# Scaling Wearable Cognitive Assistance

Junjue Wang
junjuew@cs.cmu.edu

June 2018

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

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

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

**Keywords:** Wearable Cognitive Assistance, Edge Computing, Cloudlet, Scalability

# Contents

# 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* [**?** ]. 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. [**?** ] [**?** ] [**?** ] 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. [**?** ] [**?** ] explored using computers carried in a backpack to provide auditory cues in order to help the blind navigate. Davis et al. [**?** ] [**?** ] developed a context-sensitive intelligent visitor guide leveraging hand-portable multimedia systems. While these research work pioneered cognitive assistance and its related fields, their robustness and functionality were limited by the hardware technologies of their time.

More recently, as underlying technologies experience significant advancement, a new genre of applications, Wearable Cognitive Assistance (WCA) [**?** ] [**?** ], 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 [**?** ] 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 [**?** ], 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* [**?** ]. In 2017, Chen et al [**?** ] 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 [**? ?** ]. At the Mobile World Congress in February 2018,

wearable cognitive assistance was the focus of an entire session [**?** ]. For AI-based military use cases, this class of applications is the centerpiece of "Battlefield 2.0" [**?** ]. By mid-2019, WCA was being viewed as a prime source of "killer apps" for edge computing [**? ?** ].

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 [**?** ] 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 [**?** ] [**?** ] [**?** ]. On top of Gabriel, several prototype applications have been built, such as Ping-Pong Assistance, Lego Assistance, Sandwich Assistance, and Ikea Lamp Assembly Assistance. Using these applications as benchmarks, Chen et al. [**?** ] 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 preclude operation at such low utilization. In contrast, resources are often highly utilized and congested when serving many users. How to efficiently scale Gabriel applications to a large number of users remains to be answered. Second, previous work on the Gabriel framework reduces application development efforts by managing client-server communication, network flow control, and cognitive engine discovery. However, the framework does not address the most time-consuming parts of creating a wearable cognitive assistance application. Experience has shown that developing computer vision modules that analyze video feeds is a time-consuming and painstaking process that requires special expertise and involves rounds of trial and error. Developer tools that alleviate the time and the expertise needed can greatly facilitate the creation of these applications.

## 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 should enable the most 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 offered load from WCA mobile client when oversubscription occurs.
2. We provide a profiling-based cloudlet resource allocation mechanism that takes into account the unique adaptation behaviors of different applications.
3. We create a suite of development and deployment tools to reduce the time and lower the barrier of entry for WCA development.

## 1.2 Thesis Overview

The remainder of this dissertation is organized as follows.

- In Chapter **??**, we describe application-agnostic techniques to reduce bandwidth consumption.
- In Chapter **??**, we propose and evaluate application-specific techniques to reduce offered load. We demonstrate their effectiveness with minimal impact on result latency.
- In Chapter **??**, we present a resource management mechanisms that takes into the account application adaptation characteristics to optimize system-wide aggregation metrics.
- In Chapter **??**, we introduce a suite of development tools for quick prototyping.
- In Chapter **??**, we showcase a deployment system to facilitate deploying WCAs on cloudlets.

# Chapter 2

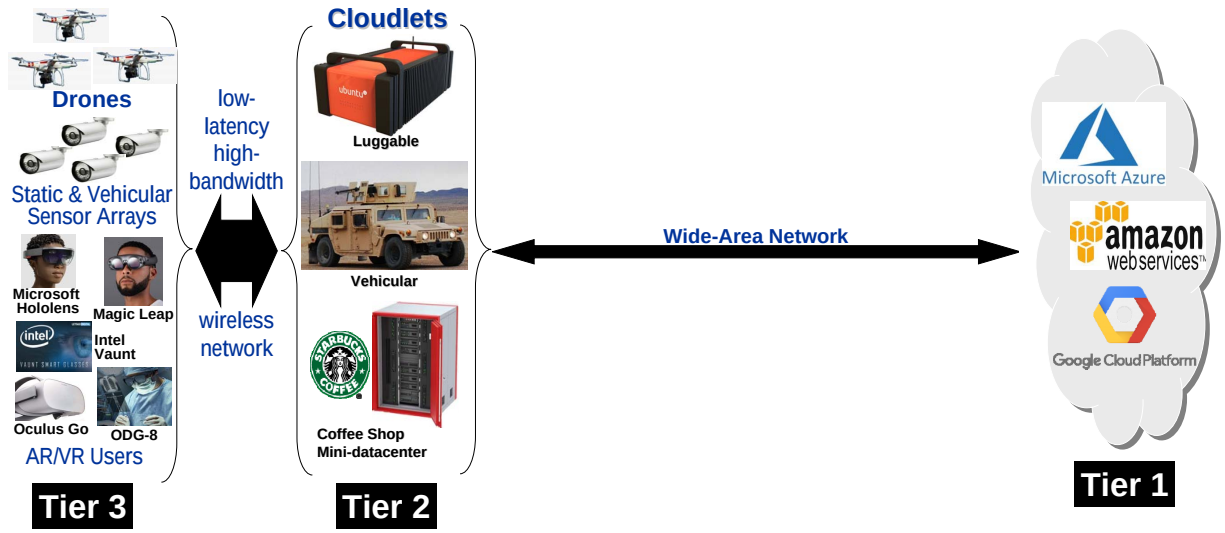
# Background

## 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" [**?** ], "micro data centers (MDCs)" [**?** ], "fog" [**?** ], and "mobile edge computing (MEC)" [**?** ] 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. [**?** ] best describes the modern computing landscape with edge computing using a tiered model, shown in Figure **??**. 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 state-of-art computer vision algorithms to analyze the videos they capture. To overcome this tension, a



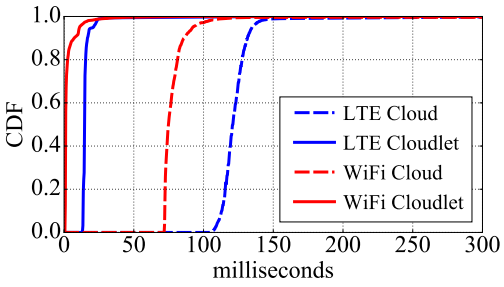

(Source: Satya et al [**?** ])

Figure 2.1: Tiered Model of Computing

tier-3 device could offload computation over network to tier-1. This capability was first demonstrated in 1997 by Noble et al. [**?** ], who used it to implement speech recognition with acceptable performance on a resource-limited mobile device. In 1999, Flinn et al. [**?** ] 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 [**?** ]. 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.

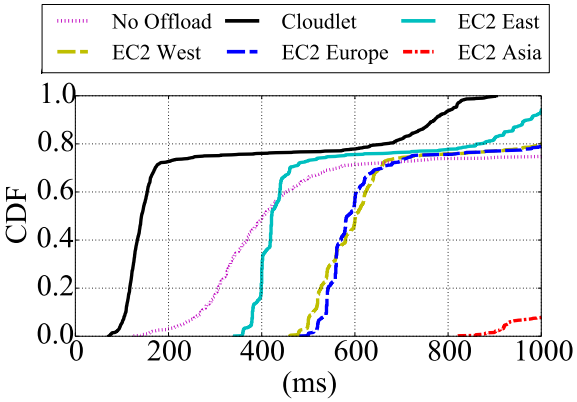
However, offloading computation to the cloud has its own downside. Because of the consolidation needed to achieve the economy of scale, today's datacenters are "far" from tier-3 devices. The latency, throughput, and cost of wide-area network (WAN) significantly limit the amount of applications that can 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. [**?** ] 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 datacenters means its aggregation network need to carry significant amount of traffic. As the number of tier-3 devices is expected to grow exponentially, these network links 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 **??**, is proposed.




(Source: Hu et al [**?** ])

Figure 2.2: CDF of pinging RTTs

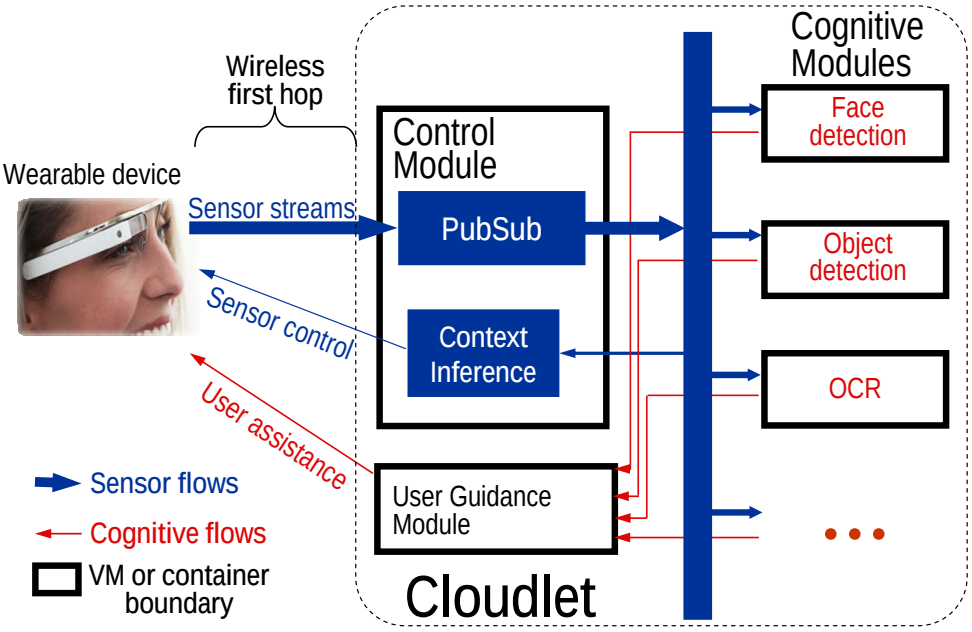



(Source: Hu et al [**?** ])

Figure 2.3: FACE Response Time over LTE

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 datacenters. Figure **??** 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 **??** 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. Cloudlet also poses unique challenges for scalability as resources are a lot more limited compared to datacenters. How to scale WCAs to many users using cloudlets is the key question this thesis set out to investigate.




(Source: Chen et al [**?** ])

Figure 2.4: Gabriel Platform

## 2.2 Gabriel Platform

The Gabriel platform [**?** **?** ], shown in Figure **??**, 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 modules 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 thesis. In addition, we

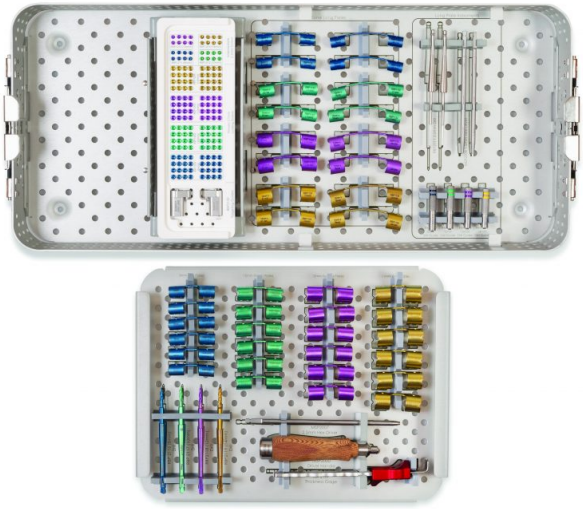


Figure 2.5: RibLoc Kit

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. Figure **??** provides a summary of applications built by Chen et al [**?** ]. These applications run on multiple wearable devices such as Google Glass, Microsoft HoloLens, Vuzix Glass, and ODG R7. At a high level, the cloudlet workflows of these applications are similar, and consist 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 state.

Leveraging lessons learned by Chen et al [**?** ], we created and implemented 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 focused 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.

