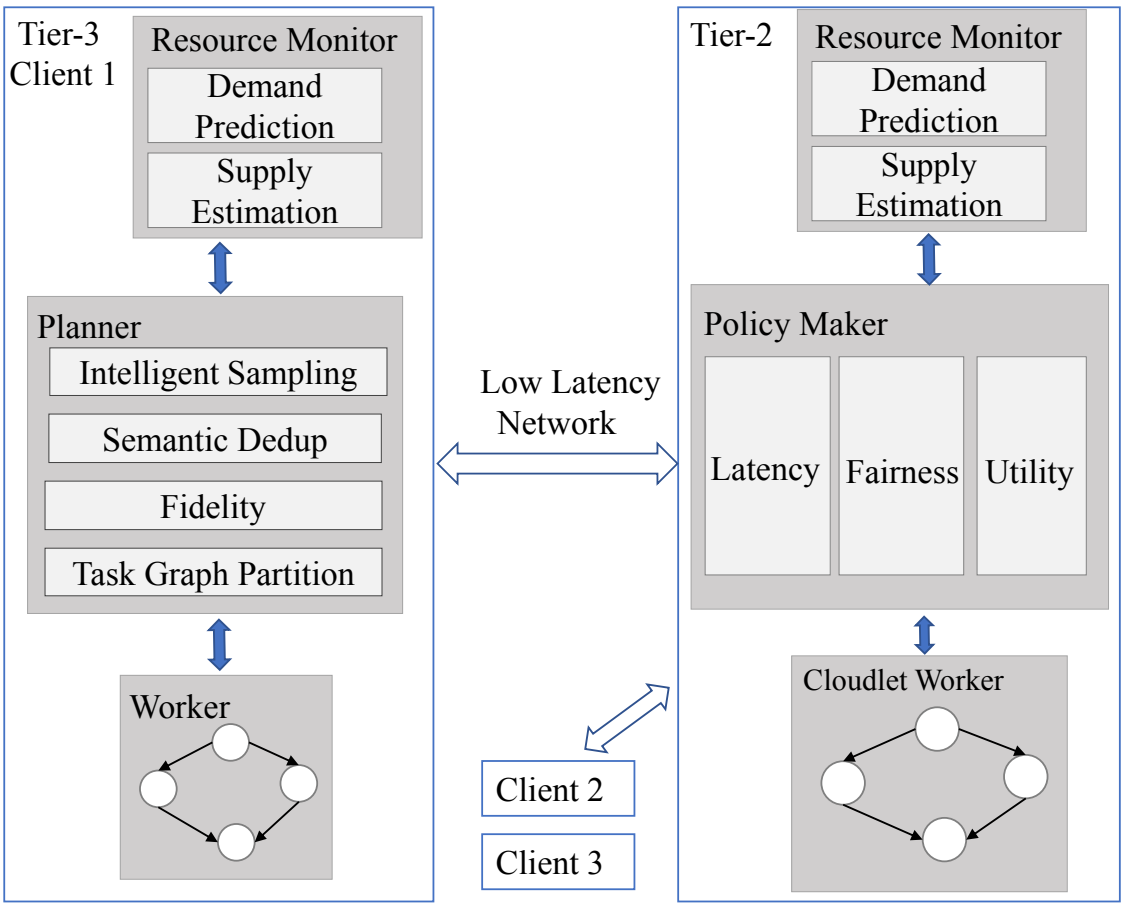


Figure 4.1: Adaptation Architecture

4.2 System Architecture

Computer vision processing is at the core of wearable cognitive assistance. We consider scenarios in which multiple Tier-3 devices concurrently offload their vision processing to a single cloudlet over a shared wireless network. The devices and cloudlet work together to adapt workloads to ensure good performance across all of the applications vying for the limited Tier-2 resources and wireless bandwidth. This is reflected in the system architecture shown in Figure 4.1.

Monitoring of resources is done at both Tier-3 and Tier-2. Certain resources, such as battery level, are device-specific and can only be monitored at Tier-3. Other shared resources can only be monitored at Tier-2: these include processing cores, memory, and GPU. Wireless bandwidth and latency are measured independently at Tier-3 and Tier-2, and aggregated to achieve better estimates of network conditions.

This information is combined with additional high-level predictive knowledge and factored into scheduling and adaptation decisions. The predictive knowledge could arise at the cloudlet (e.g., arrival of a new device, or imminent change in resource allocations), or at the Tier-3 device

(e.g., application-specific, short-term prediction of resource demand). All of this information is fed to a *policy module* running on the cloudlet. This module is guided by an external policy specification and determines how cloudlet resources should be allocated across competing Tier-3 applications. Such policies can factor in latency needs and fairness, or simple priorities (e.g., a blind person navigation assistant may get priority over an AR game).

A *planner module* on the Tier-3 device uses current resource utilization and predicted short-term processing demand to determine which workload reduction techniques (described in Section 4.4) should be applied to achieve best performance for the particular application given the resource allocations.

4.3 Adaptation Goals

For WCAs, the dominant class of offloaded computations are computer vision operations, e.g., object detection with deep neural networks (DNNs), or activity recognition on video segments. The interactive nature of these applications precludes the use of deep pipelining that is commonly used to improve the efficiency of streaming analytics. Here, end-to-end latency of an individual operation is more important than throughput. Further, it is not just the mean or median of latency, but also the tail of the distribution that matters. There is also significant evidence that user experience is negatively affected by unpredictable variability in response times. Hence, a small mean with short tail is the desired ideal. Finally, different applications have varying degrees of benefit or utility at different levels of latency. Thus, our adaptation strategy incorporates application-specific utility as a function of latency as well as policies maximizing the total utility of the system.

4.4 Leveraging Application Characteristics

WCA applications exhibit certain properties that distinguish them from other video analytics applications studied in the past. Adaptation based on these attributes provides a unique opportunity to improve scalability.

Human-Centric Timing: The frequency and speed with which guidance must be provided in a WCA application often depends on the speed at which the human performs a task step. Generally, additional guidance is not needed until the instructed action has been completed. For example, in the RibLoc assistant (Chapter 2), drilling a hole in bone can take several minutes to complete. During the drilling, no further guidance is provided after the initial instruction to drill. Inherently, these applications contain *active phases,* during which an application needs to sample and process video frames as fast as possible to provide timely guidance, and *passive phases,* during which the human user is busy performing the instructed step. During a passive phase, the application can be limited to sampling video frames at a low rate to determine when the user has completed or nearly completed the step, and may need guidance soon. Although durations of human operations need to be considered random variables, many have empirical

	Question	Example	Load-reduction Technique
1	How often are instructions given, compared to task duration?	Instructions for each step in IKEA lamp assembly are rare compared to the total task time, e.g., 6 instructions over a 10 minute task.	Enable adaptive sampling based on active and passive phases.
2	Is intermittent processing of input frames sufficient for giving instructions?	Recognizing a face in any one frame is sufficient for whispering the person's name.	Select and process key frames.
3	Will a user wait for system responses before proceeding?	A first-time user of a medical device will pause until an instruction is received.	Select and process key frames.
4	Does the user have a predefined workspace in the scene?	Lego pieces are assembled on the lego board. Information outside the board can be safely ignored.	Focus processing attention on the region of interest.
5	Does the vision processing involve identifying and locating objects?	Identifying a toy lettuce for a toy sandwich.	Use tracking as cheap approximation for detection.
6	Are the vision processing algorithms insensitive to image resolution?	Many image classification DNNs limit resolutions to the size of their input layers.	Downscale sampled frames on device before transmission.
7	Can the vision processing algorithm trade off accuracy and computation?	Image classification DNN MobileNet is computationally cheaper than ResNet, but less accurate.	Change computation fidelity based on resource utilization.
8	Can IMUs be used to identify the start and end of user activities?	User's head movement is significantly larger when searching for a Lego block.	Enable IMU-based frame suppression.
9	Is the Tier-3 device powerful enough to run parts of the processing pipeline?	A Jetson TX2 can run MobileNet-based image recognition in real-time.	Partition the vision pipeline between Tier-3 and Tier-2.

Table 4.1: Application characteristics and corresponding applicable techniques to reduce load

lower bounds. Adapting sampling and processing rates to match these active and passive phases can greatly reduce offered load. Further, the offered load across users is likely to be uncorrelated because they are working on different tasks or different steps of the same task. If inadvertent synchronization occurs, it can be broken by introducing small randomized delays in the task guidance to different users. These observations suggest that proper end-to-end scheduling can enable effective use of cloudlet resources even with multiple concurrent applications.

Event-Centric Redundancy: In many WCA applications, guidance is given when a user event causes visible state change. For example, placing a lamp base on a table triggers the IKEA Lamp application to deliver the next assembly instruction. Typically, the application needs to process video at high frame rate to ensure that such state change is detected promptly, leading to further guidance. However, all subsequent frames will continue to reflect this change, and are essentially redundant, wasting wireless and computing resources. Early detection of redundant frames through careful semantic deduplication and frame selection at Tier-3 can reduce the use of wireless bandwidth and cloudlet cycles on frames that show no task-relevant change.

Inherent Multi-Fidelity: Many vision processing algorithms can tradeoff fidelity and computation. For example, frame resolution can be lowered, or a less sophisticated DNN used for inference, in order to reduce processing at the cost of lower accuracy. In many applications, a lower frame rate can be used, saving computation and bandwidth at the expense of response latency. Thus, when a cloudlet is burdened with multiple concurrent applications, there is scope to select operating parameters to keep computational load manageable. Exactly how to do so may be application-dependent. In some cases, user experience benefits from a trade-off that preserves fast response times even with occasional glitches in functionality. For others, e.g., safety-critical applications, it may not be possible to sacrifice latency or accuracy. This in turn, translates to lowered scalability of the latter class of application, and hence the need for a more powerful cloudlet and possibly different wireless technology to service multiple users.

4.4.1 Adaptation-Relevant Taxonomy

The characteristics described in the previous section largely hold for a broad range of WCA applications. However, the degree to which particular aspects are appropriate to use for effective adaptation is very application dependent, and requires a more detailed characterization of each application. To this end, our system requests a manifest describing an application from the developers. This manifest is a set of yes/no or short numerical responses to the questions in Table 4.1. Using these, we construct a taxonomy of WCA applications (shown in Figure 4.2), based on clusters of applications along dimensions induced from the checklist of questions. In this case, we consider two dimensions – the fraction of time spent in "active" phase, and the significance of the provided guidance (from merely advisory, to critical instructions). Our system varies the adaptation techniques employed to the different clusters of applications. We note that as more applications and more adaptation techniques are created, the list of questions can be extended, and the taxonomy can be expanded.

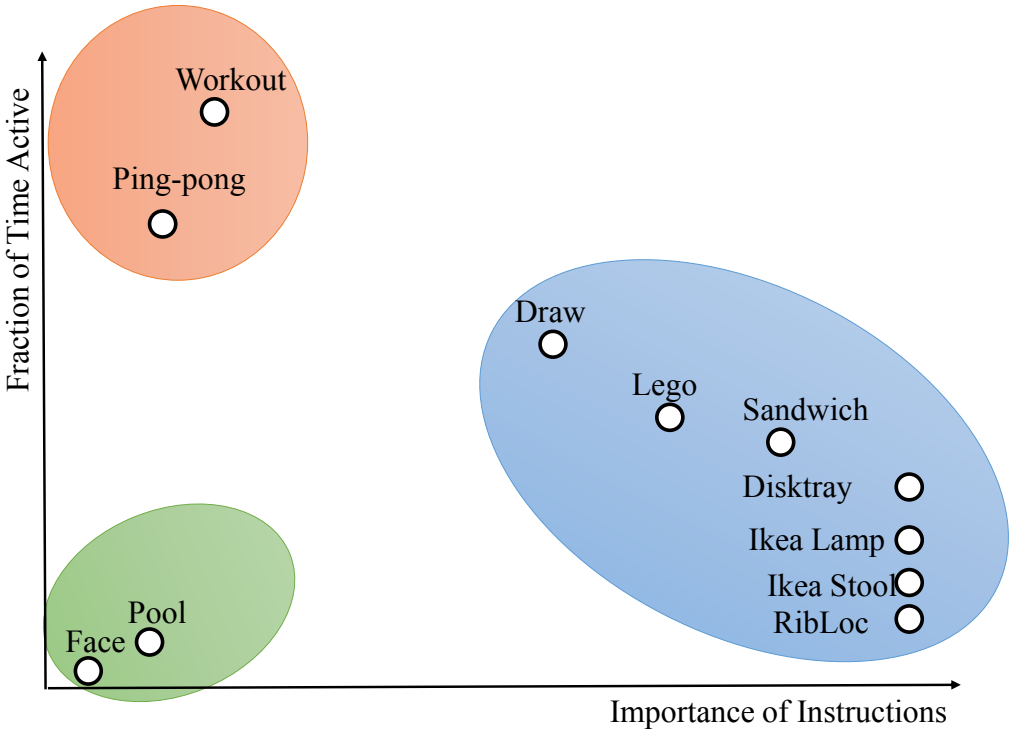


Figure 4.2: Design Space of WCA Applications

4.5 Adaptive Sampling

The processing demands and latency bounds of a WCA application can vary considerably during task execution because of human speed limitations. When the user is awaiting guidance, it is desirable to sample input at the highest rate to rapidly determine task state and thus minimize guidance latency. However, while the user is performing a task step, the application can stay in a passive state and sample at a lower rate. For a short period of time immediately after guidance is given, the sampling rate can be very low because it is not humanly possible to be done with the step. As more time elapses, the sampling rate has to increase because the user may be nearing completion of the step. Although this active-passive phase distinction is most characteristic of WCA applications that provide step-by-step task guidance (the blue cluster in the lower right of Figure 4.2), most WCA applications exhibit this behavior to some degree. As shown in the rest of this section, adaptive sampling rates can reduce processing load without impacting application latency or accuracy.

We use task-specific heuristics to define application active and passive phases. In an active application phase, a user is likely to be waiting for instructions or comes close to needing instructions, therefore application needs to be "active" by sampling and processing at high frequencies. On the other hand, applications can run at low frequency during passive phases when an instruction is unlikely to occur.

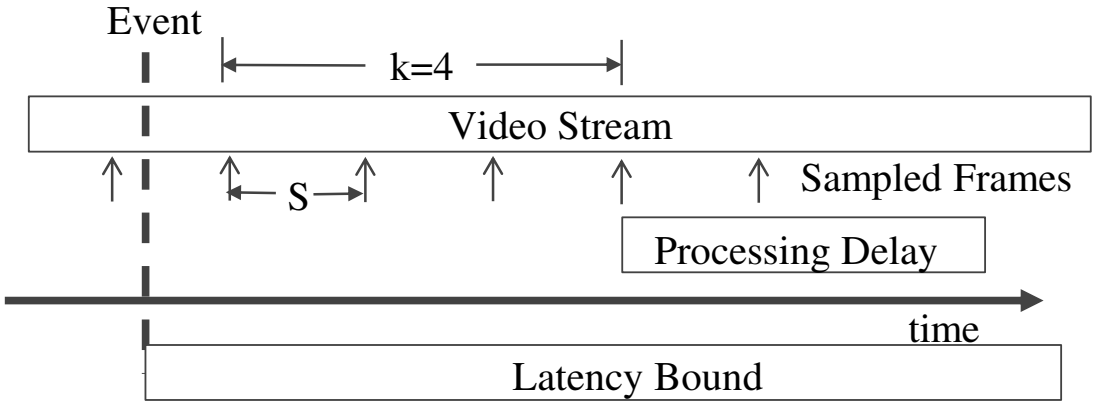


Figure 4.3: Dynamic Sampling Rate for LEGO

We use the LEGO application from Section 2.3 to show the effectiveness of adaptive sampling. By default, the LEGO application runs at active phase. The application enters passive phases immediately following the delivery of an instruction, since the user is going to take a few seconds searching and assembling LEGO blocks. The length and sampling rate of a passive phase is provided by the application to the framework. We provide the following system model as an example of what can be provided. We collect five LEGO traces with 13739 frames as our evaluation dataset.

Length of a Passive Phase: We model the time it takes to finish each step as a Gaussian distribution. We use maximum likelihood estimation to calculate the parameters of the Guassian model.