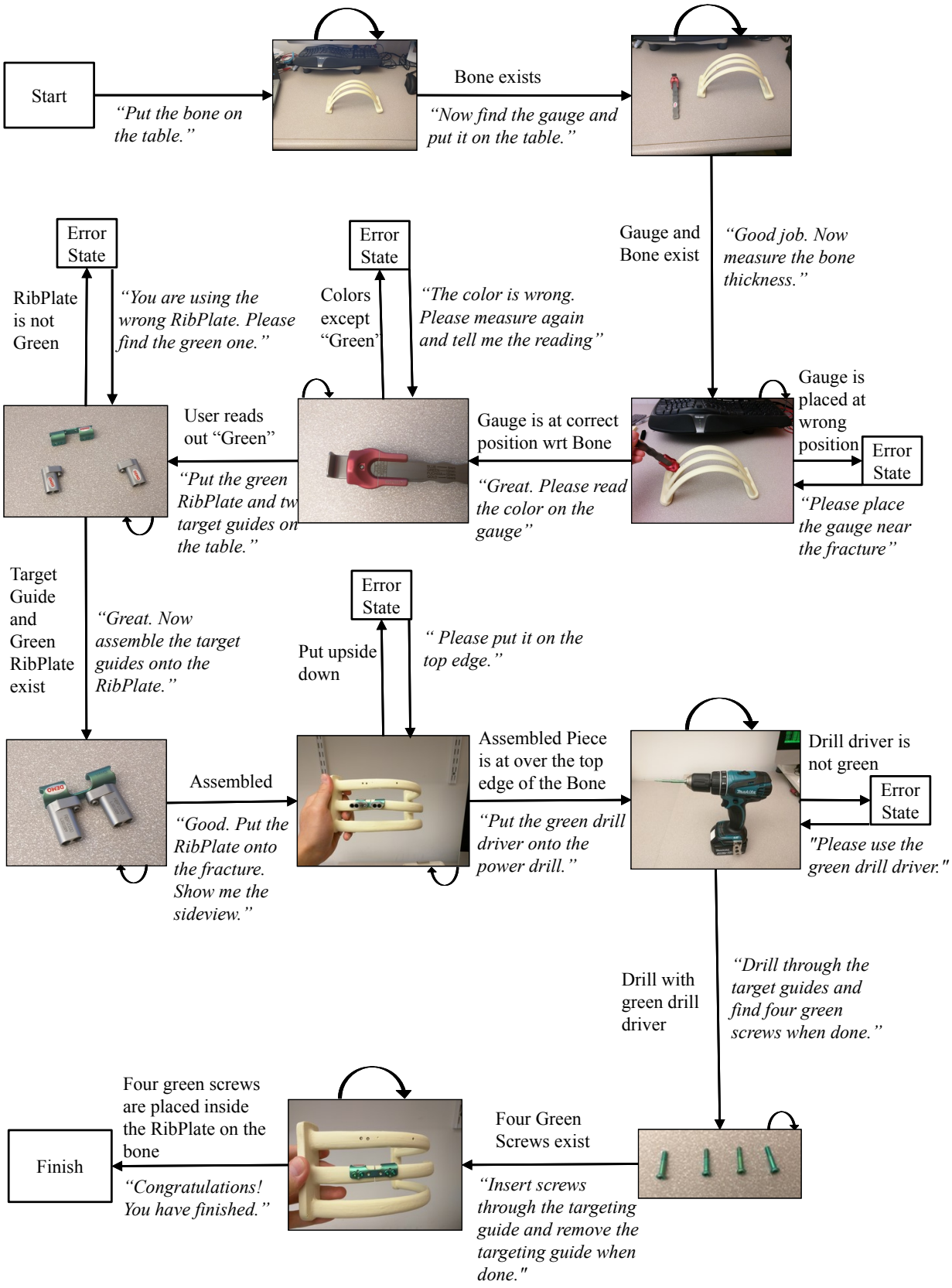



Figure 2.6: RibLoc Assistant Task State Machine

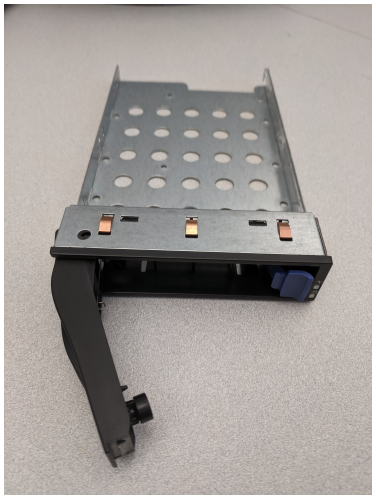


Figure 2.7: Assembled DiskTray

### 2.3.1 RibLoc Application

RibLoc Fracture Plating System [**?** ], shown in Figure **??** is used by surgeons to fixate broken ribs for fracture repair. They 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 **??** 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 shown 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 [**?** ], a computer hardware manufacturer, we created a cognitive assistance to train operators how to assemble their disk tray product. Figure **??** shows the assembled tray, which is used to host hard drives that go into a server chassis. As 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 **??**, 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

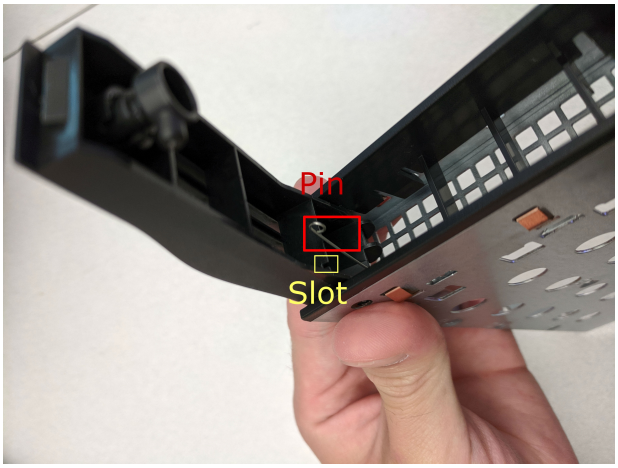



Figure 2.8: Small Parts in DiskTray WCA

| | 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 |

Figure 2.9: Application Latency Bounds (in milliseconds)

(Adapted from Chen et al [**?** ])

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 [**?** ]. 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 [**?** ]. 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 PINGPONG 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 made a hit.

Previous work [**?** ] 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.

Figure **??** shows a summary of latency bounds calculated from the previous work [**?** ]. 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 thesis, 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.

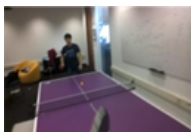
| App Name | Example Input Video Frame | Description | Symbolic Representation | Example Guidance |
|---|---|---|---|---|
| **Pool** |  | Helps a novice pool player aim correctly. Gives continuous visual feedback (left arrow, right arrow, or thumbs up) as the user turns his cue stick. Correct shot angle 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 [**?** ].<br>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.<br>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.<br>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.<br>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" |

Figure 2.10: Prototype Wearable Cognitive Assistance Applications

# Chapter 3

# Application-Agnostic Techniques to Reduce Network Transmission

WCAs continuously stream sensor data to the cloudlet. The richer a sensing modality is, the more information can be extracted. The core sensing modality of WCAs is visual data, e.g. first-person view images and videos from wearable cameras. Compared to other sensors, e.g. microphones and inertial measurement units (IMUs), cameras capture deep semantic information. With the fast decreasing costs of camera hardware, they become increasingly pervasive in recent years. In the meantime, significant advancement in computer driven by deep neural networks (DNNs) has made many previously difficult perception problems tractable. The richness and open-endness of visual data makes camera the core sensor of 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 [**?** ]. 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 [**?** ]. This is clearly not scalable.

In this chapter, we show how per-user bandwidth demand in 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. We present techniques for an adaptive computer vision pipeline for Tier-3 devices that leverages edge computing to enable dynamic optimizations. In contrast to previous works [**?** ] [**?** ] [**?** ], we leverage state-of-the-art deep neural networks (DNNs) to selectively transmit interesting data from a video stream and explore mission-specific optimizations. We present results first in the context of live video analytics for drones. As exemplars of Tier-3 devices, drones and wearable devices face similar challenges in live video analysis. Then, we showcase how these techniques can be applied to WCAs.

## 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 [**?** ]. 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 can fit on a typical Tier-3 device. State-of-art techniques in image processing use DNNs that are compute- and memory-intensive. Figure **??** 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. Both tasks used publicly available pretrained DNN models. 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 [**?** ] and the NVIDIA Jetson [**?** ] that are physically compact and light enough to be credible as wearable device platforms in a few years.

Fig. **??**, 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 [**?** ]), but rather to illustrate the differences between wearable platforms and fixed infrastructure servers.

Image classification 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. The prediction speed using two different DNNs are shown. MobileNet V1 [**?** ] is a DNN designed for mobile devices from the ground-up by reducing the number of parameters and simplifying the computation using depth-wise separable convolution. ResNet101 V1 [**?** ] is a more accurate but also more resource-hungry DNN that won the ImageNet classification challenge in 2015 [**?** ].

Object detection is a harder task than image classification, because it requires bounding boxes to be predicted around the specific areas of an image that contains a particular class of 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) [**?** ] and Faster R-CNN [**?** ]. 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 [**?** ] uses a separate region proposal neural network to predict regions of interest and has been shown to achieve higher accuracy [**?** ]. Figure **??** presents results in four columns:

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.

Figure 3.1: Deep Neural Network Inference Speed

SSD combined with MobileNet V1 or Inception V2, and Faster R-CNN combined with Inception V2 or ResNet101 V1 [**?** ]. The combination of Faster R-CNN and ResNet101 V1 is one of the most accurate object detectors available today [**?** ]. 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 mobile platforms 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.

### 3.1.2 Result Latency, Offloading & Scalability

*Result latency* is the delay between first capture of a video frame in which a particular result (e.g., image of a survivor) 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 step-by-step instruction feedback from wearable devices. In this chapter, we focus on approaches that aim for much smaller result latency: ideally, close to real-time.

A different approach is to offload video processing during flight over a wireless link to an edge computing node (cloudlet). With this approach, even a weak Tier-3 device can leverage the substantial processing capability of a ground-located 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 many years away, especially at global scale. More specialized wireless technologies, such as Lightbridge 2 [**?** ] for drones, can also be used. Regardless of specific wireless technology, the principles and techniques described in this chapter apply.

*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 in-

| Task | Detection Goal | Data Source | Data Attributes | Training Subset | Testing Subset |
|---|---|---|---|---|---|
| T1 | People in scenes of daily life | Okutama Action Dataset [**?** ] | 33 videos<br>59842 fr<br>4K@30 fps | 9 videos<br>17763 fr | 6 videos<br>20751 fr |
| T2 | Moving cars | Stanford Drone Dataset [**?** ] | 60 videos<br>522497 fr<br>1080p@30 fps | 16 videos<br>179992 fr | 14 videos<br>92378 fr<br>Combination of test videos from each dataset. |
| T3 | Raft in flooding scene | YouTube collection [**?** ] | 11 videos<br>54395 fr<br>720p@25 fps | 8 videos<br>43017 fr | |
| T4 | Elephants in natural habitat | YouTube collection [**?** ] | 11 videos<br>54203 fr<br>720p@25 fps | 8 videos<br>39466 fr | |
| T5 | Pushing or pulling Suitcases | Okutama Action Dataset | Same as T1 | Same as T1 | |

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

Figure 3.2: Benchmark Suite of Video Traces

frastructure 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 [**?** ], can only sustain much lower uplink bandwidth per cell, e.g. 10Mbps for the balloon-based LTE [**?** ]. 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: DUMB 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 for drone search to WCAs.

| 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.

Figure 3.3: Baseline Object Detection KPIs

### 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. Figure **??** presents this benchmark suite of videos, organized into five tasks. All the tasks involve detection of tiny objects on individual frames. Task T5 additionally involves action detection, which operates on short video segments rather than 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 Results

Figure **??** 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 Figure **??**, 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 Figure **??** were based on 1080p frames, while tasks T1 and T5 use the higher resolution of 4K. This will further increase demand on cloudlet resources.

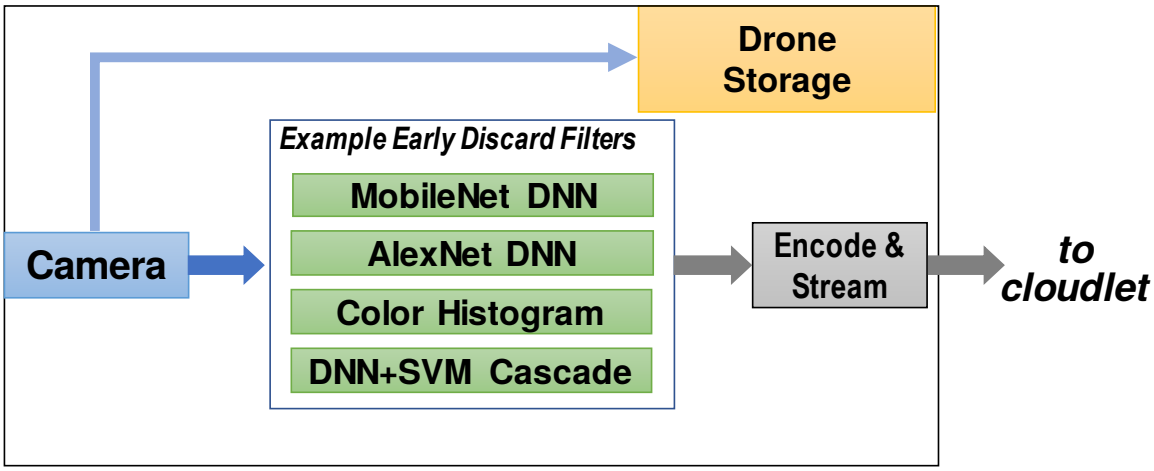



Figure 3.4: Drone-based Early Discard

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 drone 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. Previous work [**?** ] [**?** ] leveraged pixel-level features and multiple sensing modalities to select interesting frames from hand-held or body-worn cameras. In this work, we explore the use of DNNs to filter frames from aerial views. The benefits of using DNNs are twofold. First, DNNs are trained and specialized for each task, resulting in their high accuracy and robustness. Second, no additional hardware is added to existing drone 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* 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 **??**, we envision a choice of weak detectors being available as early

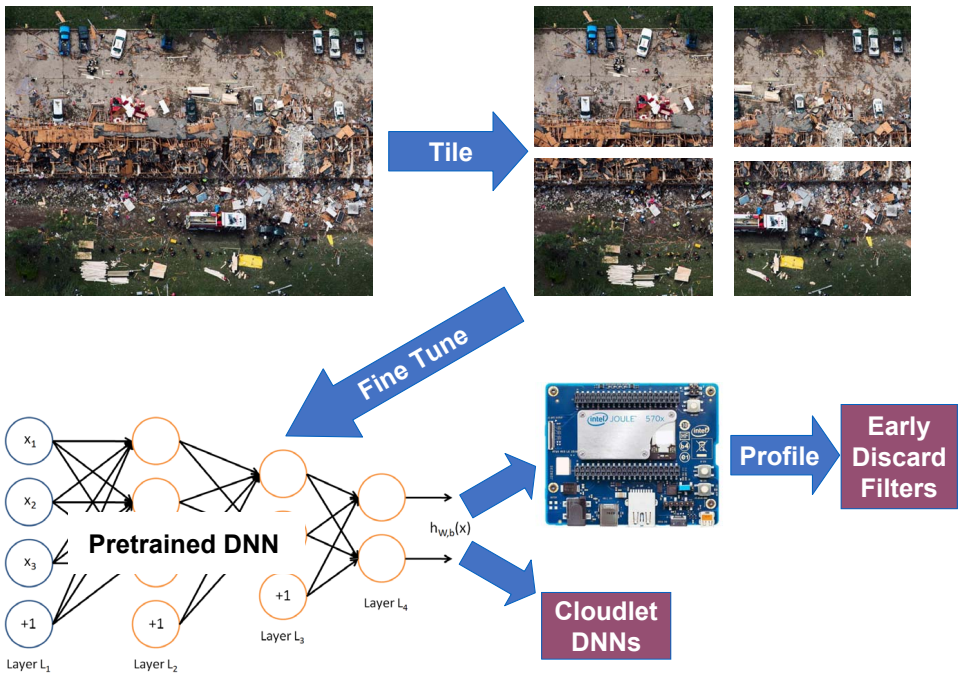



Figure 3.5: Tiling and DNN Fine Tuning

discard filters on a drone, with the specific choice of filter being mission-specific. Relative to the measurements presented in Figure **??**, early discard only requires image classification: it is not necessary to know exactly where in the frame a relevant object occurs. This suggests that MobileNet would be a good choice as a weak detector. Its speed of 13 ms per frame on Jetson yields more than 75 fps. We therefore use MobileNet on the drone for early discard in our experiments.

Pre-trained classifiers for MobileNet are available today for 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. A drone, however, has an aerial viewpoint and objects look rather different. To improve classification accuracy on drones, we used *transfer learning* [**?** ] to finetune the pre-trained classifiers on small training sets of images that were captured from an aerial viewpoint. This 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 aerial images without incurring the full cost of creating a large training set captured from an aerial viewpoint.

Drone 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 [**?** ]. To address this problem, we *tile* high resolution frames into multiple sub-frames and then perform recognition on the sub-

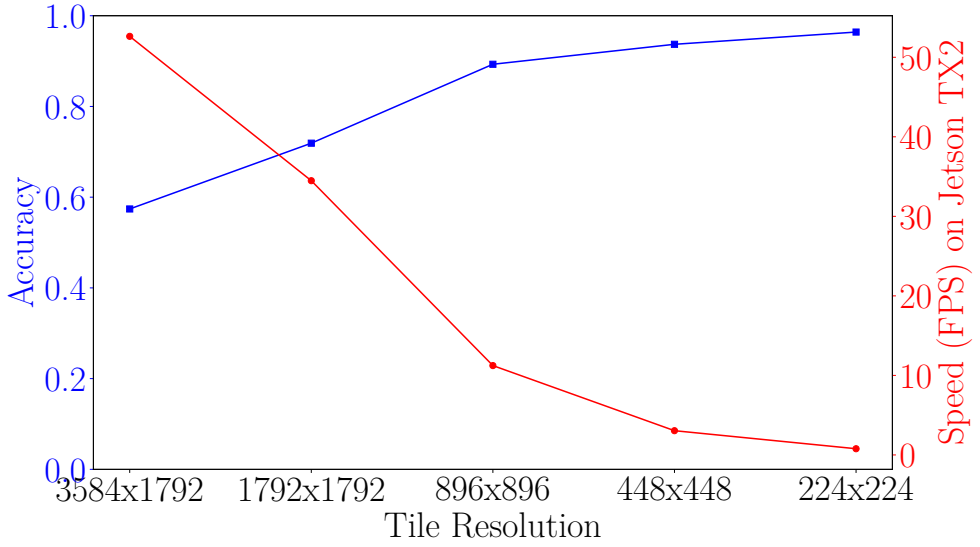



Figure 3.6: Speed-Accuracy Trade-off of Tiling

frames. This is done offline for training, as shown in Figure **??**, 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.

### 3.3.2 Experimental Setup

Our experiments on the EARLYDISCARD strategy used the same benchmark suite described in Section **??**. We used Jetson TX2 as the drone platform. We use both frame-based and event-based metrics to evaluate the MobileNet filters.

### 3.3.3 Results of Early Discard Filters

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 Figure **??**, to explore the effects of tiling. For this dataset, Figure **??** 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

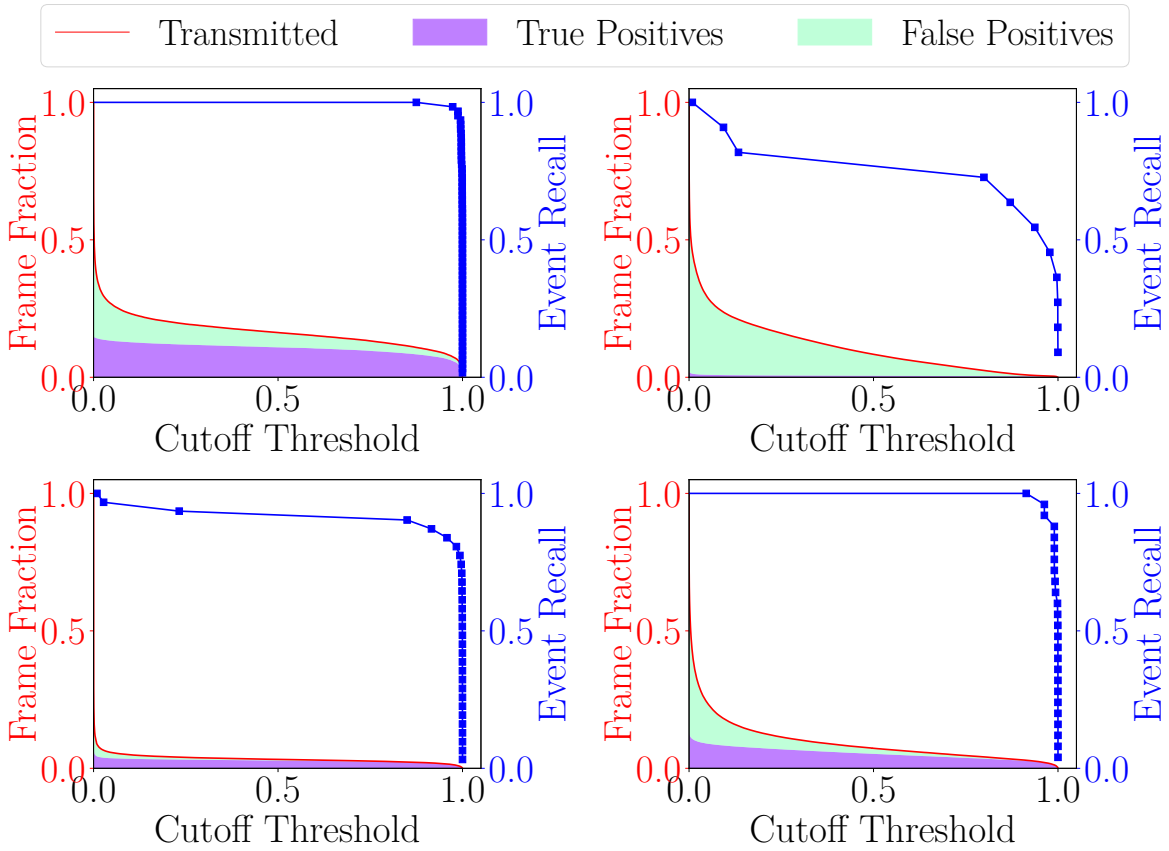



Figure 3.7: Bandwidth Breakdown

balance between the speed and accuracy. In the rest of the paper, we use two tiles per frame by default.

**Drone Filter Accuracy**: The output of a drone 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 drone storage for post-mission processing.

Figure **??** shows our results on all four tasks. Events such as detection of a raft in T3 occur in consecutive frames, all of which contain the object of interest. A correct detection of an event is defined as at least one of the consecutive frames being transmitted to the cloudlet. Blue lines in Figure **??** shows how the event recalls of drone 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 drone drops a frame containing an important event, improved cloudlet processing can-

| | 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 |

Figure 3.8: Recall, Event Latency and Bandwidth at Cutoff Threshold 0.5

not help. The results in the third column of Figure **??** confirm that there are no false negatives for T1 and T4 at a cutoff threshold of 0.5. For T2 and T3, lower cutoff thresholds are needed to achieve perfect recalls.

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

**Bandwidth**: Columns 5–7 of Figure **??** 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 drone-cloudlet video transmission. Average bandwidth is calculated as the total data transmitted divided by mission duration. Comparing column 5 of Figure **??** with column 2 of Figure **??**, 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 drone can drop many frames.

Figure **??** 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 **??** shows the effects of sending a sample of frames from the drone, without any

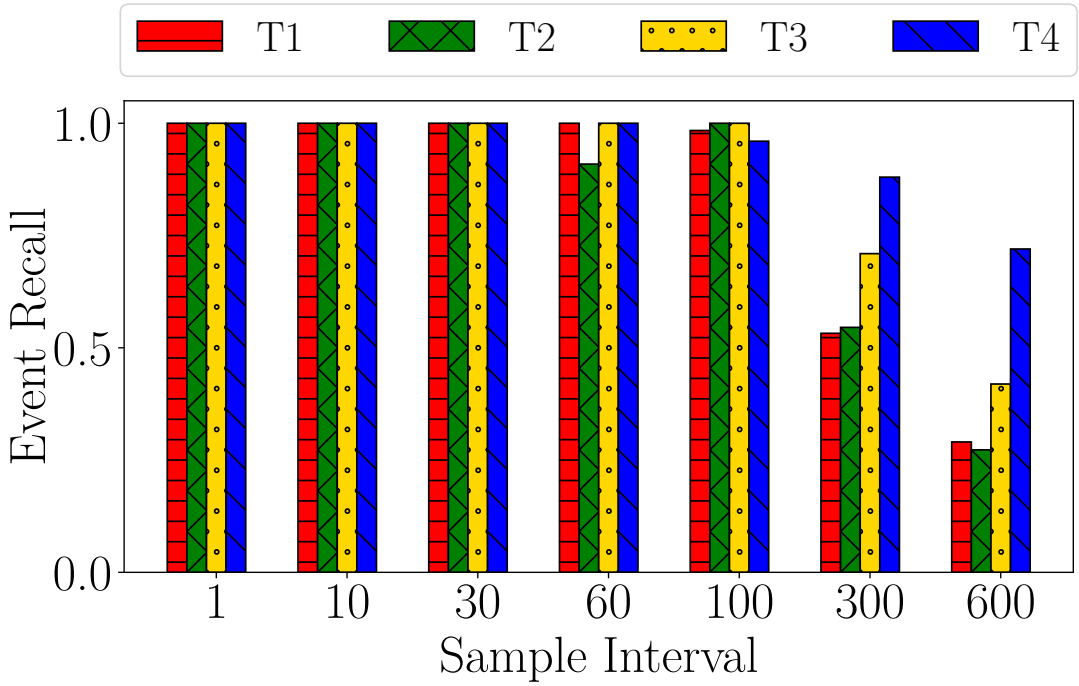



Figure 3.9: Event Recall at Different Sampling Intervals

content-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 **??** 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 (as may be needed for task T5). 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.