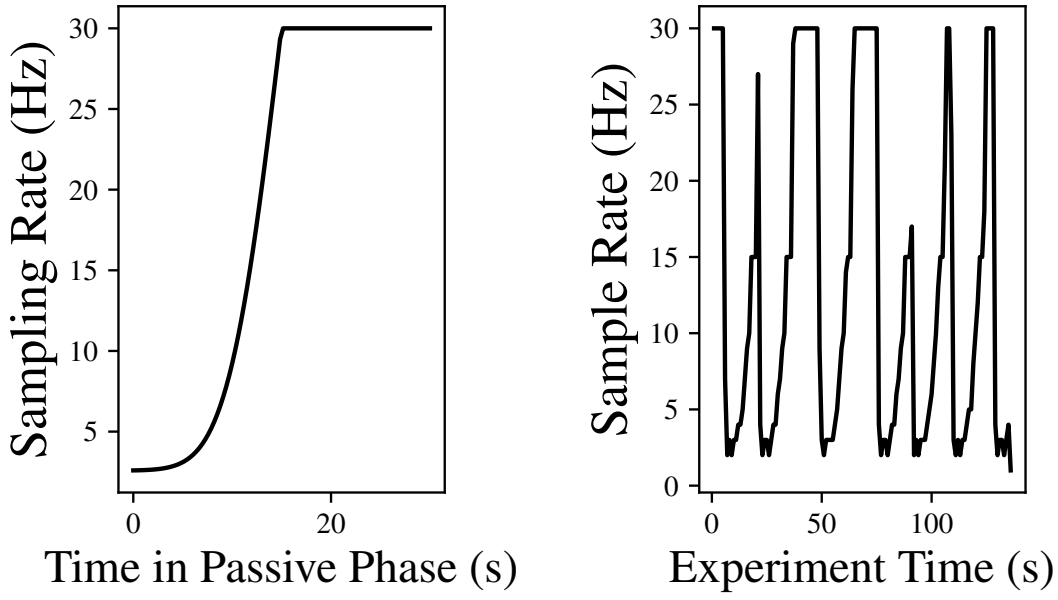
Lowest Sampling Rate in Passive Phase: The lowest sampling rate in passive phase still needs to meet application's latency requirement. Figure 4.3 shows the system model to calculate the largest sampling period S that still meets the latency bound. In particular,

$$(k-1)S + processing\_delay \leq latency\_bound$$

k represents the cumulative number of frames an event needs to be detected in order to be certain an event actually occurred. The LEGO application empirically sets this value to be 5.

Adaptation Algorithm: At the start of a passive phase, we set the sampling rate to the minimum calculated above. As time progresses, we gradually increase the sampling rate. The idea behind this is that the initial low sampling rates do not provide good latency, but this is acceptable, as the likelihood of an event is low. As the likelihood increases (based on the Gaussian distribution described earlier), we increase sampling rate to decrease latency when events are likely. Figure 4.4(a) shows the sampling rate adaptation our system employs during a passive phase. The sampling rate is calculated as

$$sr = min\_sr + \alpha * (max\_sr - min\_sr) * cdf\_Gaussian(t)$$



(a) Passive Sampling Rate

(b) Trace Sampling Rate

Figure 4.4: Adaptive Sampling Rate

sr is the sampling rate. t is the time after an instruction has been given. α is a recovery factor which determines how quickly the sampling rate rebounds to active phase rate.

Figure 4.4(b) shows the sampling rate for a trace as the application runs. The video captures a user doing 7 steps of a LEGO assembly task. Each drop in sampling rate happens after an instruction has been delivered to the user. Table 4.2 shows the percentage of frames sampled and guidance latency comparing adaptive sampling with naive sampling at half frequency. Our adaptive sampling scheme requires processing fewer frames while achieving a lower guidance latency.

4.6 IMU-based Approaches: Passive Phase Suppression

In many applications, passive phases can often be associated with the user's head movement. We illustrate with two applications here. In LEGO, during the passive phase, which begins after the user receives the next instruction, a user typically turns away from the LEGO board and starts searching for the next brick to use in a parts box. During this period, the computer vision algorithm would detect no meaningful task states if the frames are transmitted. In PING PONG application (Section 2.3), an active phase lasts throughout a rally. Passive phases are in between actual game play, when the user takes a drink, switches sides, or, most commonly, tracks down and picks up a wayward ball from the floor. These are associated with much large range of head movements than during a rally when the player generally looks toward the opposing player. Again, the frames can be suppressed on the client to reduce wireless transmission and load on

Trace	Sample Half Freq	Adaptive Sampling
1	50%	25%
2	50%	28%
3	50%	30%
4	50%	30%
5	50%	43%

(a) Percentage of Frames Sampled

	Guidance Delay (frames±stddev)
Sample Half Freq	7.6 ± 6.9
Adaptive Sampling	5.9 ± 8.2

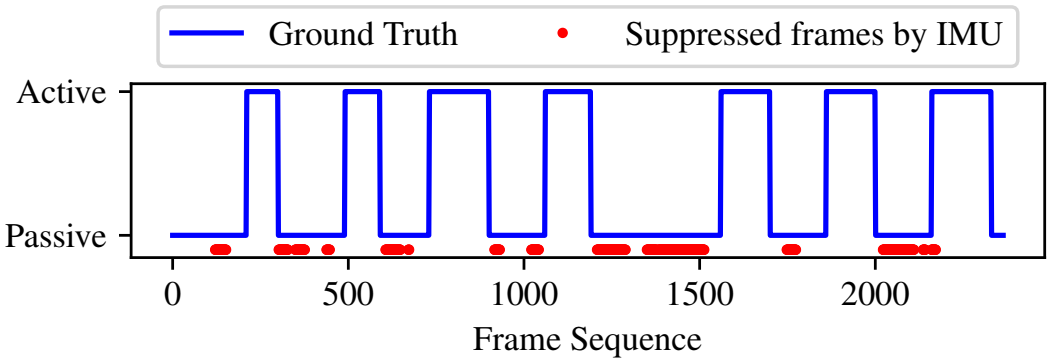
(b) Guidance Latency

Table 4.2: Adaptive Sampling Results

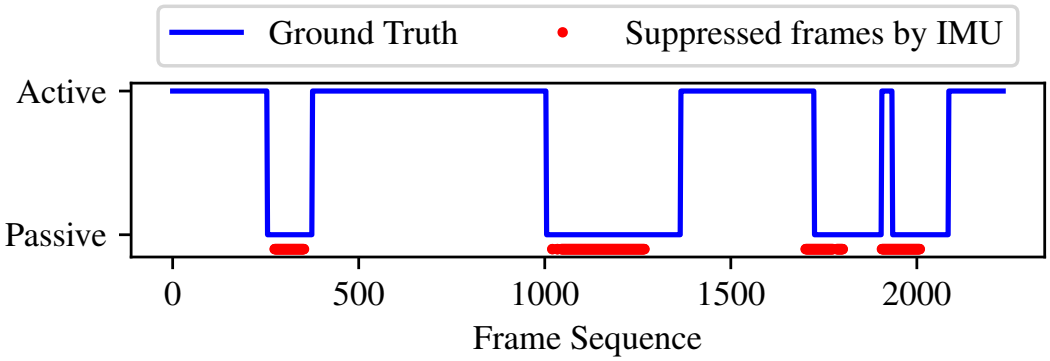
the cloudlet. In both scenarios, significant body movement can be detected through Inertial Measurement Unit (IMU) readings on the wearable device, and used to predict those passive phases.

For each frame, we get a six-dimensional reading from the IMU: rotation in three axes, and acceleration in three axes. We train an application-specific SVM to predict active/passive phases based on IMU readings, and suppress predicted passive frames on the client. Figure 4.5(a) and (b) show an example trace from LEGO and PING PONG, respectively. Human-labeled ground truth indicating passive and active phases is shown in blue. The red dots indicate predictions of passive phase frames based on the IMU readings; these frames are suppressed at the client and not transmitted. Note that in both traces, the suppressed frames also form streaks. In other words, a number of frames in a row can be suppressed. As a result, the saving we gain from IMU is orthogonal to that from adaptive sampling.

Although the IMU approach does not capture all of the passive frames (e.g., in LEGO, the user may hold his head steady while looking for the next part), when a passive frame is predicted, this is likely correct (i.e., high precision, moderate recall). Thus, we expect little impact on event detection accuracy or latency, as few if any active phase frames are affected. This is confirmed in Table 4.3, which summarizes results for five traces from each application. We are able to suppress up to 49.9% of passive frames for LEGO and up to 38.4% of passive frames in case of PING PONG on the client, while having minimal impact on application quality — incurring no delay in state change detection in LEGO, and less than 2% loss of active frames in PING PONG.



(a) LEGO



(b) PING PONG

Figure 4.5: Accuracy of IMU-based Frame Suppression

4.7 Related Work

Although edge computing is new, the techniques for reducing offered load to adapt application behaviors examined in this chapter bear some resemblance to work that was done in the early days of mobile computing.

Several different approaches to adapting application fidelity have been studied. Dynamic sampling rate with various heuristics for adaptation have been tried primarily in the context of individual mobile devices for energy efficiency [56, 65, 66, 112]. Semantic deduplication to reduce redundant processing of frames have been suggested by [42, 43, 51, 123]. Similarly, previous works have looked at suppression based on motion either from video content [58, 72] or IMUs [49]. Others have investigated exploiting multiple deep models with accuracy and resource tradeoffs [36, 50]. In addition, using tracking as approximate result of detection and recognition has been explored to leverage the temporal locality of video data and reduce computational demand [15, 115, 121]. While most of these efforts were in mobile-only, cloud-only, or mobile-cloud context, we explore similar techniques in an edge-native context.

Partitioning workloads between mobile devices and the cloud have been studied in sensor networks [74], throughput-oriented systems [21, 119], for interactive applications [15, 82], and

	Suppressed Passive Frames (%)	Max Delay of State Change Detection
Trace 1	17.9%	0
Trace 2	49.9%	0
Trace 3	27.1%	0
Trace 4	37.0%	0
Trace 5	34.1%	0

(a) LEGO

	Suppressed Passive Frames (%)	Loss of Active Frames (%)
Trace 1	21.5%	0.8%
Trace 2	30.0%	1.5%
Trace 3	26.2%	1.9%
Trace 4	29.8%	1.0%
Trace 5	38.4%	0.2%

(b) PING PONG

Table 4.3: Effectiveness of IMU-based Frame Suppression

from programming model perspectives [8]. We believe that these approaches will become important techniques to scale applications on heavily loaded cloudlets.

4.8 Chapter Summary and Discussion

In this chapter, we demonstrate that scalability can be increased by leveraging application characteristics to reduce offered load. Our approach to increased scalability is through adaptation. Specifically, we first present an adaptation-centric architecture that monitors and coordinates Tier-3 devices and the edge server. When contention arises, enabled are application-specific optimizations to reduce offered load to the edge server. In addition, we highlight two of adaptation techniques, selective sampling and IMU-based suppression. Our experiment show that they can significantly reduce the offered load. Finally, we provide a taxonomy to help developers reason about characteristics of their applications, and identify and specify reduction techniques applicable to their needs.

Chapter 5

Cloudlet Resource Management for Graceful Degradation of Service

In addition to workload reduction at the client as discussed in Chapter 4, another key aspect of adaptation lies in the resource management of cloudlet resources. We argue that naive resource management schemes using statistical multiplexing cannot satisfy the needs of edge-native applications in oversubscribed edge scenarios. Needed are intelligent mechanisms that take account of application degradation behaviors. In this chapter, we introduce and evaluate such an adaptation-centric cloudlet resource management mechanism.

In Original Gabriel which relies on operating system level statistical multiplexing for resource sharing, an increasing number of clients means less resources for each client. This results in all clients and all applications suffering from long response time. For wearable cognitive assistance, long response time results in feedback delivered too late, which has significantly decreased value. In a data-center setting, a cloud-native application, can quickly scale-up or scale-out by acquiring additional resources (e.g. instantiating more virtual machines). However, at the edge, since the total amount of hardware resources is constrained, acquiring more resources in face of a flash crowd is not possible. Instead, we make the observation that applications behave differently when the amount of resources allocated to them changes. We can leverage application adaptation characteristics to create a judicious and intelligent resource allocation plan that prioritizes some applications. In particular, we focus on mechanisms rather than policies. Our mechanism in this chapter enables external allocation policies to divide cloudlet resources by taking account of adaptation characteristics, so that quality of service can be maintained for some clients.

5.1 System Model and Application Profiles

Resource allocation has been well explored in many contexts of computer systems, including operating system, networks, real-time systems, and cloud data centers. While these prior efforts can provide design blueprints for cloudlet resource allocation, the characteristics of edge-native applications emphasize unique design challenges.

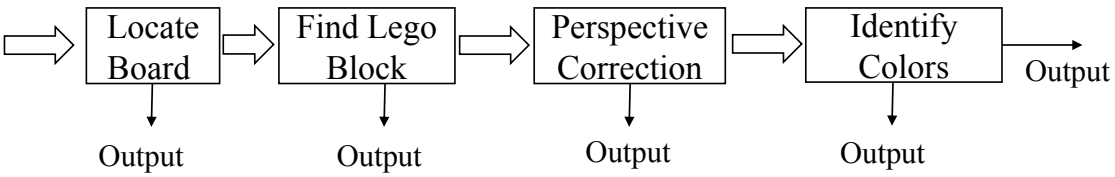


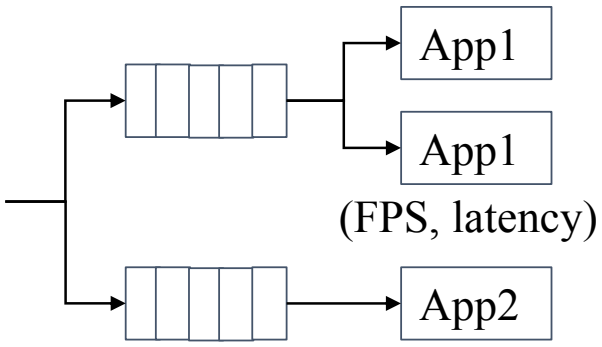
Figure 5.1: LEGO Processing DAG

The ultra-low application latency requirements of edge-native applications are at odds with large queues often used to maintain high resource utilization of scarce resources. Even buffering a small number of requests may result in end-to-end latencies that are several multiples of processing delays, hence exceeding acceptable latency thresholds. On the other hand, when using short queues, accurate estimations of throughput, processing, and networking delay are vital to enable efficient use of cloudlet resources. However, sophisticated computer vision processing represents a highly variable computational workload, even on a single stream. For example, as shown in Figure 5.1, the processing pipeline for LEGO has many exits, resulting in highly variable execution times.

To adequately provision resources for an application, one approach is to leave the burden to developers, asking them to specify and reserve a static amount of cores and memories needed for the service. However, this method is known to be highly inaccurate and typically leads to over-reservation in data-centers. For cloudlets, which are more resource constrained, such over-reservation will lead to even worse under-utilization or inequitable sharing of the available resources. Instead, we seek to create an automated resource allocation system that leverages knowledge of the application requirements and minimizes developer effort. To this end, we ask developers to provide target Quality of Service (QoS) metrics or a utility function that relates a single, easily-quantified metric (such as latency) to the quality of user experience. Building on this information, we construct offline application profiles that map multidimensional resource allocations to application QoS metrics. At runtime, we calculate a resource allocation plan to maximize a system-wide metric (e.g., total utility, fairness) specified by cloudlet owner. We choose to consider the allocation problem per application rather than per client in order to leverage statistical multiplexing among clients and multi-user optimizations (e.g., cache sharing) in an application.

5.1.1 System Model

Figure 5.2 shows the system model we consider. Each application is given a separate input queue. Each queue can feed one or more application instances, which are the units of application logic that can be replicated (e.g. a single process or several collaborative processes). Each application instance is encapsulated in a container with controlled resources. In this model, with



Only request flow is shown.

Figure 5.2: Resource Allocation System Model

adequate computational resources, clients of different applications have minimal sharing and mainly contend for the wireless network.

We use a utility-based approach to measure and compare system-wide performance under different allocation schemes. For WCA, the utility of a cloudlet response depends on both the quality of response and its QoS characteristics (e.g., end-to-end latency). The total utility of a system is the sum of all individual utilities. A common limitation of a utility-based approach is the difficulty of creating these functions. One way to ease such burden is to position an application in the taxonomy described in Section 4.4.1 and borrow from similar applications. Another way is to calculate or measure application latency bounds, such as through literature review or physics-based calculation as done in [17].

The system-wide performance is a function of the following independent variables.

(a) the number of applications and the number of clients of each application;

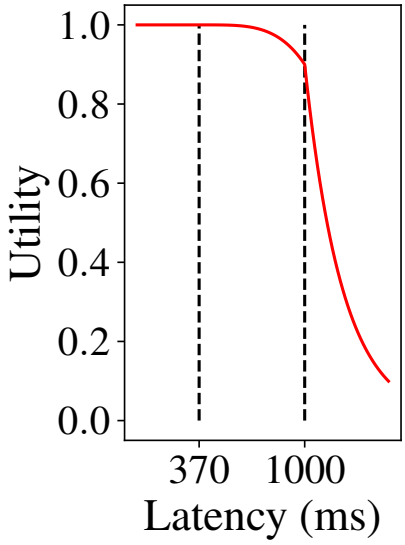
(b) the number of instances of each application;

(c) the resource allocation for each instance;

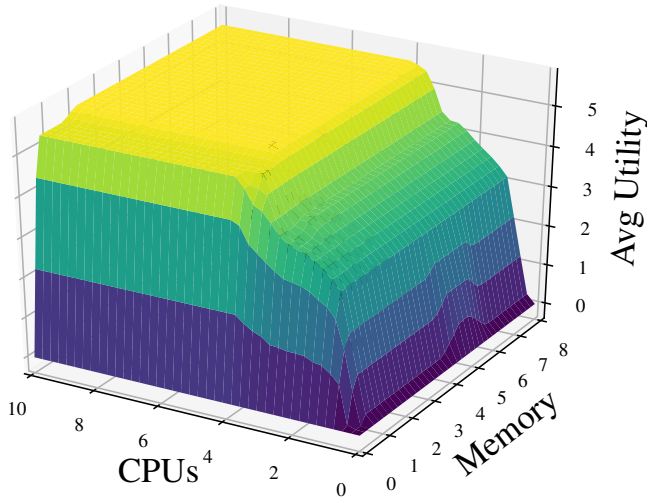
Although (a) is not under our control, Gabriel is free to adapt (b) and (c). Furthermore, to optimize system performance, it may sacrifice the performance of certain applications in favor of others. Alternatively, it may choose not to run certain applications.