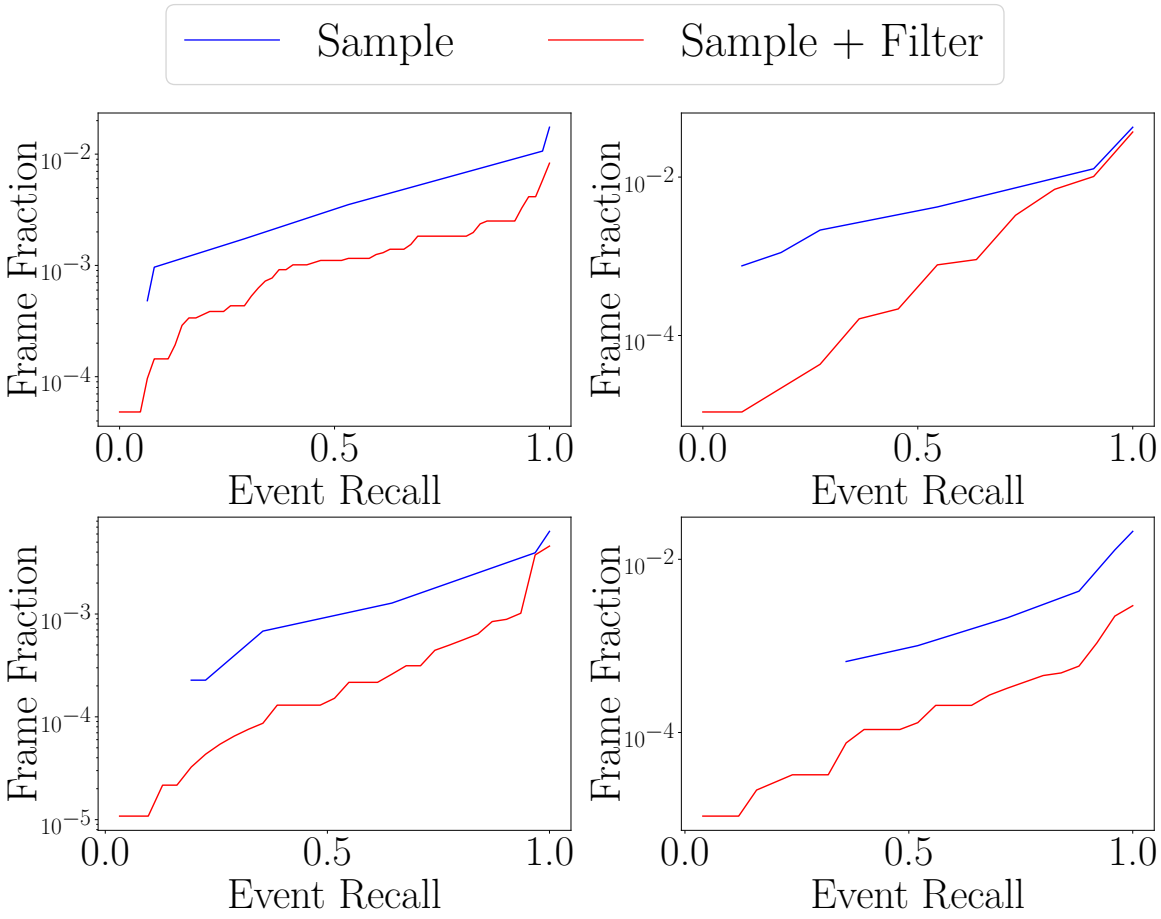



Figure 3.10: Sample with Early Discard. Note the log scale on y-axis.

### 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. Figure **??** 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.

| 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 40 [**?** ].

Figure 3.11: Test Dataset Size With Different Encoding Settings

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 see that the most accurate model (Faster-RCNN ResNet101) on low quality videos performs similarly to the less accurate model (Faster-RCNN InceptionV2) on high quality JPEG images. This negates the benefits of using the state-of-art models.

In this system, 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 the paper, 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 Strategy To Improve Early Discard

### 3.4.1 Description

Just-in-time-learning (JITL) tunes the drone pipeline to the characteristics of the current mission in order to reduce transmitted false positives from the drone, and therefore reduce wasted bandwidth. It is inspired by the cascade architecture from the computer vision community [**?** ], 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.

Periodically during a drone mission, a JITL filter is trained on the cloudlet using the frames transmitted from the drone. 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, 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 drone to run as a cascade filter after the EarlyDiscard DNN.

True/false positive frames have high temporal locality throughout a drone mission. 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 flight, 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) [**?** ] 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.

### 3.4.2 Experimental Setup

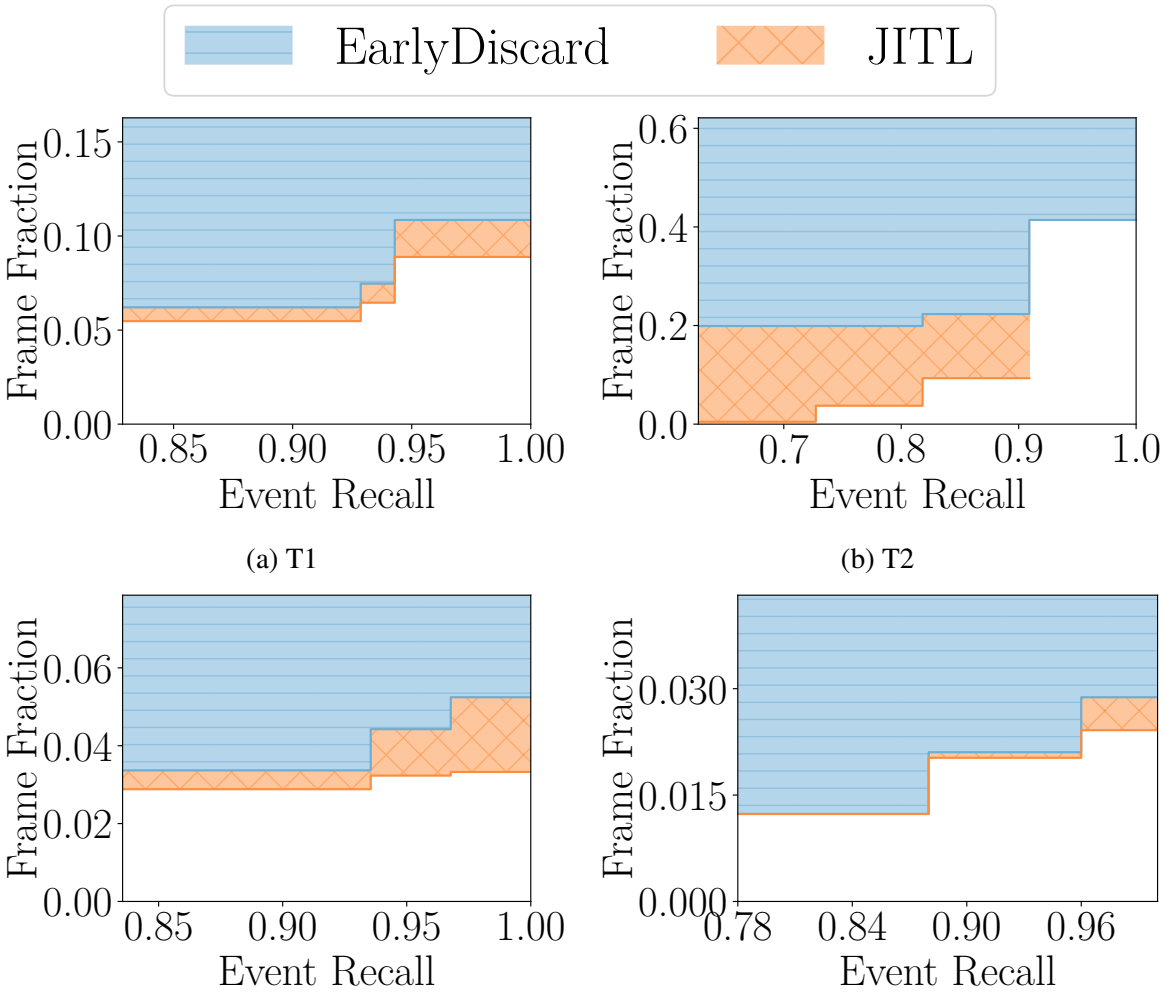
We used Jetson TX2 as our drone platform and evaluated the JITL strategy on four tasks, T1 to T4. For the test videos in each task, we began with the EarlyDiscard filter only and gradually trained and deployed JITL filters. Specifically, every ten seconds, we trained an SVM using the frames transmitted from the drone 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 drone 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.

### 3.4.3 Results

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 **??** 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 drone filter. 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 [**?** ] could be explored to synthesize true positives from the background of the current flight.

## 3.5 Applying EARLYDISCARD and JITL to WCA




(a) T1

(b) T2

(c) T3

(d) T4

Figure 3.12: JITL Fraction of Frames under Different Event Recall

# 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 **??**. 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 **??**, 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 **??**, offered load can be further reduced with application assistance. Scalability at the edge can thus be better achieved for applications that have been designed with this goal in mind. We refer to applications that are specifically written for edge

computing as *edge-native applications.* These applications are deeply dependent on 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. In earlier work [**?** ], we have shown that even modest computation 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 [**?** ].

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 [**? ?** ], shown in Figure **??**. 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.

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

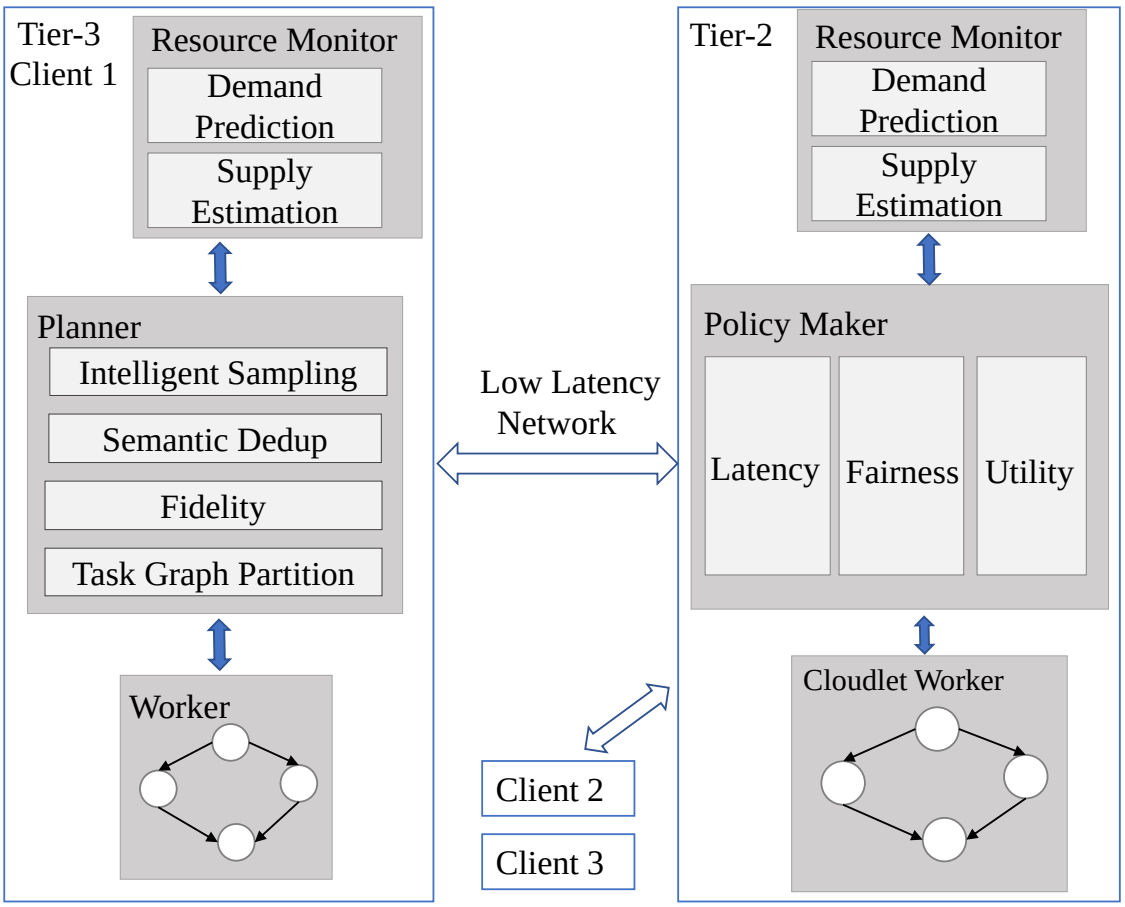



Figure 4.1: System Architecture

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 **??** for more details). We both approaches, and combine them using the system architecture shown in Figure **??**. We assume benevolent and collaborative clients in the system.