5.1.2 Application Utility and Profiles

We build application profiles offline in order to estimate latency and throughput at runtime. First, we ask developers to provide a utility function that maps QoS metrics to application experience. Figure 5.3(a) and Figure 5.4(a) show utility functions for two applications based on latency bounds identified by [17] for each request. Next, we profile an application instance by running it under a discrete set of cpu and memory limitations, with a large number of input requests. We record the processing latency and throughput, and calculate the system-wide utility per unit time. We interpolate between acquired data points of (system utility, resources) to produce continuous

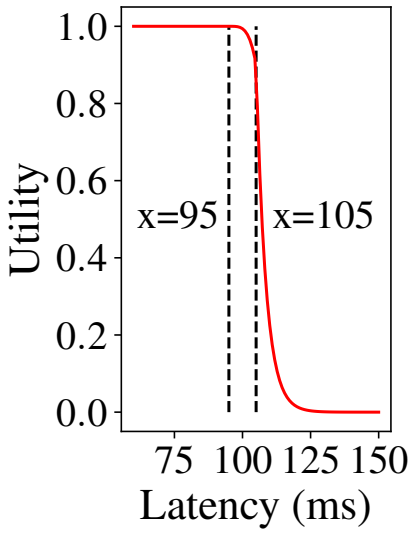


(a) Utility For FACE

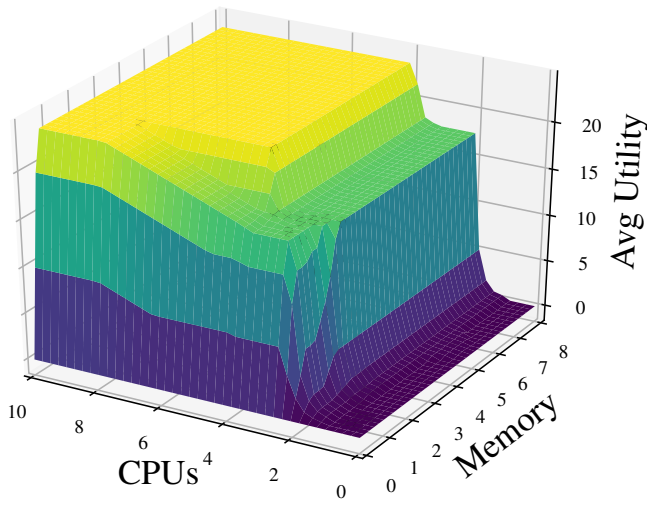


(b) Profile for FACE

Figure 5.3: FACE Application Utility and Profile



(a) Utility For POOL



(b) Profile for POOL

Figure 5.4: POOL Application Utility and Profile

functions. Hence, we effectively generate a multidimensional resource to utility profile for each application.

We make a few simplifying assumptions to ensure profile generation and allocation of resources by utility are tractable. First, we assume utility values across different applications are comparable. Furthermore, we assume utility is received on a per-frame basis, with values that are normalized between 0 and 1. Each frame that is sent, accurately processed, and replied within its latency bound receives 1, so a client running at 30 FPS under ideal conditions can receive a maximum utility of 30 per second. This clearly ignores variable utility of processing particular frames (e.g., differences between active and passive phases), but simplifies construction of profiles and modeling for resource allocation; we leave the complexities of variable utility to future work. Figure 5.3(b) and Figure 5.4(b) show the generated application profiles for FACE and POOL. We see that POOL is more efficient than FACE in using per unit resource to produce utility. If an application needs to deliver higher utility than a single instance can, our framework will automatically launch more instances of it on the cloudlet.

5.2 Profiling-based Resource Allocation

Given a workload of concurrent applications running on a cloudlet, and the number of clients requesting service from each application, our resource allocator determines how many instances to launch and how much resource (CPU cores, memory, etc.) to allocate for each application instance. We assume queueing delays are limited by the token mechanism used in Original Gabriel, which limits the number of outstanding requests on a per-client basis.

5.2.1 Maximizing Overall System Utility

As described earlier, for each application $a \in \{\text{FACE, LEGO, PING PONG, POOL, } \dots\}$, we construct a resource to utility mapping $u_a : \mathbf{r} \rightarrow \mathbb{R}$ for one instance of the application on cloudlet, where $\mathbf{r}$ is a resource vector of allocated CPU, memory, etc. We formulate the following optimization problem which maximizes the system-wide total utility, subject to a tunable maximum per-client limit:

$$\begin{aligned} \max_{\{k_a, \mathbf{r}_a\}} \quad & \sum_a k_a \cdot u_a(\mathbf{r}_a) \\ \text{s.t.} \quad & \sum_a k_a \cdot \mathbf{r}_a \preccurlyeq \hat{\mathbf{r}} \\ & 0 \preccurlyeq \mathbf{r}_a \quad \forall a \\ & k_a \cdot u_a(\mathbf{r}_a) \leq \gamma \cdot c_a \quad \forall a \\ & k_a \in \mathbb{Z} \end{aligned} \tag{5.1}$$

In above, c_a is the number of mobile clients requesting service from application a. The total

Profile applications under varying resources;
$u_a(\mathbf{r})$: resource $\mathbf{r}$ to utility mapping for application a;
for *each application* **do**
  find the highest *utility-to-resource* ratio $\frac{u_a(\mathbf{r})}{|\mathbf{r}|}$;
end
while *leftover system resource* **do**
  Find the application with the largest *utility-to-resource* $\frac{u_a(\mathbf{r}_a^*)}{|\mathbf{r}_a^*|}$, which has not been allocated resources;
  Allocate k_a application instances, each with resource $\mathbf{r}_a^*$, such that k_a is the largest integer with $k_a \cdot u_a(\mathbf{r}_a^*) \leq \gamma \cdot c_a$;
end

Figure 5.5: Iterative Allocation Algorithm to Maximize Overall System Utility

resource vector of the cloudlet is $\hat{\mathbf{r}}$. For each application a, we determine how many instances to launch — k_a, and allocate resource vector $\mathbf{r}_a$ to each of them. A tunable knob γ regulates the maximum utility allotted per application, and serves to enforce a form of partial fairness (no application can be given excessive utility, though some may still receive none). The larger γ is, the more aggressive our scheduling algorithm will be in maximizing global utility and suppressing low-utility applications. By default, we set $\gamma = 10$, which, based on our definition of utility, roughly means resources will be allocated so no more than one third of frames (from a 30FPS source) will be processed within satisfactory latency bounds for a given client.

Solving the above optimization problem is computationally difficult. We thus use an iterative greedy allocation algorithm as shown in Figure 5.5. In our implementation, we exploit the `cpu-shares` and `memory-reservation` control options of Linux Docker containers. It puts a soft limit on containers' resource utilization only when they are in contention, but allows them to use as much left-over resource as needed.

5.3 Evaluation

We use five WCA applications, including FACE, PING PONG, LEGO, POOL, and IKEA for evaluation [16, 17]. These applications are selected based on their distinct requirements and characteristics to represent the variety of WCA apps. IKEA and LEGO assist users step by step to assemble an IKEA lamp or a LEGO model. While their 2.7-second loose latency bound is less stringent than other apps, the significance of their instructions is high, as a user could not proceed without the instruction. On the other hand, users could still continue their tasks without the instructions from FACE, POOL, and PING PONG assistants. For POOL and PING PONG, the speed of an instruction is paramount to its usefulness. For example, any instruction that comes 105ms after a user action for POOL is no longer of value, because it is too late to guide the next action.

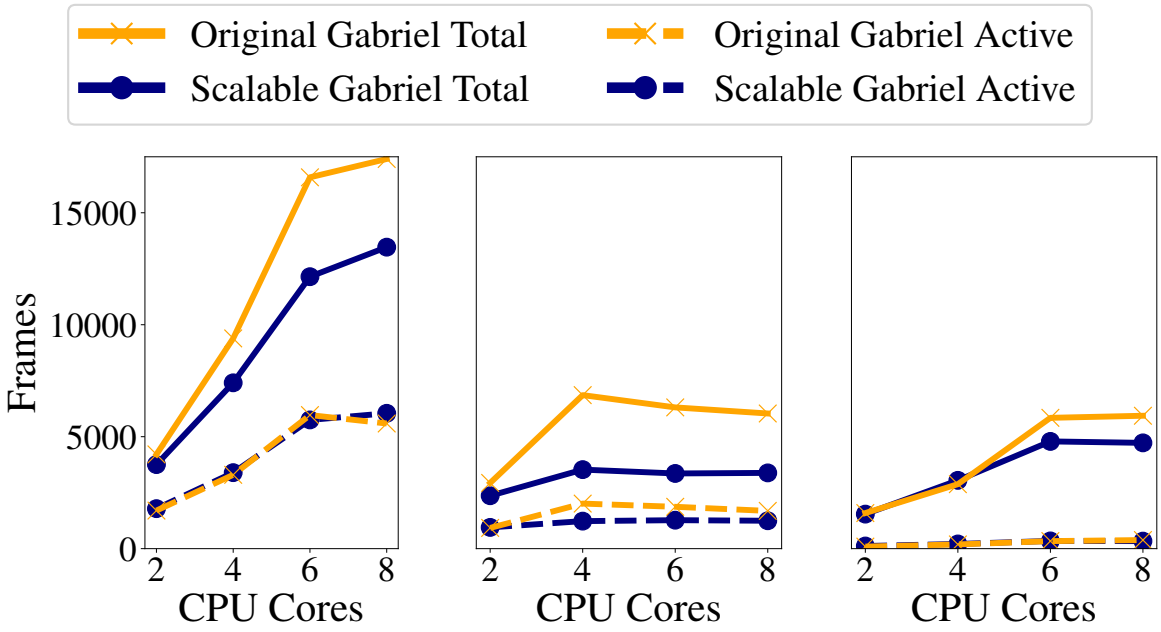


Figure 5.6: Effects of Workload Reduction

5.3.1 Effectiveness of Workload Reduction

We first evaluate the effectiveness of all of the workload reduction techniques explored in Chapter 4. For this set of experiments, we do not use multiple concurrent applications. Adaptation-centric cloudlet resource allocation is not enabled for a controlled setup. We use four Nexus 6 mobile phones as clients. They offload computation to a cloudlet over Wi-Fi links. We run PING PONG, LEGO, and POOL applications one at a time with 2, 4, 6, and 8 cores on the edge server. We constrain the number of cores available using Linux cgroup. Figure 5.6 shows the total number of frames processed with and without workload reduction. The yellow lines for Original Gabriel do not have workload reduction while the blue lines for Scalable Gabriel do. The solid lines represent the total number of frames offloaded. The dashed lines represent the number of active frames, those frames that actually contain user state information. Note that although the offered work is greatly reduced, the processed frames for active phases of the application have not been affected. Thus, we confirm that we can significantly reduce cloudlet load without affecting the critical processing needed by these applications.

5.3.2 Effectiveness of Resource Allocation

We next evaluate our adaptation-centric resource allocation mechanism on a server machine with 2 Intel® Xeon® E5-2699 v3 processors, totaling 36 physical cores running at 2.3 Ghz (turbo boost disabled) and 128 GB memory. We dedicate 8 physical cores (16 Intel® hyper threads) and 16 GB memory as cloudlet resources using cgroup. We run 8 experiments with

Exp #	Number of Clients					
	Total	FACE	LEGO	POOL	PING PONG	IKEA
1	15	3	3	3	3	3
2	20	4	4	4	4	4
3	23	5	5	4	4	5
4	25	5	5	5	5	5
5	27	5	6	6	5	5
6	30	5	7	6	6	6
7	32	5	7	7	7	6
8	40	8	8	8	8	8

Table 5.1: Resource Allocation Experiment Setup

increasing numbers of clients across four concurrent applications. The total number of clients gradually increases from 15 to 40. Table 5.1 shows the breakdown of the number of clients used for each experiment. Note that these clients are running simultaneously, resulting in heavier and heavier contention. We generate application adapation profiles offline using the method discussed in Section 5.2. We leverage these profiles to optimize for maximizing the total system utility. Figure 5.7 shows how the total system utility changes as we add more clients and hence more workload. The yellow line represents the Original Gabriel which relies on the operating system alone to divide system resources. The blue line shows our Scalable Gabriel approach. In the beginning, while the system is under-utilized, we see that the Original Gabriel yields slightly higher total utility. However, as contention increases, Original Gabriel's total utility quickly drops, eventually more than 40%, since every client contends for resources in an uncontrolled fashion. All applications suffer, but the effects of increasing latencies are vastly different among different applications. In contrast, scalable Gabriel maintains a high level of system-wide utility by differentially allocating resources to different applications based on their sensitivity captured in the adaptation profiles.

Figure 5.8 and Figure 5.9 provide insights into how scalable Gabriel strikes the balance. We present both application throughput in terms of average frames per second and latency in terms of 90%-tile response delay. Latencies are better controlled as resources are dedicated to applications with high utility, and more clients are kept within their latency bounds. Of course, with higher contention, fewer frames per second can be processed for each client. Original Gabriel degrades applications in an undifferentiated fashion. Scalable Gabriel, in contrast, tries to maintain higher throughput for some applications at the expense of the others, e.g. LEGO up to 27 clients. The accuracies of application profiles influence how well Scalable Gabriel can manage latency. Run-time resource demand could deviate from profiles due to the differences in the request content (e.g. image content). Profile inaccuracies result in the overshoot of POOL and IKEA 90%-tile latencies in Figure 5.8, as the profiles underestimate their resource demand and overestimate LEGO resource demand when the number of clients is low. When the system becomes more crowded, the throttling of LEGO throughput reduces such an effect.

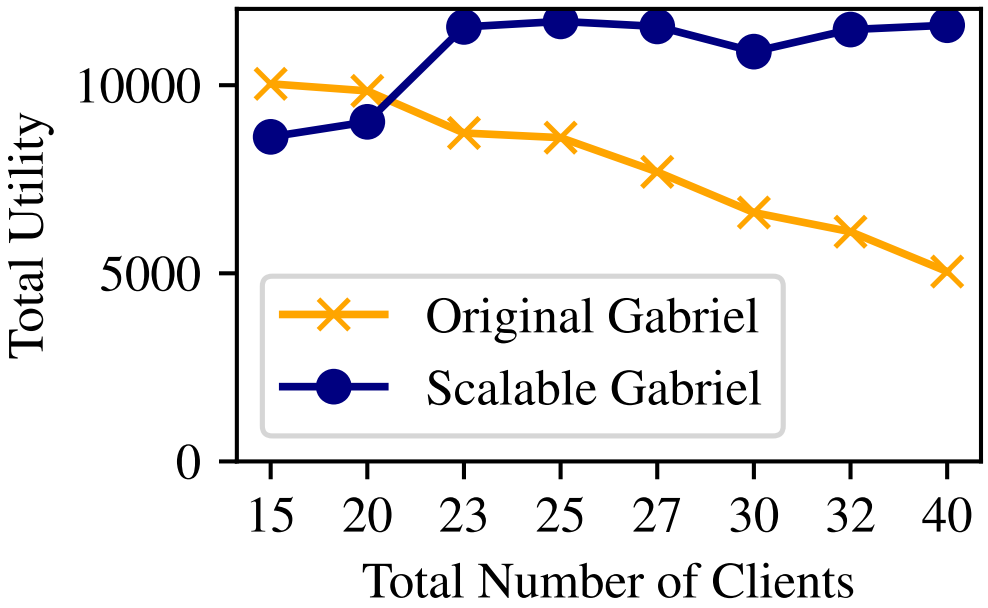
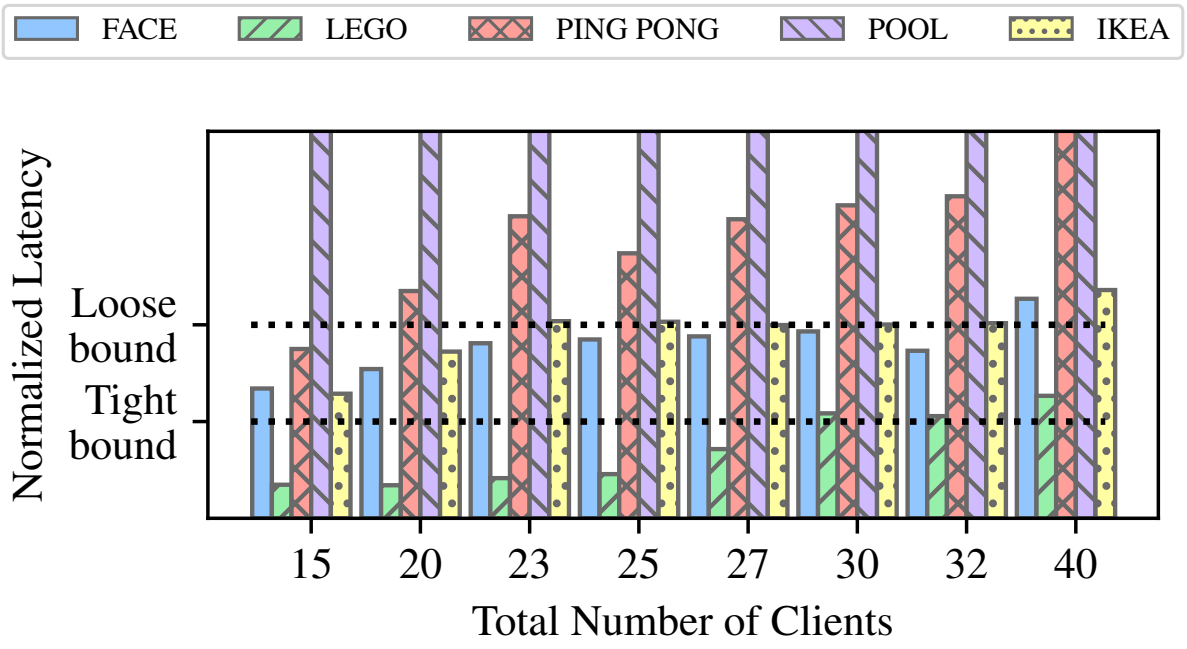


Figure 5.7: Total Utility with Increasing Contention

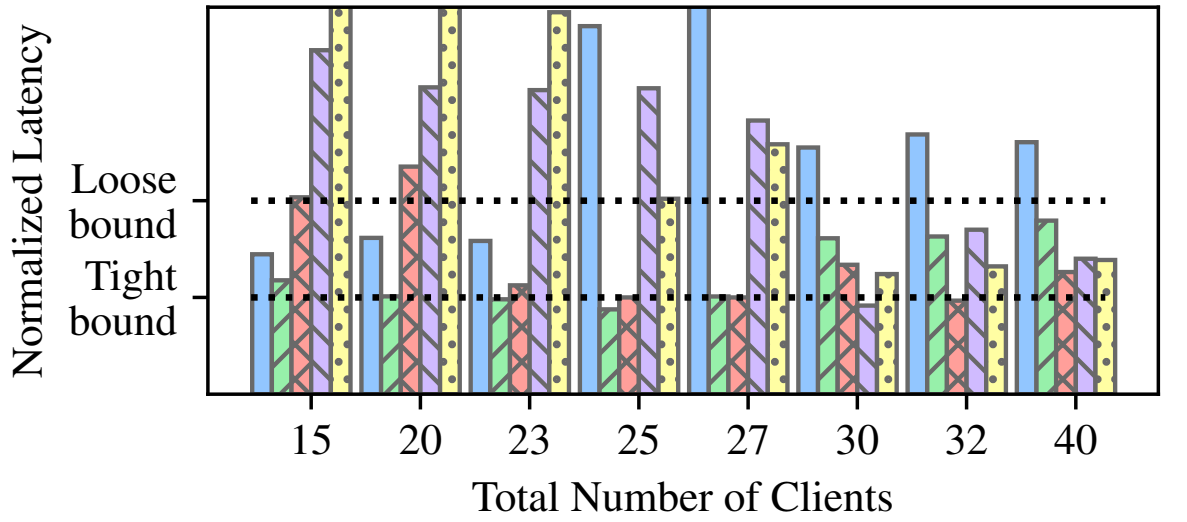
5.3.3 Effects on Guidance Latency

We next evaluate the combined effects of workload reduction and resource allocation in our system. We emulate many users running multiple applications simultaneously. All users share the same cloudlet with 8 physical cores and 16 GB memory. We conduct three experiments, with 20 (4 clients per app), 30 (6 clients per app), and 40 (8 clients per app) clients. Each client loops through pre-recorded video traces with random starting points. Figure 5.10 and Fig 5.11 show per client frame latency and FPS achieved. The first thing to notice is that concurrently utilizing both sets of techniques does not cause conflicts. In fact, they appear to be complementary and latencies remain in better control than using resource allocation alone.

The previous plots consider per request latencies. The ultimate goal of our work is to maintain user experience as much as possible and degrade it gracefully when overloaded. For WCA applications, the key measure of user experience is guidance latency, the time between the occurrence of an event and the delivery of corresponding guidance. Note that guidance latency is different than per request latency, as a guidance may need not one but several frames to recognize a user state. Figure 5.13 shows boxplots of per-application guidance latency for the concurrent application experiments above. The red dotted line denotes the application-required loose bound. It is clear that our methods control latency significantly better than the baseline. Scalable Gabriel is able to serve at least 3x number of clients when moderately loaded while continuing to serve more than half of the clients when severely loaded. In these experiments, the utility is maximized at the expense of the FACE application, which provides the least utility per resource consumed.



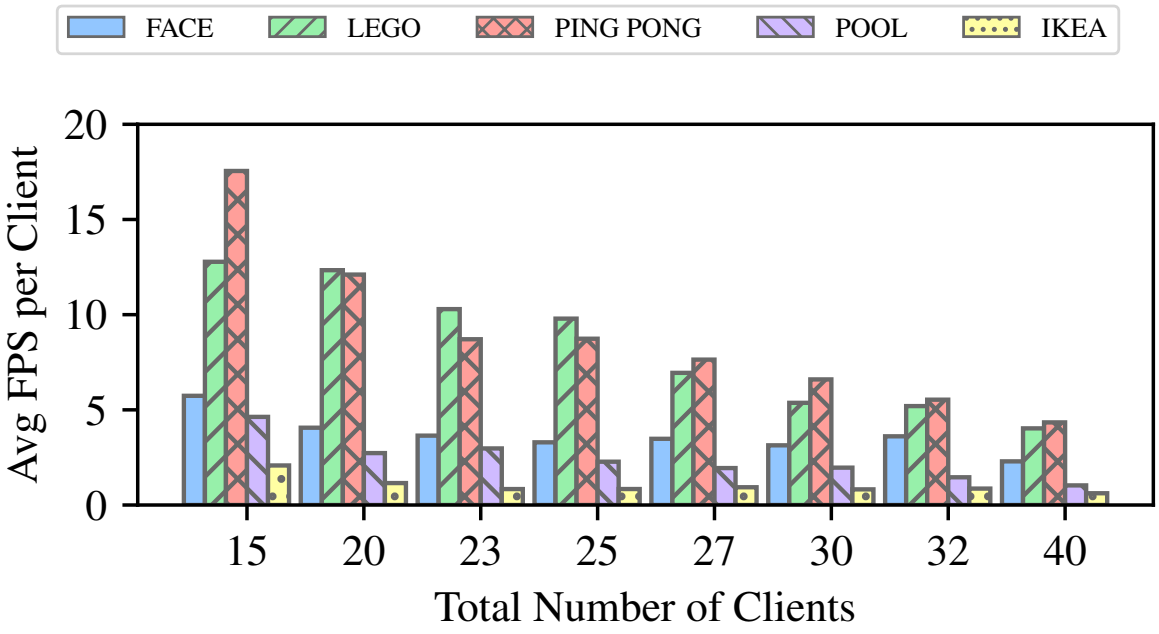
(a) Original Gabriel



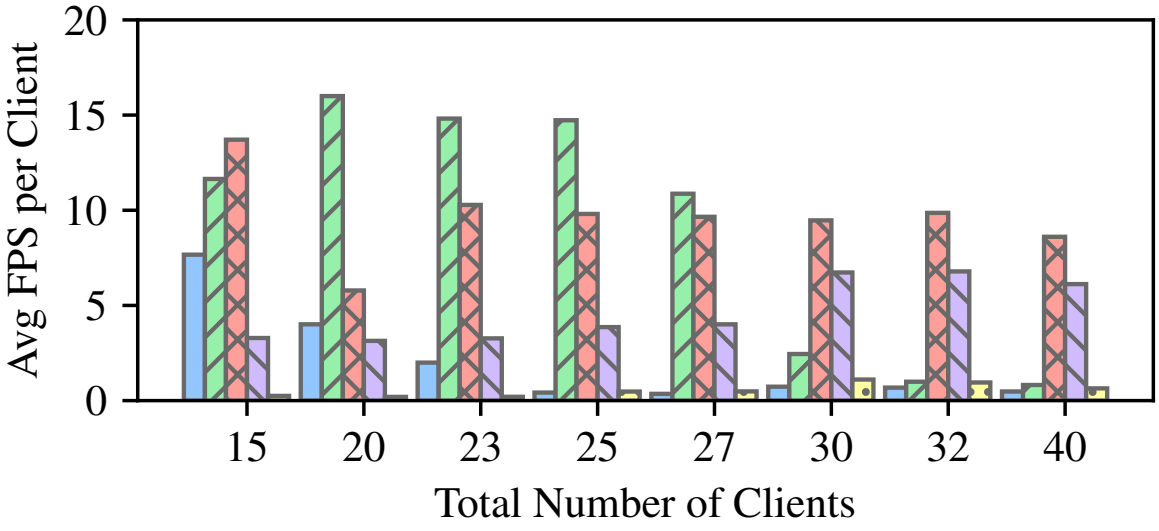
(b) Scalable Gabriel

The normalization is by per-application tight and loose bounds [17].
The allocation policy is to maximize the overall system utility.

Figure 5.8: Normalized 90%-tile Response Latency

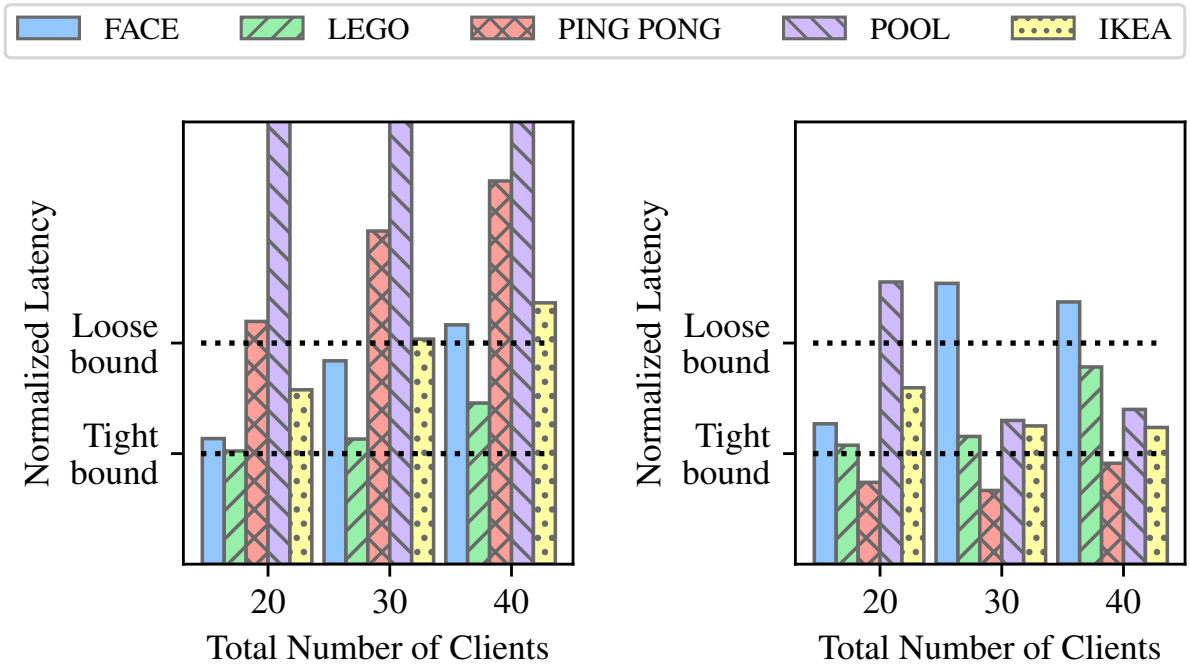


(a) Original Gabriel



(b) Scalable Gabriel

Figure 5.9: Average Processed Frames Per Second Per Client

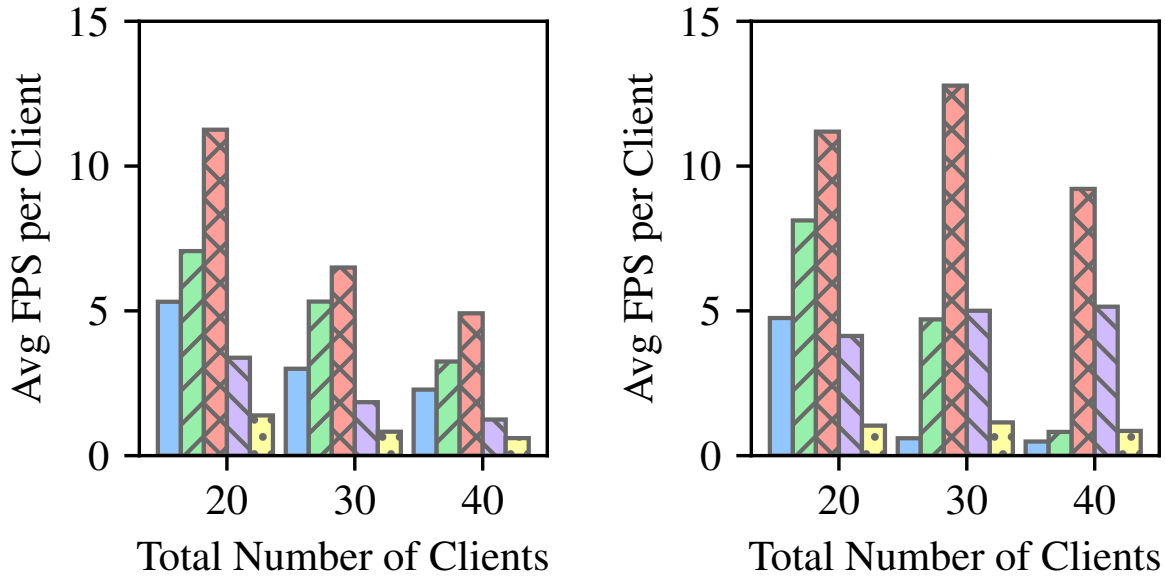


(a) Original Gabriel

(b) Scalable Gabriel

The normalization is by per-application tight and loose bounds [17].

Figure 5.10: Normalized 90%-tile Response Latency



(a) Original Gabriel

(b) Scalable Gabriel

Figure 5.11: Processed Frames Per Second Per Application

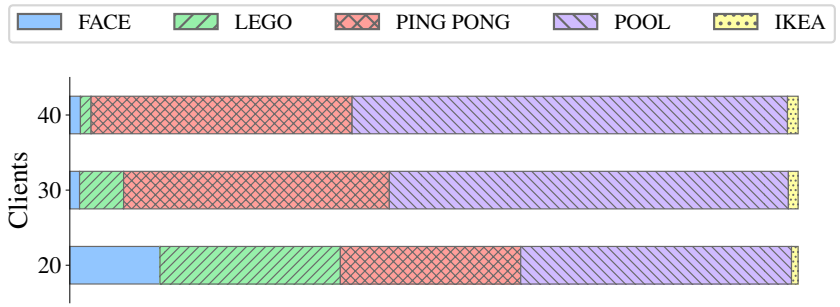


Figure 5.12: Fraction of Cloudlet Processing Allocated

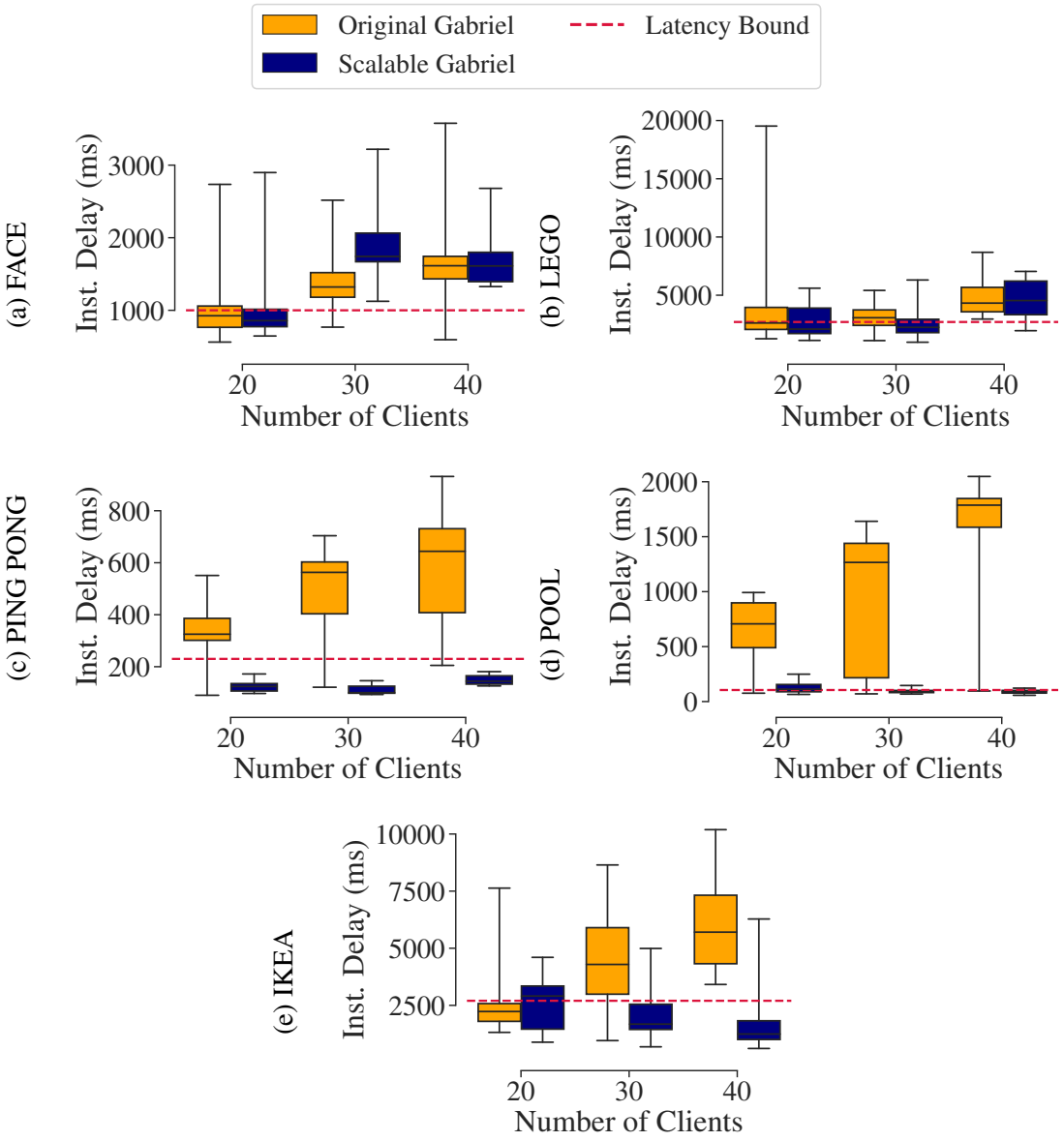


Figure 5.13: Guidance Latency Compared to Loose Latency Bound

At the highest number of clients, scalable Gabriel sacrifices the LEGO application to maintain the quality of service for PINGPONG and POOL. This differentiated allocation is reflected in Figure 5.12. In contrast, with original Gabriel, none of the applications are able to regularly meet deadlines.