## 4.2 System Architecture

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 **??**.

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 **??**) 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 **??**), drilling a hole in bone can take several minutes

| | **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

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 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 **??**. Using these, we construct a taxonomy of WCA applications (shown in Figure **??**), based on clusters of applications along dimensions induced from the checklist of questions. In

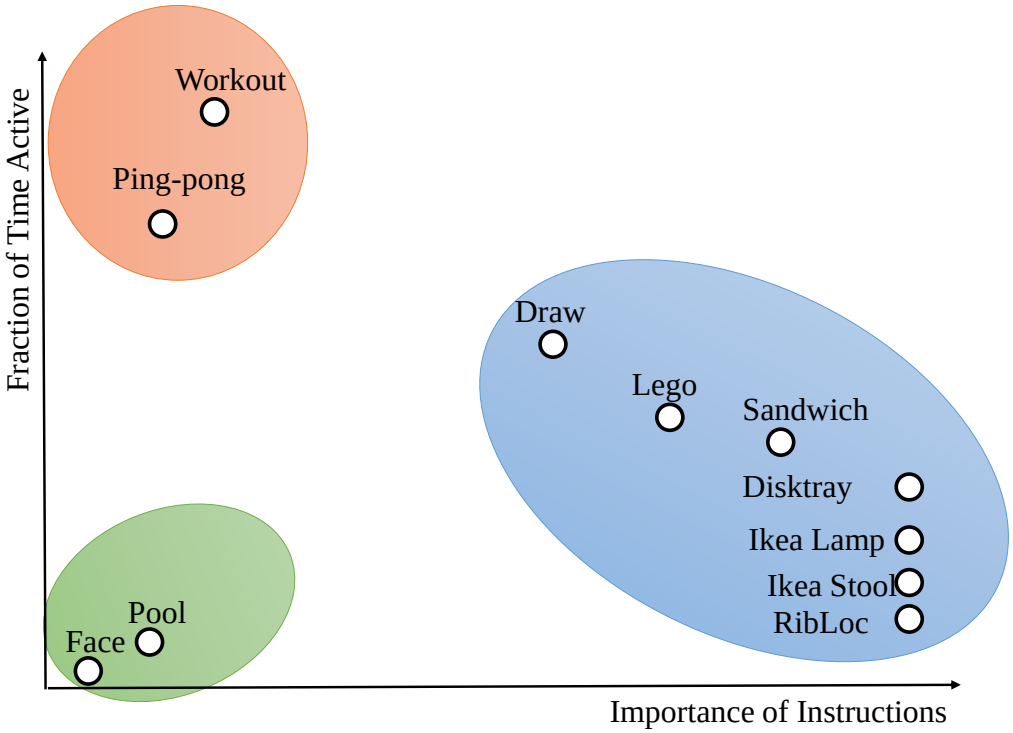



Figure 4.2: Design Space of WCA Applications

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.

## 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 **??**), 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

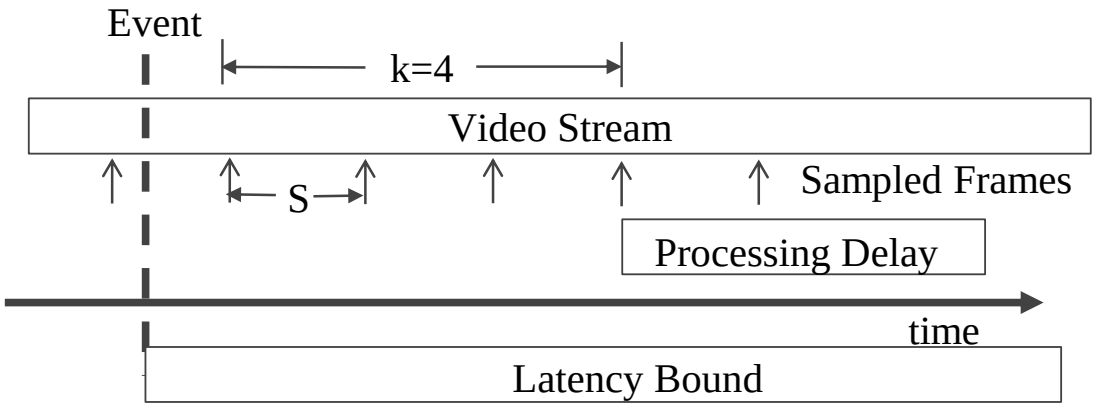



Figure 4.3: Dynamic Sampling Rate for LEGO

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.

We use the LEGO application from Section **??** 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.

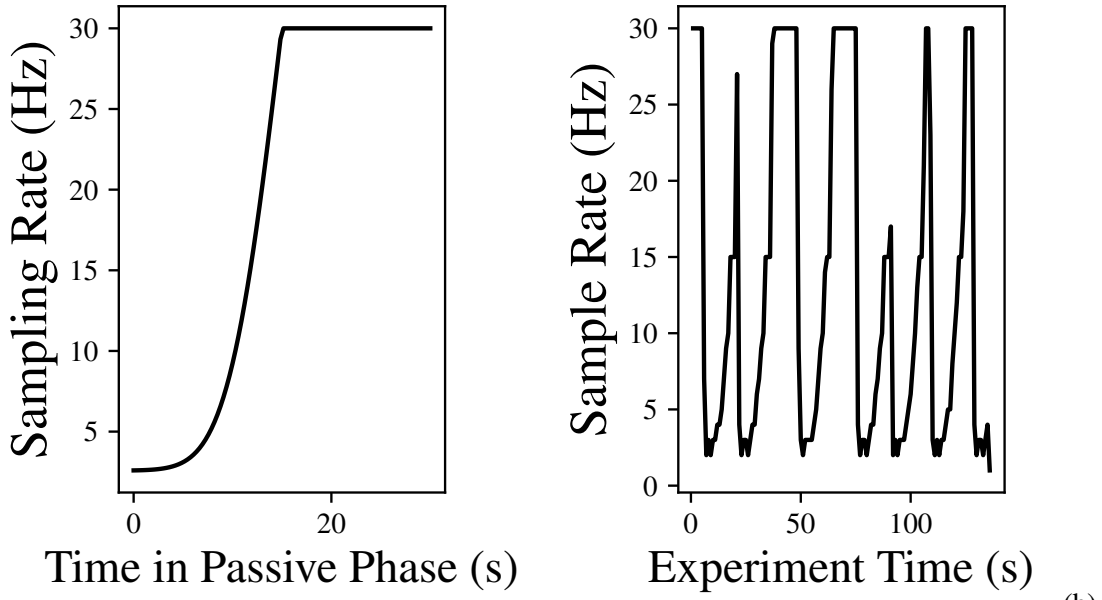
**Length of a Passive Phase:** We model the time it takes to finish each step as a Gaussian distribution. We use maximum likelihood estimation to calculate the parameters of the Guassian model.

**Lowest Sampling Rate in Passive Phase:** The lowest sampling rate in passive phase still needs to meet application's latency requirement. Figure **??** 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




(a) Passive Sampling Rate (b) Trace Sampling Rate

Figure 4.4: Adaptive Sampling Rate

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 **??**(a) shows the sampling rate adaptation our system employs during a passive phase. The sampling rate is calculated as

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

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

Figure **??**(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 **??** 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

| 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 | $7.6 \pm 6.9$ |
| Adaptive Sampling | $5.9 \pm 8.2$ |

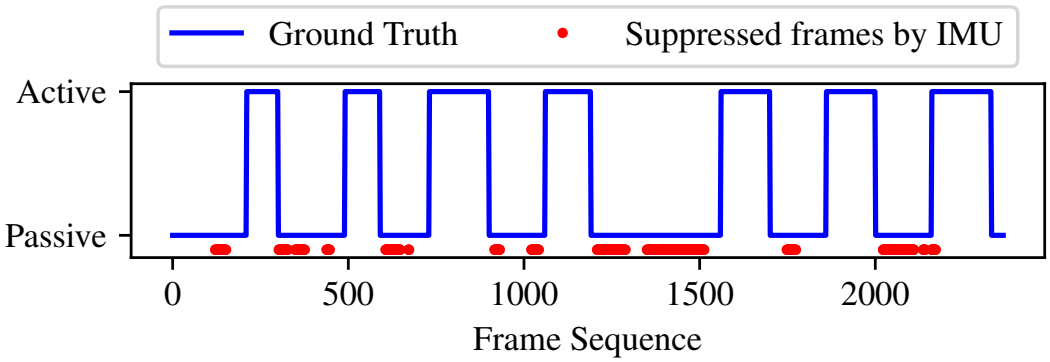
(b) Guidance Latency

Table 4.2: Frames Sampled and Guidance Latency

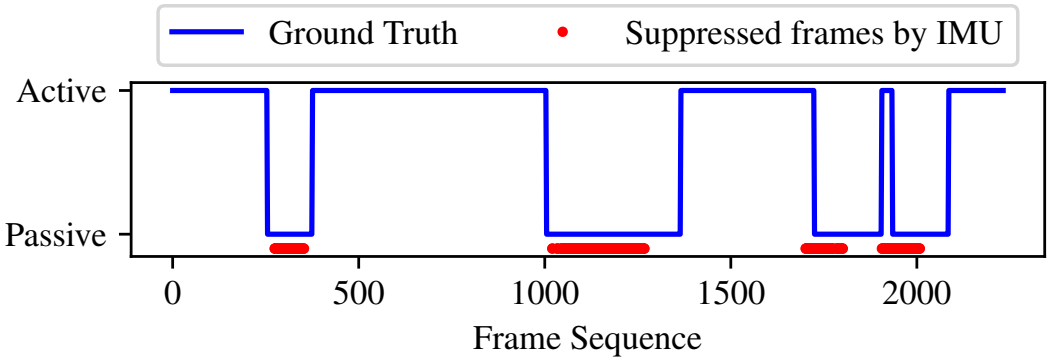
algorithm would detect no meaningful task states if the frames are transmitted. In PING PONG application (Section **??**), an active phase lasts throughout a rally. Passive phases are in between actual game play, when the user takes a drink, switches sides, or, most commonly, tracks down and picks up 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 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 **??**(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 **??**, 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.

Odyssey [**?** ] and extensions [**?** ] 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* [**?** ]; 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.

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 [**? ? ? ?** ]. Semantic deduplication to reduce redundant processing of frames have been suggested by [**? ? ? ?** ]. Similarly, previous works have looked at suppression based on motion either from video content [**? ?** ] or IMUs [**?** ]. Others

| | 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

have investigated exploiting multiple deep models with accuracy and resource tradeoffs [**?** **?** ]. 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 [**?** **?** **?** ]. 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 [**?** ], throughput-oriented systems [**?** **?** ], for interactive applications [**?** **?** ], and from programming model perspectives [**?** ]. We believe that these approaches will become important techniques to scale applications on heavily loaded cloudlets.

# Chapter 5

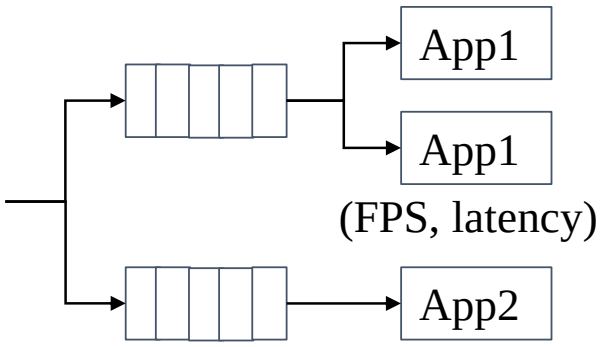
# Cloudlet Resource Management for Graceful Degradation of Service

## 5.1 System Model and Application Profiles

A complementary method to improve scalability is through judicious allocation of cloudlet resources among concurrent application services. 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.