5.4 Related Work

Deadline-based scheduling algorithms and mechanisms have been studied extensively in the literature. Many work [52, 106, 107, 111] exist in the context of real-time systems. Some use utility functions for scheduling tasks with soft deadlines [60, 83]. While we draw inspiration from these systems, our resource allocation focuses on application adaptation characteristics in addition to meeting latency requirements. Many mobile systems leverage application adaptation. Notably, Odyssey [75] and extensions [30] proposed upcall-based collaboration between a mobile's operating system and its applications to adapt to variable wireless connectivity and limited battery. Such adaption was purely reactive by the mobile device; in our context, adaptation for a collection of devices can be centrally managed by their cloudlet, with failover to reactive methods as needed. Exploration of tradeoffs between application fidelity and resource demand led to the concept of *multi-fidelity applications* [94]; such concepts are relevant to our work, but the critical computing resources in our setting are those of the cloudlet rather than the mobile device. More recently, [12, 24, 79, 100, 113, 118] study cluster scheduling in data-center context. These systems target a diverse range of workload, take resource reservation demands, and focus on the scalability of scheduling. In contrast, our resource allocation scheme focuses on edge-native applications, in particular, wearable cognitive assistance, and profile their characteristics for better outcomes in oversubscribed edge scenarios. Closely related, dynamic resource management in the cloud for video analytics have been explored by [32, 53, 101]. Some also leverage profile-based adaptation for more efficient video analytics resource management [47, 50, 122]. However, most of these systems focus on throughput-oriented analytics application on large clusters. In contrast, we target interactive performance on relatively small edge deployments.

5.5 Chapter Summary and Discussion

More than a decade ago, the emergence of cloud computing led to the realization that applications had to be written in a certain way to take full advantage of elasticity of the cloud. This led to the concept of "cloud-native applications" whose scale-out capabilities are well matched to the cloud. The emergence of edge computing leads to another inflection point in application design. As last two chapters have shown, edge-native applications have to be written in a way that is very different from cloud-native applications if they are to be scalable. We explore client workload reduction and server resource allocation to manage application quality of service in the face of contention for cloudlet resources. We demonstrate that our system is able to ensure that in overloaded situations, a subset of users are still served with good quality of service rather than equally sharing resources and missing latency requirements for all.

Chapter 6

Wearable Cognitive Assistance Development Tools

While previous chapters address the system challenges of scaling wearable cognitive assistance at the edge, another key obstacle to the widespread adoption of WCAs is the level of specialized skills and the long development time needed to create new applications. Such expertise includes task domain knowledge, networked application development skills, and computer vision insights. Researchers [16] report that it typically takes multiple person-months of effort to create a single application. The majority of development time is spent on learning new computer vision techniques, selecting robust algorithms to use through trial and error, and implementing the application. The high barrier to entry significantly limits the number of wearable cognitive assistants available today. Clearly, this is not scalable.

In this chapter, we reflect on the existing ad hoc WCA development process, propose a new development methodology centered around DNN-based object detection, and present development tools that we have built to lower the barrier of WCA development. Our goal is to simplify the process so that a small team (1-2 people) of a task expert and a developer, without computer vision expertise, can create an initial version of a Gabriel application within a few days. This is a productivity improvement of at least one order of magnitude. Refinement and tuning may take longer, but can be guided by the early use of the application.

Simplification is difficult. The application needs a precise definition of the end-point state of each step (e.g., a particular screw mounted into a workpiece), yet needs to be tolerant of alternative paths to reaching that end point (e.g., hand-tightening versus using a screwdriver). We assume the small development team has knowledge of the task specifics and general programming skills, but not necessarily experience with wearable devices or computer vision.

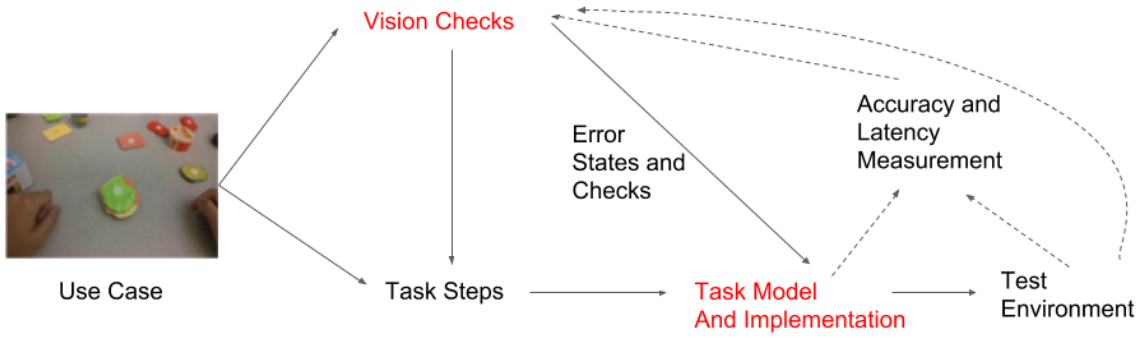


Figure 6.1: Ad Hoc Development Workflow

6.1 Ad Hoc WCA Development Process

The existing ad hoc development process of WCAs can be described as figure 6.1. Developers first work with the task expert to identify and pinpoint a specific use case. With the use case in mind, the development team needs to identify critical visual features and states that can be reliably recognized with machine vision. In the meantime, the use case is broken down into individual steps. For each step, feedback messages to users are created based on detected visual states. Potential errors are also enumerated and included as additional steps. We refer to the complete set of the steps annotated with visual states and feedback as the *Task Model*. For example, for the LEGO wearable cognitive assistance [17], the task model contains the sequence of blocks to build up the final assembled Lego piece, together with potentially misused blocks. The visual states to recognize are the pieces currently on the lego board: the shape and the color of individual lego block, and their composed shape and color.

Notably, a task model is not only determined by the task itself, but also influenced heavily by visual states that can be reliably detected. In fact, it is common to alter the sequence of steps or introduce additional steps to reduce the difficulties and improve the reliability of computer vision recognition. Since there is a human in the loop (the user), relying on humans to do what they are good at is the main reason that wearable cognitive assistance can be implemented without solving the generic perception and planning problems intrinsic to robotics. For example, a frequently used technique is to ask the user to hold the object of interests at a fixed viewpoint shown in the image guidance. Narrowing the viewing angle makes the recognition problem more tractable.

The application task model serves as a blueprint for implementation. For each step in the task model, developers select and implement computer vision algorithms. Custom logic is also written to handle step transitions and interface with the Gabriel platform. After initial implementation, test runs are conducted to evaluate the robustness of computer vision and measure the end-to-end application latency. The implementation process is typically iterative to allow trials

and errors when choosing computer vision algorithms.

The ad hoc development process has unnecessary high requirements and burden for WCA creators in the following aspects.

1. Developers need to be familiar with various computer vision algorithms to determine what visual states can be recognized reliably. The knowledge of selecting algorithms for specific objects, environments, and setups requires years of experience. This high bar of entry hinders the creation of WCAs.
2. It takes a significant amount of time to implement computer vision algorithms in a trial and error fashion. Several months of coding time could turn into a fruitless outcome when the algorithm turns out to be a bad fit. For sophisticated WCAs with tens or hundreds of visual states, such a trial and error methodology is not scalable. Needed are unified and automated methods to create visual state recognizers.
3. The tight coupling of task model design and visual state recognition calls for deep understanding in both task domain and computer vision. When visual states in the task model end up being too difficult for even the state-of-art algorithms, the development team needs to adjust the task model to either use alternative visual states or employ user assistance. For instance, in the RibLoc application, during development, a task step was changed to asking the user to read out a word on the gauge instead of performing optical character recognition on the lightly-colored characters. Close collaborations among task experts and developers are needed due to this tight intertwinement between task model design and computer vision algorithm exploration. Methodologies and tools that shorten the turn-around time when a task model changes can significantly reduce the overall development time.

6.2 A Fast Prototyping Methodology

To streamline the development process and lower the barrier of entry, we propose a new prototype methodology which focuses on the following two aspects.

1. Use deep neural network (DNN) based object detection as the universal building block for visual state recognition.
2. Use finite state machine to represent and implement application task model.

We first introduce the concepts of this methodology and describe its benefits in this section and then introduce tools we have built to automate the development process in subsequent sections.

6.2.1 Objects as the Universal Building Blocks

To replace manual selection of computer vision algorithms through trial and error, we propose to use DNN-based object detection as the fundamental building block to detect user states. Using object detection as the universal building block means that each visual state should be decomposed and described as a collection of individual objects along with their attributes and rela-

Figure 6.2: A Frame Seen by the PING PONG Assistant

tionships. We argue that objects as fundamental building blocks provide expressive and flexible constructions capable of describing a wide range of visual user states important to WCAs. Intuitively, the addition and removal of objects from a scene is straightforward to express. Spatial relationships among objects (e.g. close to, on the left of, overlap significantly) capture semantic visual features that can be used to infer user states. For example, the detection of two Ikea furniture pieces close to each other could indicate that they have been assembled together. Moreover, temporal relationships can add confidence to the inference made from spatial relationship. The detection of two pieces of furniture closely connected to each other in the last 30 frames implies strongly that they have been assembled together. Of course, there are limitations on how reliable the inference could be. Nonetheless, objects together with their spatial and temporal relationships with one another, provide a flexible reasoning framework to decompose user states into visual states.

To illustrate the effectiveness of this object centered approach, we showcase how existing applications built without this methodology in mind can be easily expressed. Figure 6.2 shows an image that will trigger the PING PONG assistance to provide an instruction to hit the ball to the left. We can express this visual state with a collection of objects, namely the Ping-Pong table, the ball, and the opponent. These objects should satisfy the following spatial relationship for the instruction "hit to the LEFT" to be generated. To improve the accuracy of our instruction, we can require the spatial relationship of these objects to be held true for at least two consecutive frames before an instruction is produced.

- The Ping-Pong ball is above the Ping-Pong table for an active rally.
- The opponent is on the right side of the Ping-Pong table.
- The ball is also on the right side of the Ping-Pong table.

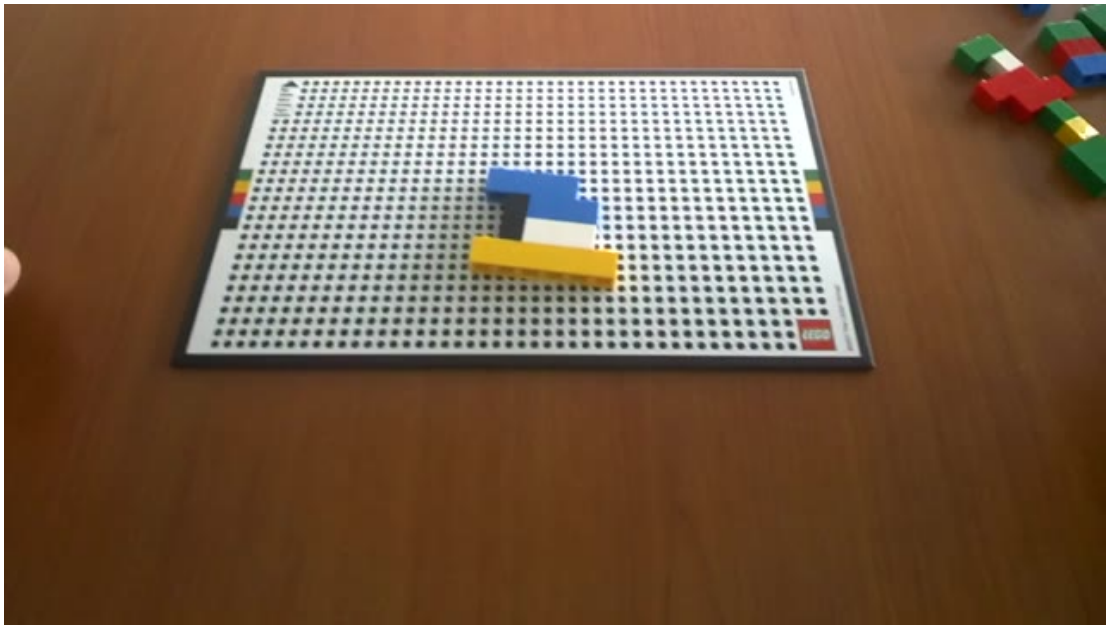


Figure 6.3: A Frame Seen by the LEGO Assistant

Similarly, we can express the user states using objects for another application LEGO. Figure 6.3 shows a visual state for the assembled pieces. Here the collection of objects needs to be present are two blue Lego blocks, two black blocks, one white block, and one yellow block. The spatial relationships among these block include:

- The yellow block should be at the bottom and longer than all other blocks.
- The white block should be in the middle of a blue block and the yellow block. It should have the same width as the blue block.
- Two black blocks should be on the left of a blue block and the white block. It should also be on the bottom of a blue block.
- One blue bock should be at the top of all other pieces.

One challenge with object detection is to obtain the absolute scale. In this example, it would be difficult to get the exact width of the lego block (e.g. whether it is a 1x4 or 1x3 lego piece) with object detection alone. However, once the location of object is detected, additional processing can be used to identify object attributes. For example, we can leverage the dots on the lego board to calculate the absolute size of lego pieces.

The key benefit of using object detection as the universal unit to recognize visual states is the possibility of automation. In recent years, deep neural networks have dramatically increased the accuracy of object detection [125]. In 2008, the best object detector, based on deformable part model [29], achieved a mean average precision (mAP) of 21% on the Pascal Visual Object Classes dataset [28]. In less than ten years, deep neural networks [38, 39, 61, 85] have quadrupled the accuracy to 84% on the same dataset. For a wide range of real-world objects, DNNs have shown to be effective in both accurate identification of classes and localization of

object bounding boxes. They can even differentiate subtly different classes, such as breeds of dogs. Many commercial products now offer features using DNN object detection. For example, Google Photos [33] and Pinterest Visual Search [48] allow users to search for images containing object of interest using text.

In addition to significant improvement in accuracy, DNNs also provide a unified method to create object detectors. Modern DNN-based object detection employs an end-to-end learning approach, in which one or multiple deep neural networks are trained to predict the object classes and locations directly from RGB images. Gone is the need to hand craft features. Instead, DNNs, on their own, find distinguishable features during the training process as they are presented with labeled examples of different objects. The substitution of custom CV with machine-learned models makes automation possible. A typical DNN training process consists of the following steps.

1. Collect images of object of interests from diverse background, perspectives, and lighting conditions.
2. Annotate these images with bounding boxes and class names to create a dataset of training data, evaluation data, and test data.
3. Implement a DNN-based object detection network, typically using machine learning frameworks, such as Tensorflow [6] and PyTorch [80].
4. Continuously train and evaluate a DNN model using the labeled dataset.
5. Test the accuracy of the trained DNN model on the test data.

While a unified training method eliminates manual feature engineering, creating a DNN-based object detector is still both time-consuming and painstaking for the following reasons. First, DNNs often have millions of model parameters and therefore requires millions of labeled examples to train from scratch. Collecting and labeling such large amount of training data takes significant manual labor. Second, implementing a DNN correctly for object detection is not trivial and still requires significant amount of knowledge of machine learning. For example, state-of-art object detectors uses custom layers different from the standard convolutional layer for better performance. Data augmentation, batch normalization, and drop out are needed at training time for better results through optimization, but should be disabled during inference time. To streamline the process of creating DNN-based object detectors, we provide a web application OpenTPOD (Section 6.3) that hides the implementation details from the developers and allow them to train a DNN object detector from a web browser without writing a single line of code.