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 **??**, 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




Only request flow is shown.

Figure 5.1: Resource Allocation System Model

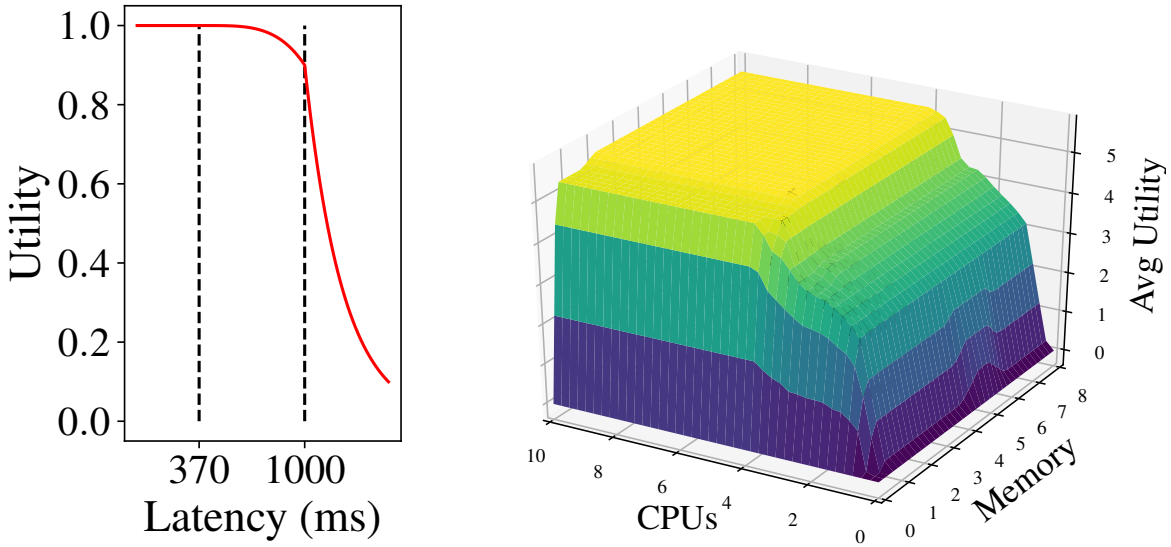
on this information, we construct offline application profiles that map multidimensional resource allocations to application QoS metrics. At runtime, we calculate a resource allocation plan to maximize a system-wide metric (e.g., total utility, fairness) specified by cloudlet owner. We choose to consider 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 **??** shows the system model we consider. Each application is given a separate input queue. Each queue can feed one or more application instances. Each application instance is encapsulated in a container with controlled resources. In this model, with 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 **??** 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 [**?** ].

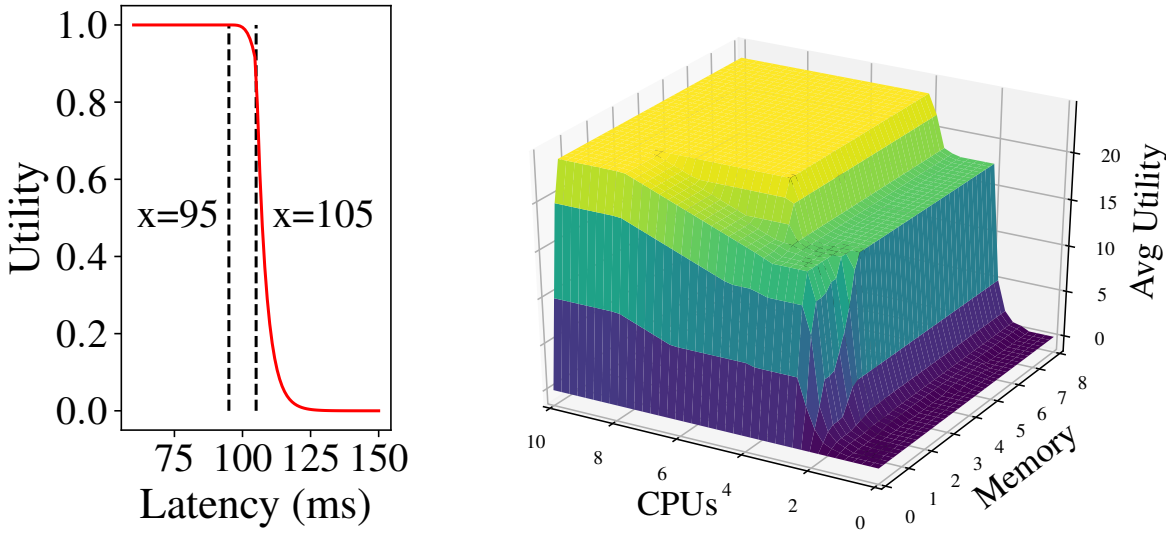
The system-wide performance is a function of the following independent variables: (a) the number of applications and the number of clients of each application; (b) the number of instances of each application; and, (c) the resource allocation for each instance. Although (a) is not under our control, Gabriel 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.




(a) Utility For FACE

(b) Profile for FACE

Figure 5.2: FACE Application Utility and Profile




(a) Utility For POOL

(b) Profile for POOL

Figure 5.3: POOL Application Utility and Profile

### 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 **??**(a) and Figure **??**(a) show utility functions for two applications based on latency bounds identified by [**?** ] 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 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 **??**(b) and Figure **??**(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 described in the Gabriel framework [**?** ], 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:

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

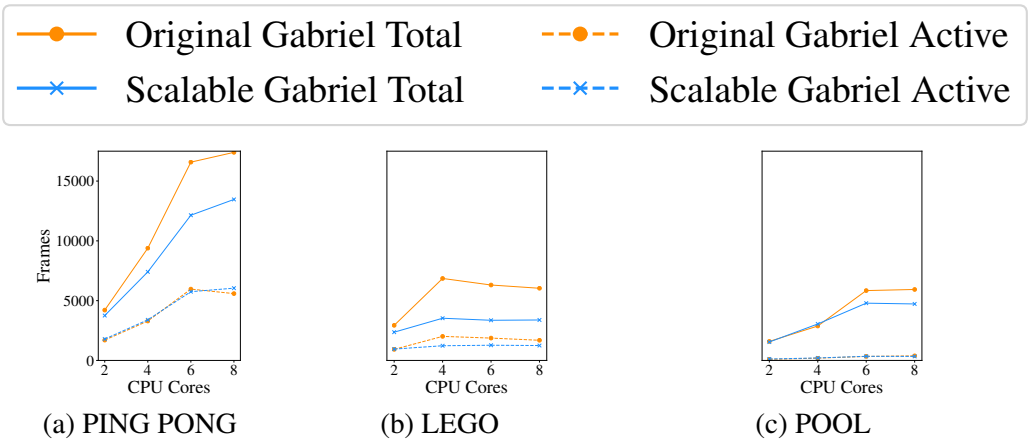
In above, $c_a$ is the number of mobile clients requesting service from application $a$. The total resource vector of the cloudlet is $\hat{\mathbf{r}}$. For each application $a$, we determine how many instances to launch — $k_a$, and allocate resource vector $\mathbf{r}_a$ to each of them. A tunable knob $\gamma$ 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 $\gamma$ 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 follows: For each application profile $u_a(\mathbf{r})$, we find the resource point that gives the highest $\frac{u_a(\mathbf{r})}{|\mathbf{r}|}$, i.e., *utility-to-resource* ratio. Denote this point as $\mathbf{r}_a^*$. We start with the application with the largest $\frac{u_a(\mathbf{r}_a^*)}{|\mathbf{r}_a^*|}$. We allocate $k_a$ application instances, each with resource $\mathbf{r}_a^*$, such that $k_a$ is the largest integer with $k_a \cdot u_a(\mathbf{r}_a^*) \leq \gamma \cdot c_a$. If there is leftover resource, we move to the application with the next highest utility-to-resource ratio and repeat the process.

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 [**?** ] [**?** ]. 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.




(a) PING PONG (b) LEGO (c) POOL

Figure 5.4: Effects of Workload Reduction

| 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 Experiments

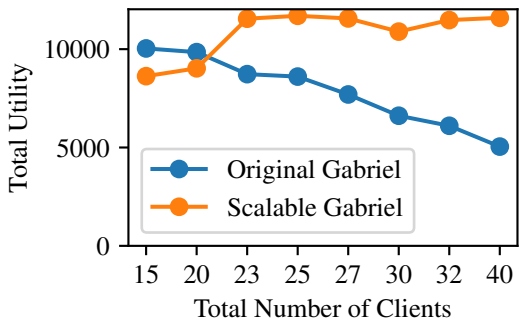



Figure 5.5: Total Utility with Increasing Contention

### 5.3.1 Effectiveness of Workload Reduction

We first evaluate the effectiveness of all of the workload reduction techniques explored in Section **??**. For this set of experiments, we do not use multiple concurrent applications or resource allocation. We use four Nexus 6 mobile phones as clients, connecting to a cloudlet over a Wi-Fi link. We run PING PONG, LEGO, and POOL applications one at a time with 2, 4, 6, and 8 cores available on the server. Figure **??** shows the total number of frames processed with and without workload reduction. 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 resource allocation 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 increasing numbers of clients across four concurrent applications with a total of 15 to 40 clients. The breakdown of the number of clients used for each experiment is given in Table **??**. We use offline generated application profiles discussed in Section **??** to optimize for total system utility. Figure **??** shows how the system-wide total utility changes as we add more clients to the workload, under the original Gabriel approach and the scalable Gabriel approach. We see that original Gabriel's total utility drops more than 40% as contention increases, 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 utility profiles.

Figure **??** and Figure **??** provide insights into how scalable Gabriel strikes the balance. 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 25 clients.

### 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 **??** and Fig **??** 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. Figure **??** shows boxplots of per-application guidance latency for the concurrent application experiments above. The red 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 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. At the highest number of clients, scalable Gabriel sacrifices the LEGO application to maintain the quality of service for the other two. This differentiated allocation is reflected in Figure **??**. In contrast, with original Gabriel, none of the applications are able to regularly meet deadlines.

## 5.4 Related Work

Although edge computing is new, the techniques for scalability examined in this paper bear some resemblance to work that was done in the early days of mobile computing, and more recent cloud management work.

Odyssey [**?** ] and extensions [**?** ] proposed upcall-based collaboration between a mobile's operating system and its applications to adapt to variable wireless connectivity and limited battery. Exploration of tradeoffs between application fidelity and resource demand led to the concept of *multi-fidelity applications* [**?** ]; 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.

Several different approaches to adapting application fidelity have been studied. Dynamic sampling rate with various heuristics for adaptation have been tried primarily in the contexts of individual mobile devices for energy efficiency [**?** **?** **?** **?** ]. Semantic deduplication to reduce redundant processing of frames have been suggested by [**?** **?** **?** **?** ]. Similarly, previous works have looked at suppression based on motion either from video content [**?** **?** ] or IMUs [**?** ]. Others have investigated exploiting multiple deep models with accuracy and resource tradeoff [**?** **?** ]. 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 [**?** ], throughput-oriented systems [**?** **?** ], for interactive applications [**?** **?** ], and from programming model perspective [**?** ]. We believe that these approaches will become important techniques to scale applications on heavily loaded cloudlets.

Dynamic resource allocation schemes on the cloud for video processing have been well ex-

plored [**? ? ?** ]. More recently, profile-based adaptation of video analytics [**? ? ?** ] focused on throughput-oriented analytics application on large clusters or cloud. In contrast, our goals focus on interactive performance on relatively small edge deployments.

## 5.5 Conclusion and Future Work

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, as well as tools and techniques to easily create such applications.

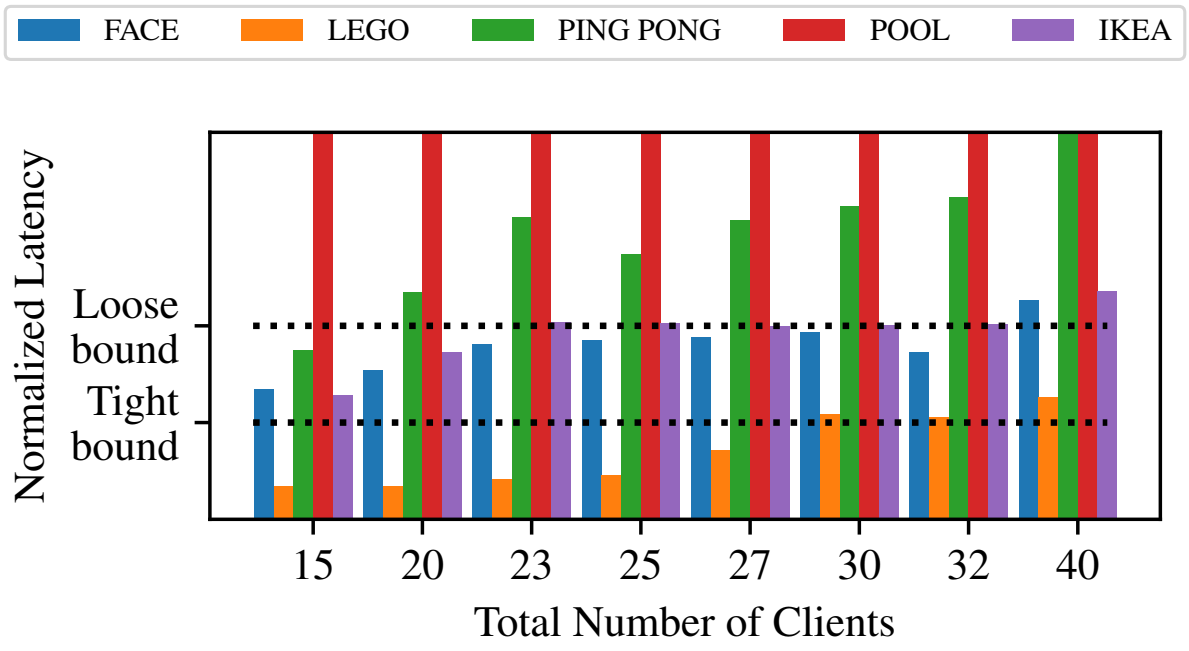
The emergence of edge computing leads to another inflection point in application design. In particular, it leads to "edge-native applications" that are deeply dependent on attributes such as low latency or bandwidth scalability that can only be obtained at the edge. However, as this paper has shown, edge-native applications have to be written in a way that is very different from cloud-native applications if they are to be scalable.

The is the first work to show that cloud-native implementation strategies that focus primarily on dynamic scale-out are unlikely to be effective for scalability in edge computing. Instead, edge-native applications need to adapt their network and cloudlet resource demand to system load. As the total number of Tier-3 devices associated with a cloudlet increases, the per-device network and cloudlet load has to decrease. This is a fundamental difference between cloud-native and edge-native approaches to scalability.

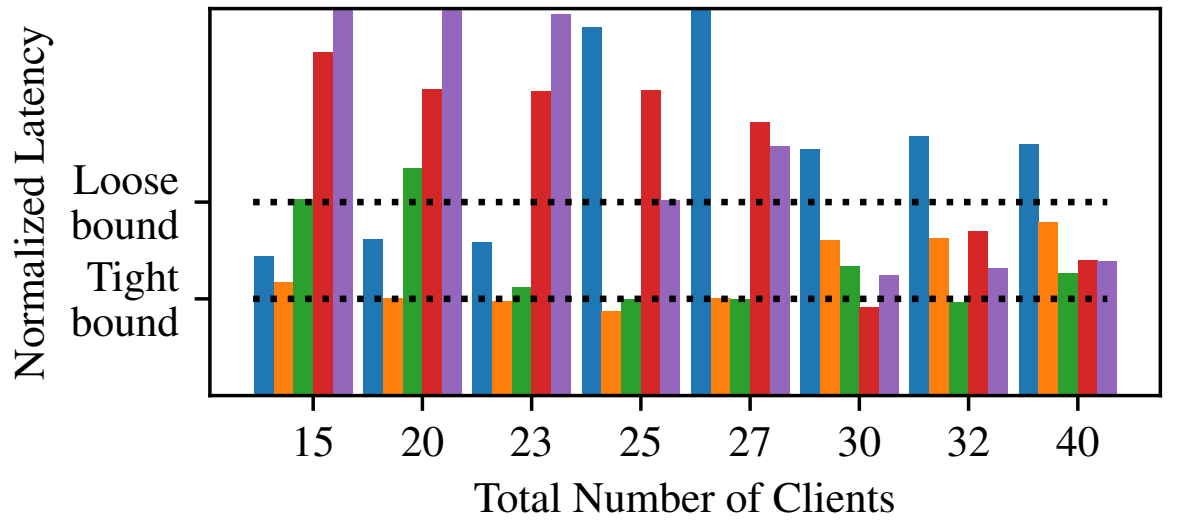
In this paper, we explore client workload reduction and server resource allocation to manage application quality of service in the face of contention for cloudlet resources. We demonstrate that our system is able to ensure that in overloaded situations, a subset of users are still served with good quality of service rather than equally sharing resources and missing latency requirements for all.

This work serves as an initial step towards practical resource management for edge-native applications. There are many potential directions to explore further in this space. We have alluded to some of these earlier in the paper. One example we briefly mentioned is 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 paper. 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 contains 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.



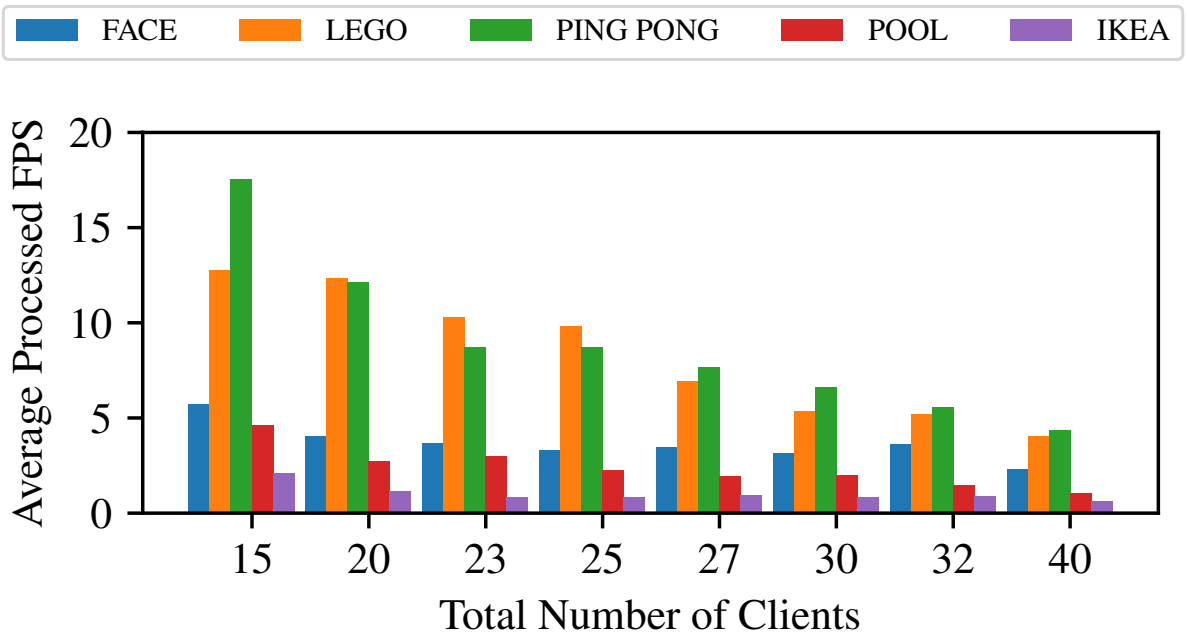

(a) Original Gabriel



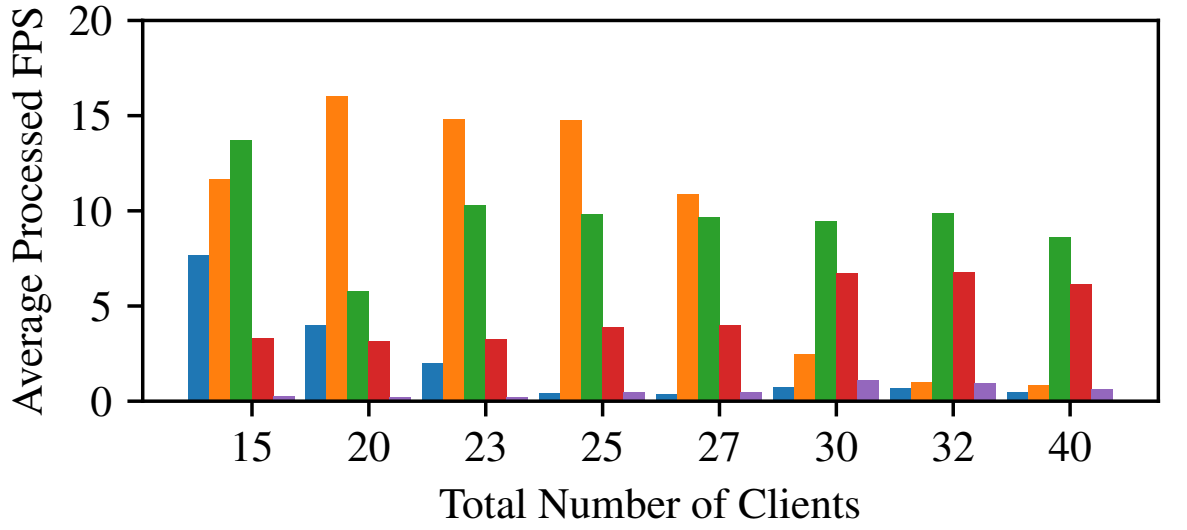

(b) Scalable Gabriel

The normalization is by per-application tight and loose bounds [**?** ]