6.2.2 Finite State Machine (FSM) as Application Representation

While the Gabriel platform handles data acquisition and transmission from mobile wearable devices to a cloudlet, application developers still need to write custom business logic for their own use cases. Since the task model can change frequently during development, a fast turn-around time to implement changes can reduce the overall development time. Therefore, we

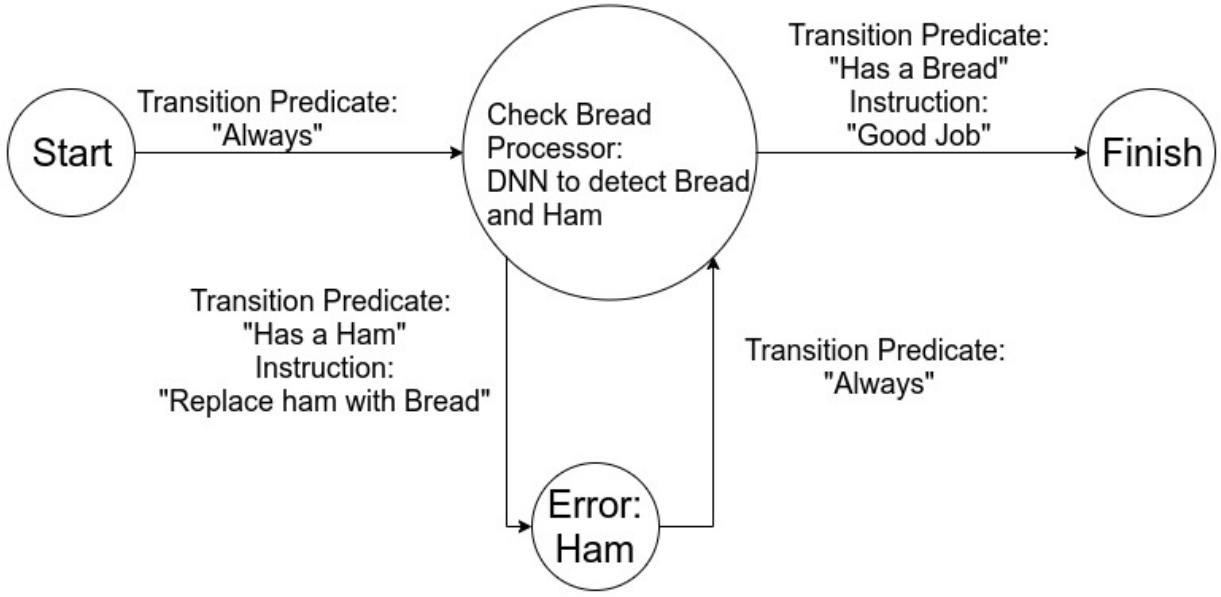


Figure 6.4: Example WCA FSM

propose a finite state machine representation of application logic and a GUI tool to enable fast turn-around.

Our choice of finite state machine representation is based on the observation that WCAs, in their nature, consist of states. A FSM State corresponds to a user state at a particular time instance. At a given state, the WCA runs computer vision processing specific to the state to analyze the current scene. We refer to the computer vision algorithms as Processors for the state. The outputs of Processors are symbolic states of current scene. For example, a symbolic state can be a vector of objects. Transitions to other states happen when the symbolic states meet some criteria, for example, the presence of a new object. We refer to these criteria as Predicates for transitions. A State can have multiple transitions into different adjacent States. Each Transition has its own Predicate. Transitions also have the concept of precedence. The first Transition whose Predicate is satisfied is triggered for a state change. In addition to predicates, Transitions also have guidance instructions. When a Transition is taken, the corresponding guidance instruction is delivered to the user.

Figure 6.4 shows an example of a simple WCA represented as a finite state machine. This example WCA checks whether a piece of bread is present. If so, it congratulates the user. On the other hand, if it detects a piece of ham is present, a corrective guidance is sent to the user. There are four states in total. The application starts from the “Start” state and immediately transitions to the “Check Bread” state as the predicate is “Always”. At the “Check Bread” state, for each frame, a DNN to detect bread and ham is run to extract the symbolic state. Then, every transition is checked to see if its predicate is satisfied. If a “Bread” is detected, the transition predicate to the “Finish” state is satisfied and hence the transition taken. The corresponding instruction “Good Job” is delivered to the user. Similarly, when in “Check Bread” state, if a “Ham” is detected, the transition to the “Error: Ham” state is satisfied and taken. The instruction “Replace Ham with

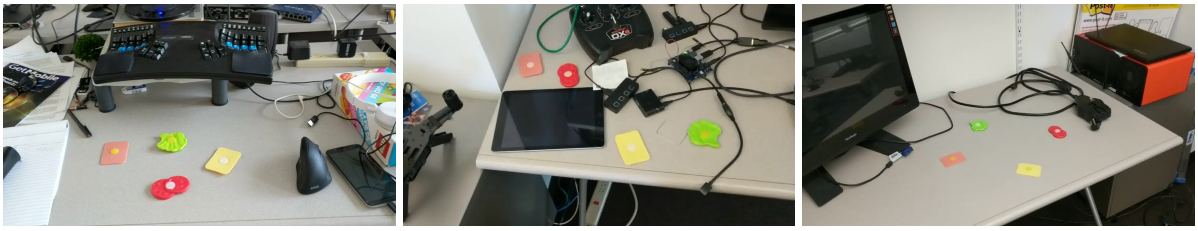


Figure 6.5: OpenTPOD Training Images Examples

Bread" would be delivered as the corrective guidance to the user.

With an FSM representing custom application logic, we impose structure on the the WCA application. We provide OpenWorkflow (Section 6.4), which consists of a python API and a web GUI, to enable developers to quickly create a FSM-based cognitive assistant.

6.3 OpenTPOD: Open Toolkit for Painless Object Detection

Training a DNN for object detection is not trivial. In particular, it involves constructing a correctly-labeled training data set with millions of positive and negative examples. The training process itself may take days to complete, and requires a set of arcane procedures to ensure both convergence and efficacy of the model. Fortunately, one does not typically need to train a DNN from scratch. Rather, pretrained models based on public image data sets such as ImageNet are publicly available. Developers can adapt these pretrained models to detect custom classes for new applications, through a process called *transfer learning*. The key assumption of transfer learning is that much of the training that teaches the model to discover low-level features, such as edges, textures, shapes, and patterns that are useful in distinguishing objects can be reused. Thus, adapting a pretrained model for new object classes requires only thousands or tens of thousands of examples and hours of training time.

However, even with transfer learning, collecting a labeled training set of several thousand examples per object class can be a daunting and painful task. In addition, implementing object detection DNNs itself requires expertise and takes time. OpenTPOD is a web-based tool we developed to streamline the process of creating DNN-based object detectors for fast prototyping. It provides a tracking-assisted labeling interface for speedy annotation and a DNN training and evaluation portal that leverages transfer learning to hide the nuances of DNN creation. It greatly reduces the labeling effort while constructing a dataset, and automates training an object detection DNN model.

6.3.1 Creating a Object Detector with OpenTPOD

Using OpenTPOD to create object detectors is straight-forward. A developer first captures videos of objects from various viewing angles and diverse backgrounds. For example, these videos can be acquired on a mobile phone. At 30 frames per second, ten minutes of video footage contain

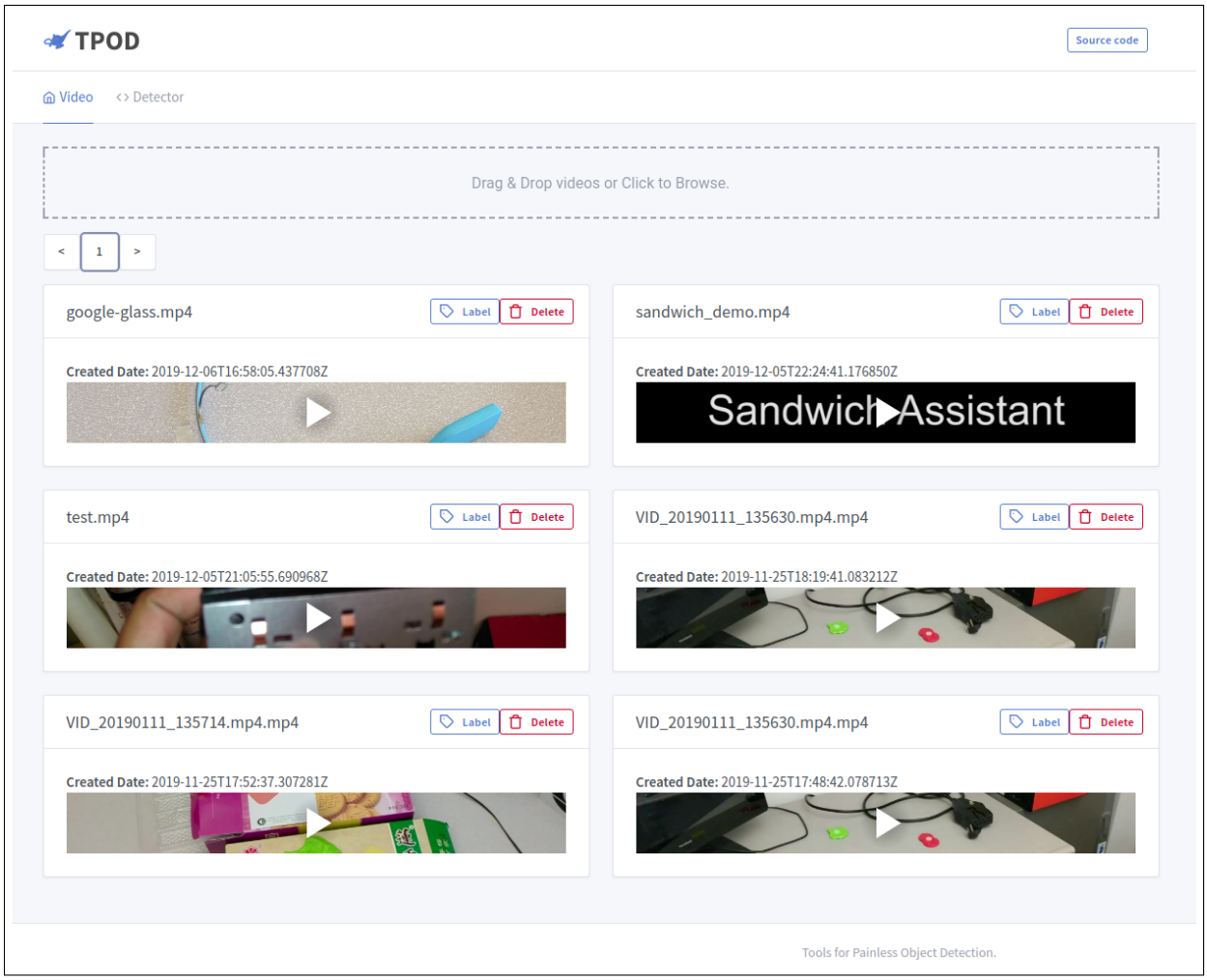


Figure 6.6: OpenTPOD Video Management GUI

18000 frames, which is already a reasonable amount of training data for transfer learning. Larger amounts of training data typically help increase the accuracy, although diminishing returns apply. Moreover, it is preferred to collect training examples in environment similar to the test environment to make the training examples exhibit similar distribution as test data. Figure 6.5 shows three example training images for a toy cooking set. Note that the image background is randomly cropped to be used as negative examples. These background examples illustrate to the neural network what an object of interest does *not* look like. Therefore, it is recommended that the background contains common objects in the test environment that may cause confusion.

Next, developers upload the collected training videos to OpenTPOD either directly from the phone or from a computer. As shown in Figure 6.6, OpenTPOD helps user to store and organize training videos. In addition, OpenTPOD assists users to annotate the training videos by integrating an external labeling tool into the GUI. Uers can annotate objects by draw bounding boxes on the uploaded video frames. To facilitate the annotation process, OpenTPOD leverages the fact that the individual frames come from a continuous video shoot. It automatically generates

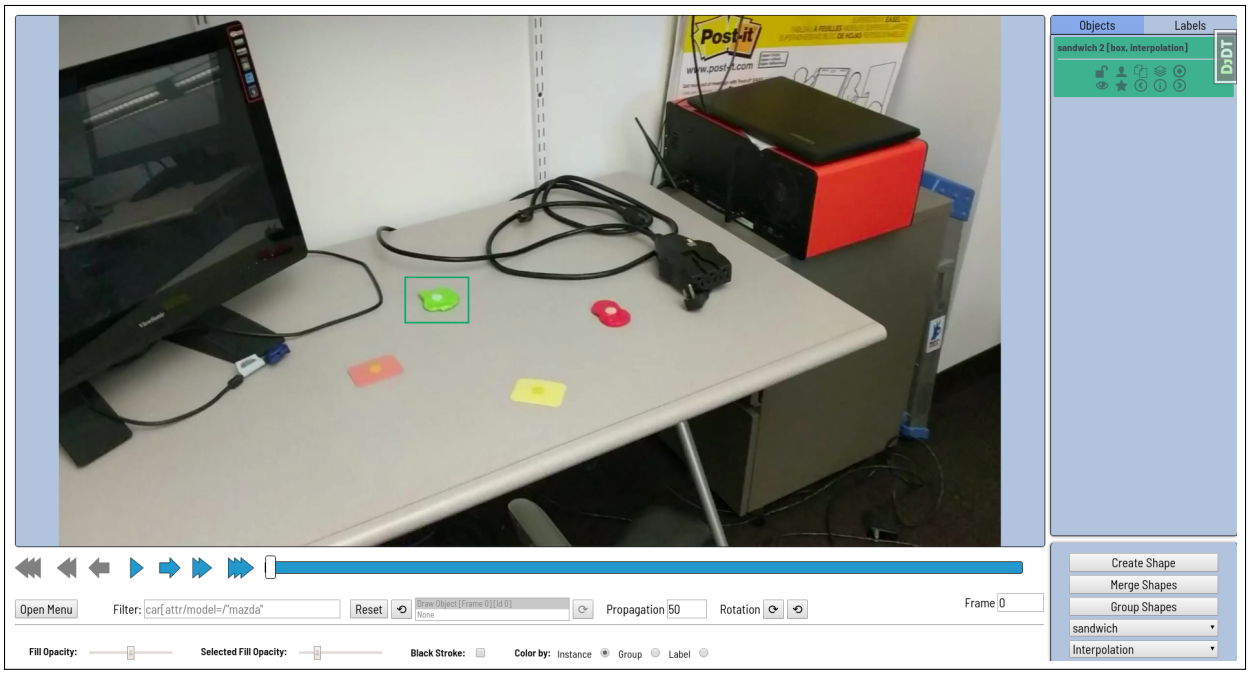


Figure 6.7: OpenTPOD Integration of an External Labeling Tool

bounding box in subsequent frames either using a correlation tracking algorithm [22] or linear interpolation. As a result, users only need to label a few key frames in a video with the rest of frames auto-populated with generated labels. Of course, tracking is not perfect, and the bounding box may drift off over time. In this case, the user can forward or rewind the video to adjust the bounding box. The adjustment of bounding boxes re-initializes the tracking for subsequent frames. From our experience usage, this approach of labeling followed by tracking can reduce the number of frames that the user needs to manually label by a factor of 10-20x.

With videos annotated, users can move on to create a training dataset on the webpage by selecting videos to use. OpenTPOD performs a data cleaning and augmentation pass to prepare a high quality training set. Due to inter-frame correlation, adjacent frames within a video may appear to be substantially similar or identical. These highly correlated frames violate the independent and identically distributed assumption of training data for DNNs. OpenTPOD eliminates these near duplicate examples. Optionally, data augmentation can be employed. It adds synthetic images, created by cutting and pasting the object of interests on varying backgrounds, at different scales and rotations. Such augmentation helps produce more robust object detectors [27].

Then, by submitting a form on the web GUI, users can launch the DNN training process without writing a single line of code. Under the hood, OpenTPOD automates the transfer learning process. OpenTPOD supports several state-of-the-art object detection networks, including FasterRCNN-ResNet [39, 85] and SSD-MobileNet [41, 62]. These different networks exhibit varying accuracy versus speed trade-offs. While FasterRCNN-ResNet achieves higher accuracies on standard datasets, its training and inference time are significantly longer than SSD-MobileNet. Application developers should choose the appropriate DNN network based on their

accuracy and latency constraints. Negative examples are mined from the video background; these are parts of the frames not included in the object bounding boxes. The training process generates downloadable Tensorflow models in the end. The OpenTPOD web GUI provides training monitoring through Tensorflow visualization library Tensorboard [110]. OpenTPOD also provides a container image that can serve the downloaded model files for inference.

Overall, OpenTPOD can greatly reduce both the labeling effort and in-depth machine learning knowledge required to train and deploy a DNN-based detector. The OpenTPOD demo video can be found at `https://youtu.be/UHnNLrD6jTo`. OpenTPOD has been used by tens of researchers and students to build wearable cognitive assistance. For example, a group of master students in CMU mobile and pervasive computing class successfully used OpenTPOD to build an assistant for using Automated External Defibrillator (AED) machines.

6.3.2 OpenTPOD Case Study With the COOKING Application

To quantify how much OpenTPOD can help reduce labeling efforts, we conducted a case study to train detectors for the COOKING cognitive assistant [16]. We trained object detectors for 9 components of a toy sandwich, including bread, ham, lettuce, cucumber, cheese, half sandwich, wrong ham, tomato, and full sandwich. We collected 53 short training videos, annotated all of them, and trained object detectors with OpenTPOD.