Figure 5.6: Normalized 90%-tile Response Latency

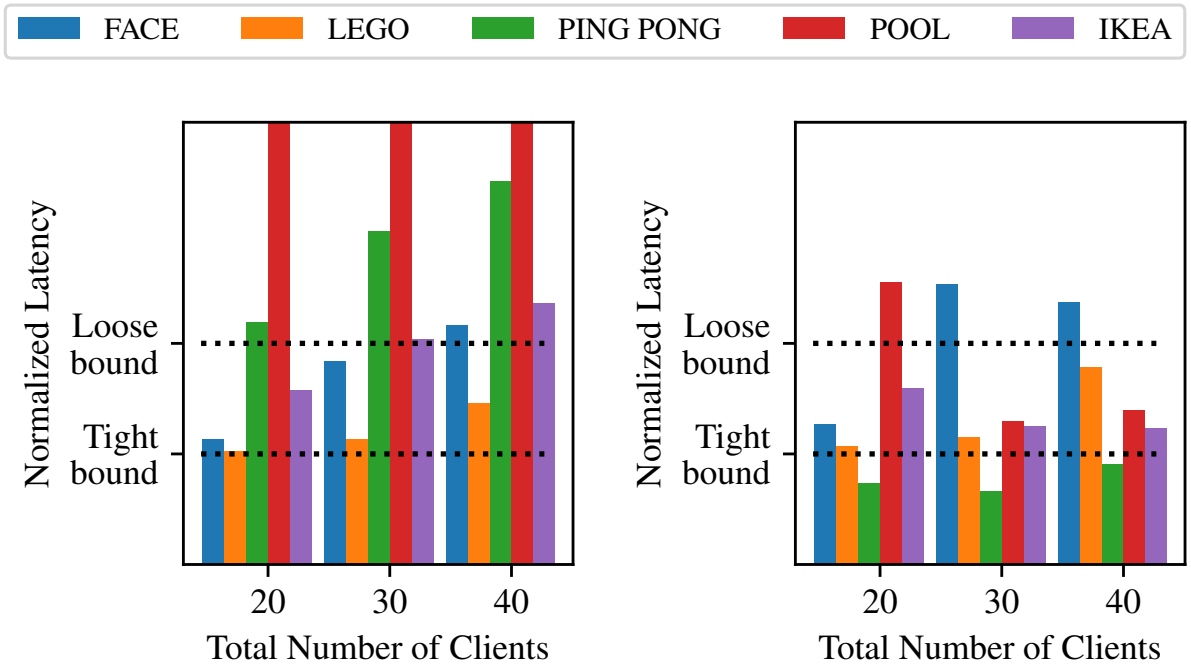



(a) Original Gabriel




(b) Scalable Gabriel

Figure 5.7: Average Processed Frames Per Second Per Client

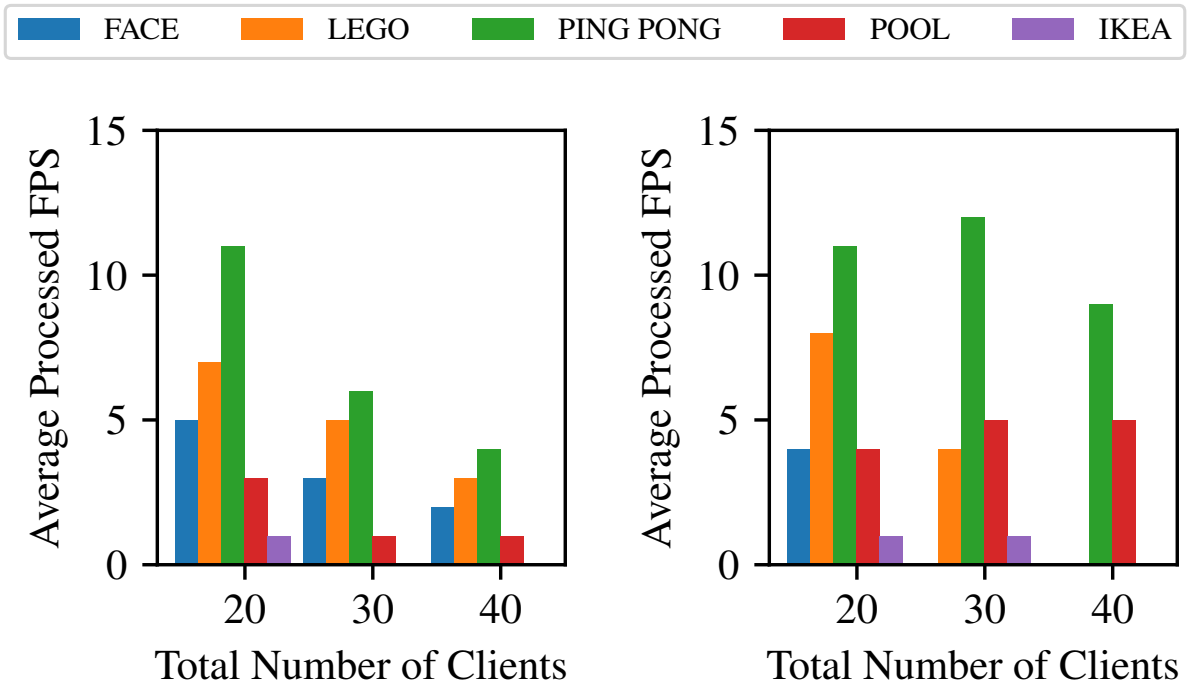



(a) Original Gabriel

(b) Scalable Gabriel

The normalization is by per-application tight and loose bounds [**?** ]

Figure 5.8: Normalized 90%-tile Response Latency




(a) Original Gabriel

(b) Scalable Gabriel

Figure 5.9: Processed Frames Per Second Per Application

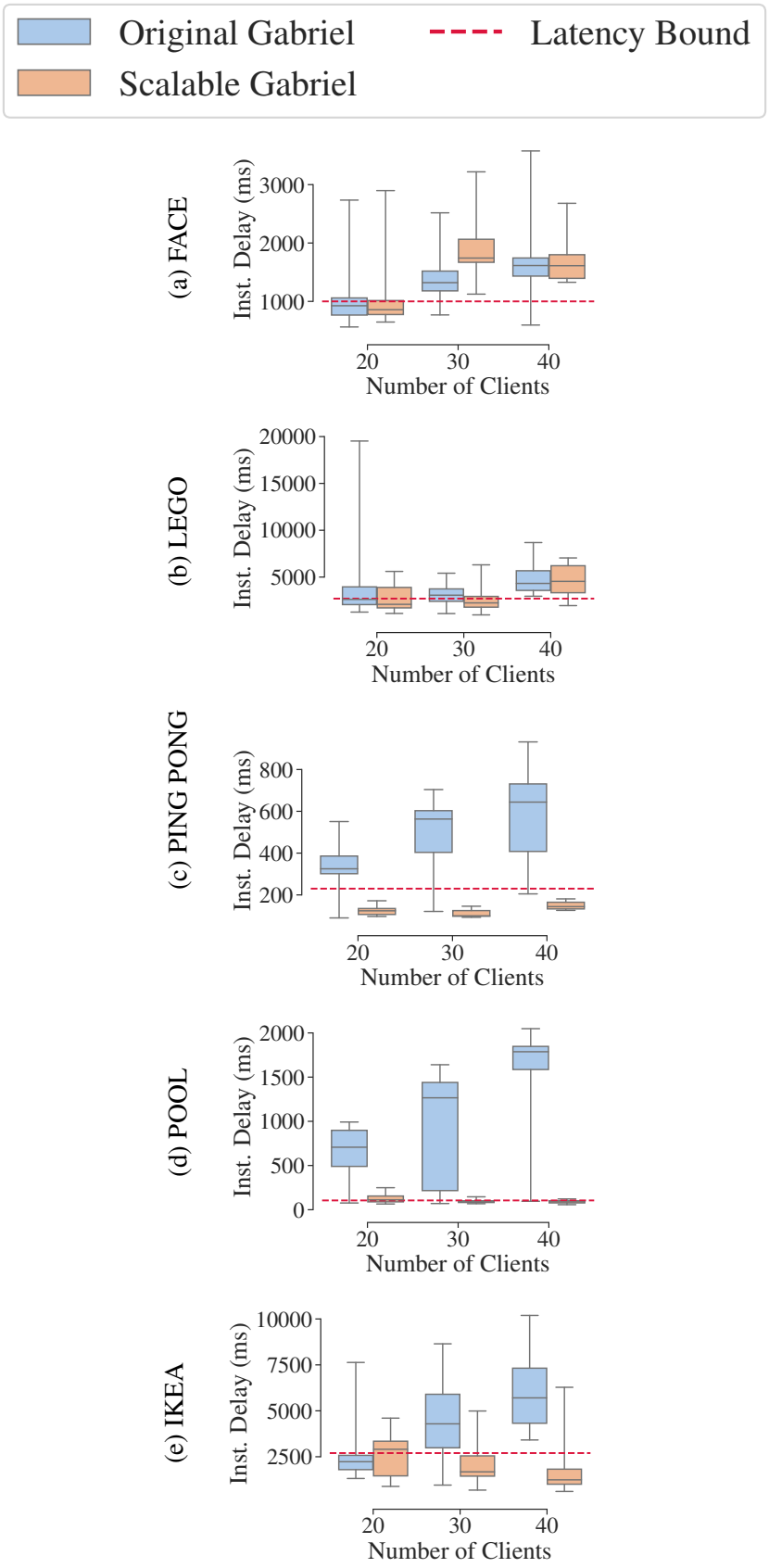



Figure 5.10: Guidance Latency

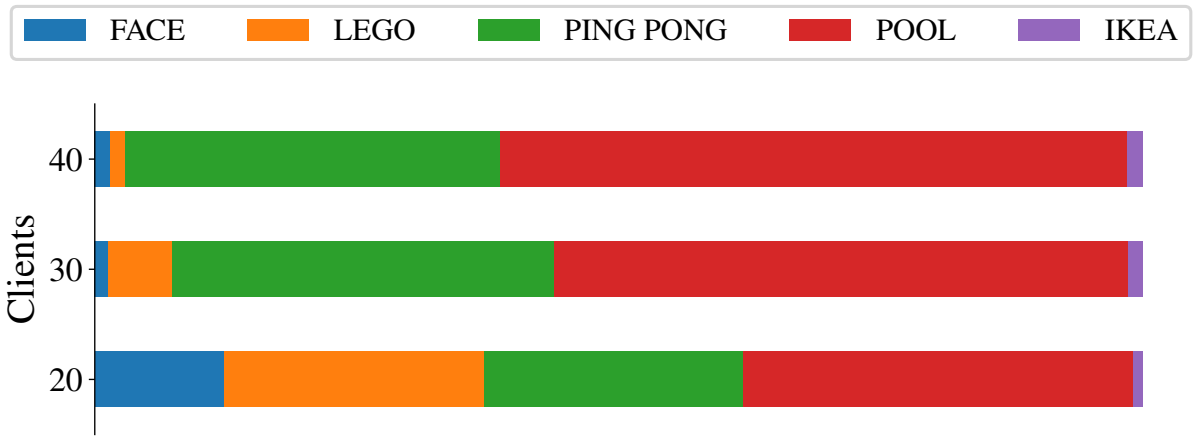



Figure 5.11: Fraction of Cloudlet Processing Allocated

# Chapter 6

# Simplifying Application Development

## 6.1 Tools For Painless Object Detection (TPOD)

## 6.2 Finite State Machine Authoring Tools

## 6.3 Discussion

# Chapter 7

# Simplifying Application Deployment

## 7.1 Gabriel Deployment System

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

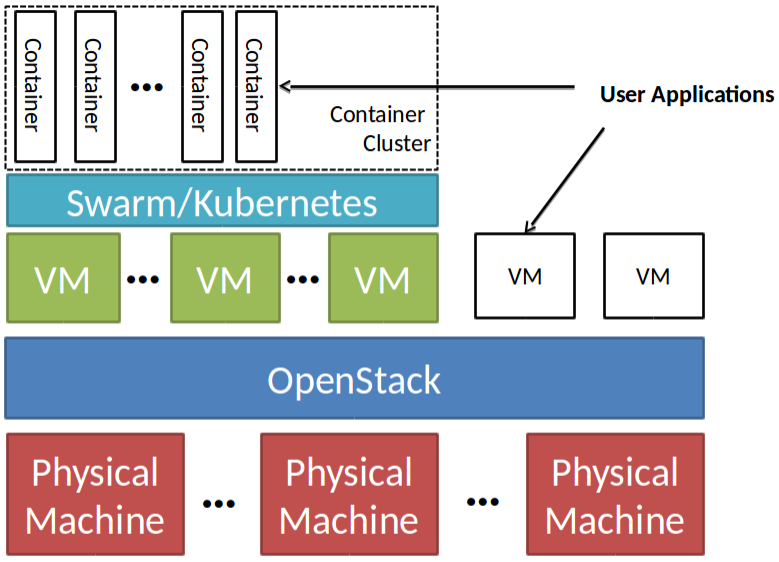



Figure 7.1: Container and Virtual Machine Virtualization on Cloudlets

Virtual machines (VMs) have been used to provide isolation among edge node tenants [**?** ]. While VMs virtualize at the hardware level and provide strong isolation among tenants, they also have a large footprint since each virtual machine has its own kernel. In the meantime, some applications do not need the strong isolation provided by VMs. For example, applications

published by the same developer or company would have stronger trusts about the integrity of the software. Therefore, for these applications, using more lightweight virtualization constructs, for instance, containers, can help reduce the application footprint on cloudlets.

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

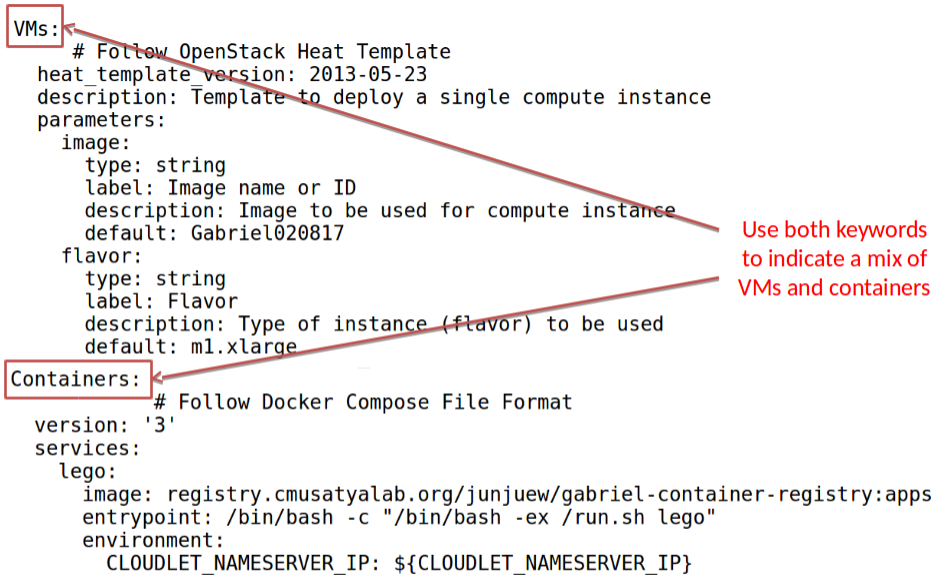



Figure 7.2: Application Deployment Template with Virtualization Type Specified

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

## 7.2 Cloudlet Gateway

## 7.3 Gabriel Deployment

## 7.4 Discussion

# Chapter 8

# Conclusion and Future Work