In total, we collected 21218 video frames. In contrast, we only manually labeled 91 frames on OpenTPOD annotation interface. The rest of frames are automatically labeled by tracking. The entire labeling session took 80 minutes. The total number of frames used for training is 21013 due to training set optimization by OpenTPOD. We fine-tuned from a FasterRCNN-VGG network. The transfer learning process took 54 minutes to finish on a NVIDIA K40c GPU.

6.3.3 OpenTPOD Architecture

OpenTPOD adopts a Model-View-Controller (MVC) design [55]. Figure 6.8 shows the OpenTPOD architecture. The OpenTPOD frontend handles UI logic and is built using React, a declarative, efficient, and flexible JavaScript library for creating user interfaces [105]. React enables OpenTPOD frontend code to be modular and reusable. The frontend has three major components: video management, detector management, and an external labeling tool. OpenTPOD integrates the labeling tool CVAT [78] into its frontend UI through HTML iframe embedding. The integration of an labeling tool into the OpenTPOD GUI unifies the workflow. Users no longer need to install and set up another piece of software. The OpenTPOD frontend communicates with the backend through a set of RESTful APIs [86].

OpenTPOD backend is developed using the Django web framework [40] and served with the Nginx web server [73] and the Gunicorn gateway [34]. The backend implements RESTful apis to create, read, update, and delete *Video* and *Detector* resources. It also interfaces with the backend of the external labeling tool to read and write annotations. Long running background jobs, such as DNN model training and video extraction, are processed asynchronously using a task queue

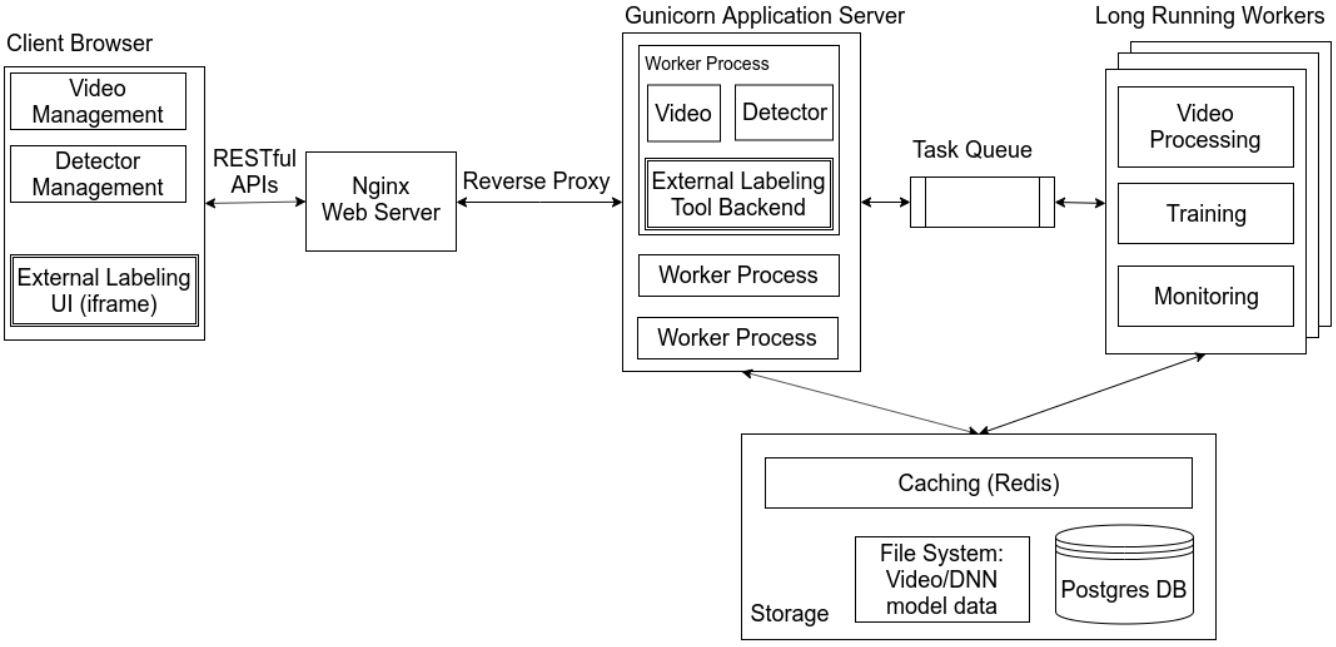


Figure 6.8: OpenTPOD Architecture

and worker processes. Metadata about videos and DNN models are stored in a relational database while large binary files, such as video files and trained DNN model weights, are stored in the file system. An in-memory caching layer on top of the file system and database is created using the Redis key-value store. The entire application is containerized and can be deployed easily onto bare-metal or virtualized environments.

One key design choice of OpenTPOD is to be extensible for new DNN architectures. OpenTPOD provides a standard interface, shown in Figure 6.9, in order to easily add new DNNs implementation for transfer learning. OpenTPOD refers to these DNN implementations as DNN *Providers*. Without modifying other components, providers can be added and integrated by implement this interface. The built-in *Provider* is the Tensorflow Object Detection API [109], which supports transfer learning for a wide set of object detectors.

property **training_configurations**: user customizable hyperparameters;
init(dnn_type, training_set, output_directory): initialization;
train(training_configurations): launch training;
export(): export trained models;
visualization(): monitor training process;

Figure 6.9: OpenTPOD Provider Interface

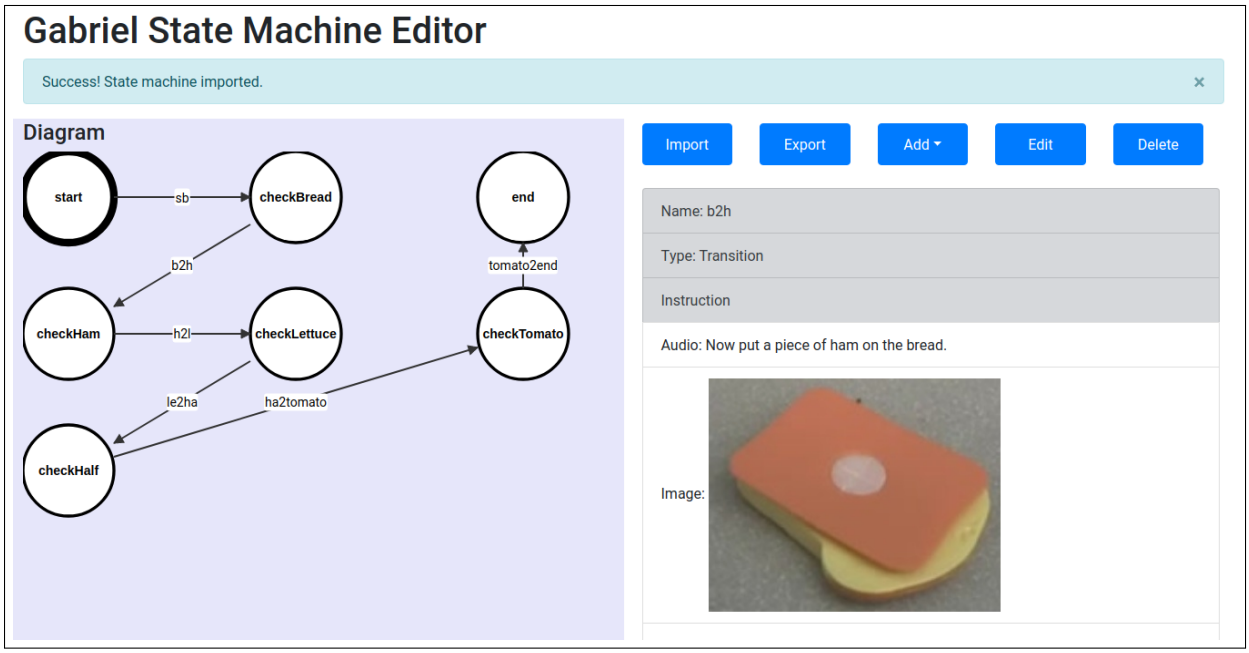


Figure 6.10: OpenWorkflow Web GUI

6.4 OpenWorkflow: FSM Workflow Authoring Tools

In addition to creating DNN-based object detectors, developers need to write custom logic to implement the WCA task model running on the cloudlet. In this section, we introduce OpenWorkflow, an FSM authoring tool that provides a Python library and a GUI to enable fast implementation to allow for quick development iteration.

As discussed in Section 6.2.2, the WCA cloudlet logic can be represented as a finite state machine. The FSM representation allows us to impose structure and provide tools for task model implementation. OpenWorkflow consists of a web GUI that allows users to visualize and edit a WCA FSM within a browser, a python library that supports the creation and execution of a FSM, and a binary file format that efficiently stores the FSM. The OpenWorkflow video demo can be found at `https://youtu.be/L9ugONLpnwc`.

6.4.1 OpenWorkflow Web GUI

Figure 6.10 shows the OpenWorkflow web GUI. Users can create a WCA FSM from the GUI by editing states and transitions. State processors, e.g. the computer vision processing logic to run in a given state, can be specified by a container url. User guidance can be added through adding text, video urls or by uploading images. The Web GUI also supports import and export functionalities to interface with other WCA tools. The exported FSM is in a custom binary format and can be executed by the OpenWorkflow Python library.

The GUI is implemented as a pure browser-based user interface, using React [105]. No web

backend is needed. This makes the GUI easy to set up and deploy. The user only needs to open an HTML file in a browser to use the tool.

6.4.2 OpenWorkflow Python Library

Another way to programmatically create a FSM is through the OpenWorkflow Python library. The library provides python APIs to create and modify FSMs. The python APIs provides additional interfaces to add custom computer vision processing as functions and ad hoc transition predicates for customization.

In addition, the Python library provides a state machine executor that takes a WCA FSM (e.g. made with the Web GUI) and launches a WCA program using the Gabriel platform. The program is then ready to be connected by Gabriel Android Client. The WCA program follows the logic defined in the state machine. A Jupyter Notebook [54] is also provided to make it possible to launch the program from a browser. This library has been made available on The Python Package Index for easy installation.

6.4.3 OpenWorkflow Binary Format

We define a custom FSM binary format for WCAs that can be read and wrote using multiple programming languages. We use the serialization library Protocol Buffers to generate language-specific stub code. Figure 6.11 shows a summary of the serialization format.

6.5 Lessons for Practitioners

While the methodology and the suite of tools provided in this Chapter offer a recipe to follow when creating new wearable cognitive assistants, there are several valuable lessons we consider worth noting. In this section, we summarize and distill essential knowledge learned from our prototyping experience for practitioners.

Define Objects By Appearance

With object detection at the core of the prototyping methodology, carefully defining the objects to detect is crucial to the accuracy and the robustness of an assistant. Contrary to conventional definition of objects in computer vision, it is also often beneficial to define objects by strict appearance when building cognitive assistants. Many objects, when looking from various perspectives, appear significantly different. Defining objects by appearance means considering only similar views of an object to be the same class. Views that appear noticeably different from other views should be labeled as different classes when training the object detectors. Of course, the application logic can still maintain the knowledge that these different views, in fact, are associated with the same item. Considering different views as different classes makes many detection

```
// represents the trigger condition
message TransitionPredicate {
  string name = 1;
  string callable_name = 2;
  map<string, bytes> callable_kwargs = 3; // arguments
  string callable_args = 4; // arguments
}

message Instruction {
  string name = 1;
  string audio = 2; // audio in text format.
  bytes image = 3;
  bytes video = 4;
}

message Transition {
  string name = 1;
  // function name of the trigger condition
  repeated TransitionPredicate predicates = 2;
  Instruction instruction = 3;
  string next_state = 4;
}

// represent feature extraction modules
message Processor {
  // input are images
  // outputs are key/value pairs that represents application state
  string name = 1;
  string callable_name = 2;
  map<string, bytes> callable_kwargs = 3; // arguments
  string callable_args = 4; // arguments
}

message State {
  string name = 1;
  repeated Processor processors = 2; // extract features
  repeated Transition transitions = 3;
}

message StateMachine {
  string name = 1;
  repeated State states = 2; // all states
  map<string, bytes> assets = 3; // shared assets
  string start_state = 4;
}
```

Figure 6.11: OpenWorkflow FSM Binary Format

tasks inherently easier and often results in higher recognition accuracy. The fundamental reason that allows for this view-based detection for a higher accuracy is that we can leverage the human in the loop to create good conditions for machine intelligence. A commonly used technique in the implementation of several existing prototypes is to ask users to show a specific view of the objects. A few examples exist in the workflow of RibLoc application, as shown in Figure 2.6.

Consider Partial Objects

When building cognitive assistants for step-by-step assembly tasks, a pragmatic technique is to consider detecting small parts of objects and reason about their spatial relationship to verify assembly. Treating a small portion of an object to be a standalone item makes many computer vision checks tractable. For example, in Figure 2.8, although the slot is a small portion of the larger cap. It would be very difficult to verify that the pin has been put into the slot with only the detected bounding boxes of the larger cap. Treating the slot, in addition to the cap, as an object of itself and building an object detector for it makes the check possible. In fact, following the rule of defining objects by appearance, we trained two separate object detectors for both the slot with a pin and the slot without a properly placed pin.

Leverage the Human in the Loop

Compared to fully automated robotic systems, cognitive assistance systems have a unique advantage — the user in the loop. The availability of a collaborative human able and willing to follow instructions enables many out-of-band techniques to reduce the difficulty of the visual perception problem. For example, in RibLoc application (Figure 2.6), the imprinted words on the gauge has too low contrast to be recognized. Instead, since imprinted are simple words of colors, we rely on the user to read them out and perform speech recognition of a few keywords, which is much simpler and reliable. In general, developers should consider using carefully designed unambiguous instructions to ask users to create favorable conditions for the assistant in order to solve hard or intractable recognition and perception problems.

6.6 Chapter Summary and Discussion

Wearable cognitive assistants are difficult to develop due to the high barrier of entry and the lack of development methodology and tools. In this chapter, we present a unifying development methodology, centered around object detection and finite state machine representation to systematize the development process. Based on this methodology, we build a suite of development tools that helps object detector creation and speeds up task model implementation.

Chapter 7

Conclusion and Future Work

This dissertation addresses the problem of *scaling wearable cognitive assistance* for widespread deployment. We propose system optimizations that reduce network transmission, leverage application characteristics to adapt client behaviors, and provide an adaptation-centric resource management mechanism. In addition, we design and develop a suite of development tools that lower the barrier of entry and improve developer productivity. This chapter concludes the dissertation with a summary of contributions, and discusses future research directions and challenges in this area.

7.1 Contributions

As stated in Chapter 1, the thesis validated by this dissertation is as follows:

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.

To validate this thesis, we first introduce two example wearable cognitive assistants and present measurement studies on how they would saturate existing wireless network bandwidth. We propose two application agnostic techniques to reduce network transmission. Then, leveraging WCA application characteristics, we provide an adaptation taxonomy and demonstrate techniques to reduce offered load on the client device. With these adaptation mechanisms, we design and evaluate a adaptation-centric resource allocation mechanism at the cloudlet that takes advantages of application degradation profiles. We then offer a new application development methodology and provide a suite of tools to reduce development difficulty and speed up application development process.

7.2 Future Work

7.2.1 Advanced Computer Vision For Wearable Cognitive Assistance

Most WCAs developed and discussed in this dissertation are frame-based. Namely, they assume that complete user states can be captured within a single frame. While symbolic states can be aggregated, computer vision advancement in video analysis (e.g. activity recognition), can significantly broaden the horizon and improve the robustness of WCAs. For instance, in many assembly tasks, screws become occluded when placed correctly. Current solutions, as a workaround, often consider the screw in-place as a separate object. This is not robust to wrong user actions (e.g. clockwise tightening vs counter-clockwise tightening). Activity recognition that remembers a history of objects and human actions can help solve the problem and thus enable many new WCA domains.

7.2.2 Fine-grained Online Resource Management

This dissertation opens up many potential directions to explore in practical resource management for edge-native applications. We have alluded to some of these topics earlier.

One example we briefly mentioned is the dynamic partitioning of work between Tier-3 and Tier-2 to further reduce offered load on cloudlets. In addition, other resource allocation policies, especially fairness-centered policies, such as max-min fairness and static priority can be explored when optimizing overall system performance. These fairness-focused policies could also be used to address aggressive users, which are not considered in this dissertation. While we have shown offline profiling is effective for predicting demand and utility for WCA applications, for a broader range of edge-native applications, with ever more aggressive and variable offload management, online estimation may prove to be necessary.

Another area worth exploring is the particular set of control and coordination mechanisms to allow cloudlets to manage client-offered load directly. Finally, the implementation to date only controls allocation of resources but allows the cloudlet operating system to arbitrarily schedule application processes. Whether fine-grained control of application scheduling on cloudlets can help scale services remains an open question.

7.2.3 WCA Synthesis from Example Videos

The authoring tools presented in this dissertation can be considered as a first step towards an ambitious goal of synthesizing cognitive assistants automatically from crowd-sourced expert videos. The critical missing piece is the ability to analyze and summarize a consistent task model from multiple videos. Some work [81] has started to study this challenge, although there is still a long road ahead. Much needed is signficant improvement of domain-specific video understanding. Nonetheless, many techniques discussed in this dissertation could still apply and serve as stepping stones toward fully automated creation of WCAs.

Bibliography

[1] Computex TAIPEI. https://www.computextaipei.com.tw. Last accessed: October 2019.

[2] InWin. https://www.in-win.com. Last accessed: June 2019.

[3] RibLoc Fracture Plating System. https://acuteinnovations.com/product/ribloc/. Last accessed: June 2019.

[4] Raft Dataset. http://storage.cmusatyalab.org/drone2018/raft.zip, 2018. Last access: April 2020.

[5] Elephant Dataset. http://storage.cmusatyalab.org/drone2018/elephant.zip, 2018. Last access: April 2020.

[6] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*, pages 265–283, 2016.

[7] Victor Bahl. Emergence of micro datacenter (cloudlets/edges) for mobile computing. *Microsoft Devices & Networking Summit 2015*, 2015.

[8] Rajesh Krishna Balan, Mahadev Satyanarayanan, So Young Park, and Tadashi Okoshi. Tactics-based remote execution for mobile computing. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 273–286. ACM, 2003.

[9] Mohammadamin Barekatain, Miquel Martí, Hsueh-Fu Shih, Samuel Murray, Kotaro Nakayama, Yutaka Matsuo, and Helmut Prendinger. 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.

[10] David Barrett. One surveillance camera for every 11 people in Britain, says CCTV survey. *Daily Telegraph*, July 10, 2013. Last accessed: December 2019.

[11] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, Helsinki, Finland, 2012.

[12] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. Apollo: scalable and coordinated scheduling for cloud-scale com-

puting. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14)*, pages 285–300, 2014.

[13] Gabriel Brown. Converging Telecom & IT in the LTE RAN. White Paper, Heavy Reading, February 2013.

[14] Vannevar Bush and Vannevar Bush. As we may think. *Resonance*, 5(11), 1945.

[15] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 155–168. ACM, 2015.

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

[17] Zhuo Chen, Wenlu Hu, Junjue Wang, Siyan Zhao, Brandon Amos, Guanhang Wu, Kiryong Ha, Khalid Elgazzar, Padmanabhan Pillai, Roberta Klatzky, Dan 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, October 2017.

[18] 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.

[19] Kevin Christensen, Christoph Mertz, Padmanabhan Pillai, Martial Hebert, and Mahadev Satyanarayanan. Towards a distraction-free waze. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 15–20. ACM, 2019.

[20] Luis M. Contreras and Diego R. Lopez. A Network Service Provider Perspective on Network Slicing. *IEEE Softwarization*, January 2018.

[21] 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.

[22] Martin Danelljan, Gustav Häger, Fahad Khan, and Michael Felsberg. Accurate scale estimation for robust visual tracking. In *British Machine Vision Conference, Nottingham, September 1-5, 2014*. BMVA Press, 2014.

[23] Nigel Davies, Keith Mitchell, Keith Cheverst, and Gordon Blair. Developing a context sensitive tourist guide. In *1st Workshop on Human Computer Interaction with Mobile Devices, GIST Technical Report G98-1*, volume 1, 1998.

[24] Christina Delimitrou and Christos Kozyrakis. Quasar: resource-efficient and qos-aware cluster management. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 127–144. ACM, 2014.

[25] DJI, Inc. Lightbridge 2. `https://www.dji.com/lightbridge-2`. Last accessed: November 2017.

[26] Zak Doffman. Battlefield 2.0: How Edge Artificial Intelligence is Pitting Man Against Machine. *Forbes*, November 2018.

[27] Debidatta Dwibedi, Ishan Misra, and Martial Hebert. Cut, paste and learn: Surprisingly easy synthesis for instance detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1301–1310, 2017.

[28] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[29] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

[30] Jason Flinn and Mahadev Satyanarayanan. Energy-aware Adaptation for Mobile Applications. In *Proceedings of the Seventeenth ACM Symposium on Operating systems Principles*, Charleston, SC, 1999.

[31] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The Elements of Statistical Learning*, volume 1. Springer series in statistics, 2001.

[32] Tom ZJ Fu, Jianbing Ding, Richard TB Ma, Marianne Winslett, Yin Yang, and Zhenjie Zhang. Drs: dynamic resource scheduling for real-time analytics over fast streams. In *2015 IEEE 35th International Conference on Distributed Computing Systems*, pages 411–420. IEEE, 2015.

[33] Google. Google photos. `https://www.google.com/photos/about/`, 2019. Last accessed: December 2019.

[34] WSGI Gunicorn Python. Http server for unix. *URL: http://gunicorn. org*, 2017.

[35] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. Towards Wearable Cognitive Assistance. In *Proceedings of ACM MobiSys*, 2014.

[36] 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.

[37] Devindra Hardawar. Intel's Joule is its most powerful dev kit yet. https://www.engadget.com/2016/08/16/intels-joule-is-its-most-powerful-dev-kit-yet/, August 2016. Last accessed: Sept 2018.

[38] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, Oct 2017. doi: 10.1109/ICCV.2017.322.

[39] 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.

[40] Adrian Holovaty and Jacob Kaplan-Moss. *The definitive guide to Django: Web development done right*. Apress, 2009.

[41] 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.

[42] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu. Focus: Querying large video datasets with low latency and low cost. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 269–286, 2018.

[43] 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 HotMobile 2015*, Santa Fe, NM, February 2015.

[44] 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.

[45] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[46] Hulu. Internet speed requirements for streaming HD and 4K Ultra HD. `https://help.hulu.com/en-us/requirements-for-hd`, 2017. Last accessed: May 2017.

[47] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. Videoedge: Processing camera streams using hierarchical clusters. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 115–131. IEEE, 2018.

[48] Pinterest Inc. Pinterest. `https://www.pinterest.com/`, 2019. Last accessed: December 2019.

[49] Puneet Jain, Justin Manweiler, and Romit Roy Choudhury. Overlay: Practical mobile augmented reality. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 331–344. ACM, 2015.

[50] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 253–266. ACM, 2018.

[51] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: optimizing neural network queries over video at scale. *Proceedings of the VLDB Endowment*, 10(11):1586–1597, 2017.

[52] H Kao and Hector Garcia-Molina. Deadline assignment in a distributed soft real-time system. In *[1993] Proceedings. The 13th International Conference on Distributed Computing*

Systems, pages 428–437. IEEE, 1993.

[53] Ahmed S Kaseb, Anup Mohan, and Yung-Hsiang Lu. Cloud resource management for image and video analysis of big data from network cameras. In *2015 International Conference on Cloud Computing and Big Data (CCBD)*, pages 287–294. IEEE, 2015.

[54] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90, 2016.

[55] Glenn E Krasner, Stephen T Pope, et al. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3):26–49, 1988.

[56] Nicholas D Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T Campbell. A survey of mobile phone sensing. *IEEE Communications magazine*, 48(9):140–150, 2010.

[57] Erik Learned-Miller, Gary B Huang, Aruni RoyChowdhury, Haoxiang Li, and Gang Hua. Labeled faces in the wild: A survey. In *Advances in face detection and facial image analysis*, pages 189–248. Springer, 2016.

[58] Kiron Lebeck, Eduardo Cuervo, and Matthai Philipose. Collaborative acceleration for mixed reality. Technical report, Microsoft, 2018.

[59] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 1–14. ACM, 2010.

[60] Peng Li. *Utility accrual real-time scheduling: Models and algorithms*. PhD thesis, Virginia Tech, 2004.

[61] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[62] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[63] Jack M Loomis, Reginald G Golledge, Roberta L Klatzky, Jon M Speigle, and Jerome Tietz. Personal guidance system for the visually impaired. In *Proceedings of the First Annual ACM Conference on Assistive Technologies*, pages 85–91. ACM, 1994.

[64] 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.

[65] Konrad Lorincz, Bor-rong Chen, Jason Waterman, Geoff Werner-Allen, and Matt Welsh. Resource aware programming in the pixie os. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 211–224. ACM, 2008.

[66] Konrad Lorincz, Bor-rong Chen, Geoffrey Werner Challen, Atanu Roy Chowdhury, Shyamal Patel, Paolo Bonato, Matt Welsh, et al. Mercury: a wearable sensor network platform

for high-fidelity motion analysis. In *SenSys*, volume 9, pages 183–196, 2009.

[67] LTEWorld. LTE Advanced: Evolution of LTE. http://lteworld.org/blog/lte-advanced-evolution-lte, August 2009. Last accessed: Jan 2016.

[68] Ilya Lysenkov, Victor Eruhimov, and Gary Bradski. Recognition and pose estimation of rigid transparent objects with a kinect sensor. *Robotics*, 273:273–280, 2013.

[69] Loren Merritt and Rahul Vanam. Improved rate control and motion estimation for h. 264 encoder. In *IEEE International Conference on Image Processing*, 2007.

[70] Jack Morse. Alphabet officially flips on Project Loon in Puerto Rico. http://mashable.com/2017/10/20/puerto-rico-project-loon-internet, October 2017. Last accessed: Sept 2018.

[71] Saman Naderiparizi, Pengyu Zhang, Matthai Philipose, Bodhi Priyantha, Jie Liu, and Deepak Ganesan. Glimpse: A Programmable Early-Discard Camera Architecture for Continuous Mobile Vision. In *Proceedings of MobiSys 2017*, June 2017.

[72] Saman Naderiparizi, Pengyu Zhang, Matthai Philipose, Bodhi Priyantha, Jie Liu, and Deepak Ganesan. Glimpse: A programmable early-discard camera architecture for continuous mobile vision. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 292–305. ACM, 2017.

[73] Clément Nedelcu. *Nginx HTTP Server*. Packt Publishing, 2010.

[74] Ryan Newton, Sivan Toledo, Lewis Girod, Hari Balakrishnan, and Samuel Madden. Wishbone: Profile-based partitioning for sensornet applications. In *NSDI*, volume 9, pages 395–408, 2009.

[75] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, J. Eric Tilton, Jason Flinn, and Kevin R. Walker. Agile Application-Aware Adaptation for Mobility. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, Saint-Malo, France, October 1997.

[76] NVIDIA. The Most Advanced Platform for AI at the Edge. http://www.nvidia.com/object/embedded-systems.html, 2017. Last accessed: Sept 2018.

[77] John Oakley. Intelligent Cognitive Assistants (ICA) Workshop Summary and Research Needs. https://www.nsf.gov/crssprgm/nano/reports/ICA2_Workshop_Report_2018.pdf, February 2018.

[78] OpenCV. Computer vision annotation tool. https://github.com/opencv/cvat, 2019. Last accessed: December 2019.

[79] Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. Sparrow: distributed, low latency scheduling. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 69–84. ACM, 2013.

[80] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[81] Truong-An Pham and Yu Xiao. Unsupervised workflow extraction from first-person video of mechanical assembly. In *Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications*, pages 31–36. ACM, 2018.

[82] Moo-Ryong Ra, Anmol Sheth, Lily Mummert, Padmanabhan Pillai, David Wetherall, and Ramesh Govindan. Odessa: enabling interactive perception applications on mobile devices. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, pages 43–56. ACM, 2011.

[83] Binoy Ravindran, E Douglas Jensen, and Peng Li. On recent advances in time/utility function real-time scheduling and resource management. In *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 55–60. IEEE, 2005.

[84] Tiernan Ray. An Angel on Your Shoulder: Who Will Build A.I.? *Barron's*, February 2018.

[85] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[86] Leonard Richardson and Sam Ruby. *RESTful web services*. O'Reilly Media, Inc, 2008.

[87] 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.

[88] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[89] Vishwam Sankaran. Google X's ambitious Loon and Wing projects graduate into independent companies. https://thenextweb.com/google/2018/07/12/google-xs-ambitious-loon-and-wing-projects-graduate-into-independent-companies, July 2018. Last accessed: September, 2018.

[90] Mahadev Satyanarayanan. Fundamental Challenges in Mobile Computing. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, Ottawa, Canada, 1996.

[91] Mahadev Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, 8(4), 2001.

[92] Mahadev Satyanarayanan. Augmenting Cognition. *IEEE Pervasive Computing*, 3(2), April-June 2004.

[93] Mahadev Satyanarayanan and Nigel Davies. Augmenting Cognition through Edge Computing. *IEEE Computer*, 52(7), July 2019.

[94] Mahadev Satyanarayanan and Dushyanth Narayanan. Multi-Fidelity Algorithms for Interactive Mobile Applications. In *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DialM)*, Seattle,

WA, August 1999.

[95] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4), October-December 2009.

[96] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, pages 14–23, 2009.

[97] Mahadev Satyanarayanan, Wei Gao, and Brandon Lucia. The computing landscape of the 21st century. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 45–50. ACM, 2019.

[98] Mahadev Satyanarayanan, Guenter Klas, Marco Silva, and Simone Mangiante. The Seminal Role of Edge-Native Applications. In *Proceedings of the 2019 IEEE International Conference on Edge Computing (EDGE)*, Milan, Italy, July 2019.

[99] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[100] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. Omega: flexible, scalable schedulers for large compute clusters. 2013.

[101] Krisantus Sembiring and Andreas Beyer. Dynamic resource allocation for cloud-based media processing. In *Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 49–54. ACM, 2013.

[102] Asim Smailagic and Daniel Siewiorek. Application design for wearable and context-aware computers. *IEEE Pervasive Computing*, 1(4):20–29, 2002.

[103] Asim Smailagic and Daniel P Siewiorek. A case study in embedded-system design: The vuman 2 wearable computer. *IEEE Design & Test of Computers*, 10(3):56–67, 1993.

[104] Asim Smailagic, Daniel P Siewiorek, Richard Martin, and John Stivoric. Very rapid prototyping of wearable computers: A case study of vuman 3 custom versus off-the-shelf design methodologies. *Design Automation for Embedded Systems*, 3(2-3):219–232, 1998.

[105] CACM Staff. React: Facebook's functional turn on writing javascript. *Communications of the ACM*, 59(12):56–62, 2016.

[106] John A Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C Buttazzo. *Deadline scheduling for real-time systems: EDF and related algorithms*, volume 460. Springer Science & Business Media, 2012.

[107] Christoph Steiger, Herbert Walder, and Marco Platzner. Operating systems for reconfigurable embedded platforms: Online scheduling of real-time tasks. *IEEE Transactions on computers*, 53(11):1393–1407, 2004.

[108] Sherry Stokes. New Center Headquartered at Carnegie Mellon Will Build Smarter Networks To Connect Edge Devices to the Cloud. `https://www.cmu.edu/news/stories/archives/2018/january/conix-research-center.html`, January 2018.

[109] Tensorflow Team. Tensorflow object detection api. https://github.com/tensorflow/models/tree/master/research/object_detection, 2019. Last accessed: December 2019.

[110] Tensorflow. Tensorflow Pre-trained Models Top-1 Accuracy. https://github.com/junjuew/models/tree/master/research/slim. Last accessed: December 2018.

[111] Hideyuki Tokuda, Tatsuo Nakajima, and Prithvi Rao. Real-time mach: Towards a predictable real-time system. In *USENIX Mach Symposium*, pages 73–82. Citeseer, 1990.

[112] Narseo Vallina-Rodriguez and Jon Crowcroft. Energy management techniques in modern mobile handsets. *IEEE Communications Surveys & Tutorials*, 15(1):179–198, 2012.

[113] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*, page 18. ACM, 2015.

[114] Paul Viola and Michael Jones. Robust Real-time Object Detection. In *International Journal of Computer Vision*, 2001.

[115] Junjue Wang, Brandon Amos, Anupam Das, Padmanabhan Pillai, Norman Sadeh, and Mahadev Satyanarayanan. A Scalable and Privacy-Aware IoT Service for Live Video Analytics. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, pages 38–49. ACM, 2017.

[116] Xiaoli Wang, Aakanksha Chowdhery, and Mung Chiang. SkyEyes: Adaptive Video Streaming from UAVs. In *Proceedings of the 3rd Workshop on Hot Topics in Wireless*. ACM, 2016.

[117] Xiaoli Wang, Aakanksha Chowdhery, and Mung Chiang. Networked Drone Cameras for Sports Streaming. In *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 308–318, 2017.

[118] Yi Yao, Jiayin Wang, Bo Sheng, Jason Lin, and Ningfang Mi. Haste: Hadoop yarn scheduling based on task-dependency and resource-demand. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 184–191. IEEE, 2014.

[119] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. Lavea: Latency-aware video analytics on edge computing platform. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 15. ACM, 2017.

[120] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[121] Suya You, Ulrich Neumann, and Ronald Azuma. Hybrid inertial and vision tracking for augmented reality registration. In *Proceedings IEEE Virtual Reality (Cat. No. 99CB36316)*, pages 260–267. IEEE, 1999.

[122] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. Live video analytics at scale with approximation and delay-tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementa-*

tion, pages 377–392, 2017.

[123] Tan Zhang, Aakanksha Chowdhery, Paramvir Victor Bahl, Kyle Jamieson, and Suman Banerjee. The design and implementation of a wireless video surveillance system. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 426–438. ACM, 2015.

[124] Xingzhou Zhang, Yifan Wang, and Weisong Shi. pcamp: Performance comparison of machine learning packages on the edges. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge'18)*, 2018.

[125] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *arXiv preprint arXiv:1905.05055*, 2019.