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.

Contents

Chapter 1

Introduction

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

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

To meet the latency and the compute requirements, previous research leverages edge computing and offloads computation to a cloudlet. A cloudlet [**?**] 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 applica-

tions as benchmarks, [**?**] presents empirical measurements detailing the latency contributions of individual system components. Furthermore, a multi-algorithm approach was proposed to reduce the latency of computer vision computation by executing multiple algorithms in parallel and conditionally selecting a fast and accurate algorithm for the near future.

While previous research has demonstrated the technical feasibility of wearable cognitive assistants and meeting latency requirements, many practical concerns have not been addressed. First, previous work operates the wireless networks and cloudlets at low utilization in order to meet application latency. The economics of practical deployment preclude operation at such low utilization. In contrast, resources are often highly utilized and congested when serving many users. How to efficiently scale Gabriel applications to a large number of users remains to be answered. Second, previous work on the Gabriel framework reduces application development efforts by managing client-server communication, network flow control, and cognitive engine discovery. However, the framework does not address the most time-consuming parts of creating a wearable cognitive assistance application. Experience has shown that developing computer vision modules that analyze video feeds is a time-consuming and painstaking process that requires special expertise and involves rounds of trial and error. Developer tools that alleviate the time and the expertise needed can greatly facilitate the creation of these applications.

The core contribution of this thesis is to holistic improve the scalability of wearable cognitive assistance. Scalability, in this thesis, is considered from three facets. First, from a traditional distributed system perspective, a scalable system is one that enables most associated clients with fixed amount of infrastructure and has ways to serve more clients as resources increase. Second, we also want to enable small software development team to quickly create these applications. Third, devOps should be able to easily deploy and manage applications on the fly on a variety of hardware.

1.1 Thesis Statement

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

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

1.2 Thesis Overview

The thesis is organized as follows.

Chapter 2

Background - Gabriel, (if needed)

Chapter 3

Application-Agnostic Techniques to Reduce Network Transmission

3.1 EARLYDISCARD Strategy

3.1.1 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 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.

FIGS/fig-training.pdf

Figure 3.1: Tiling and DNN Fine Tuning

FIGS/fig-tile-resolution-speed-accuracy.pdf

Figure 3.2: Speed-Accuracy Trade-off of Tiling

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-frames. This is done offline for training, as shown in Figure 3.1, 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.

3.1.2 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.

3.1.3 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 3.2 shows how speed and accuracy change with tile size. Accuracy improves as tiles become smaller, but the sustainable frame rate drops. We group all tiles from the same frame in a single batch to leverage parallelism, so the processing does not

change linearly with the number of tiles. The choice of an operating point will need to strike a balance between the speed and accuracy. In the rest of the 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 3.3 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 3.3 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 cannot help. The results in the third column of Figure 3.4 confirm that there are no false negatives for T1 and T4 at a cutoff threshold of 0.5. For T2 and T3, lower cutoff thresholds are needed to achieve perfect recalls.

Result latency: The contribution of early discard processing to total result latency is calculated as the average time difference between the first frame in which an object occurs (i.e., first occurrence in ground truth) and the first frame containing the object that is transmitted to the backend (i.e., first detection). The results in the fourth column of Figure 3.4 confirm that early discard contributes little to result latency. The amounts range from 0.1 s for T1 to 12.7 s for T3. At the timescale of human actions such as dispatching of a rescue team, these are negligible delays.

Bandwidth: Columns 5–7 of Figure 3.4 pertain to wireless bandwidth demand for the benchmark suite with early discard. The figures shown are based on H.264 encoding of each individual frames in the drone-cloudlet video transmission. Average bandwidth is calculated as the total data transmitted divided by mission duration. Comparing column 5 of Figure 3.4 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 3.3 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.

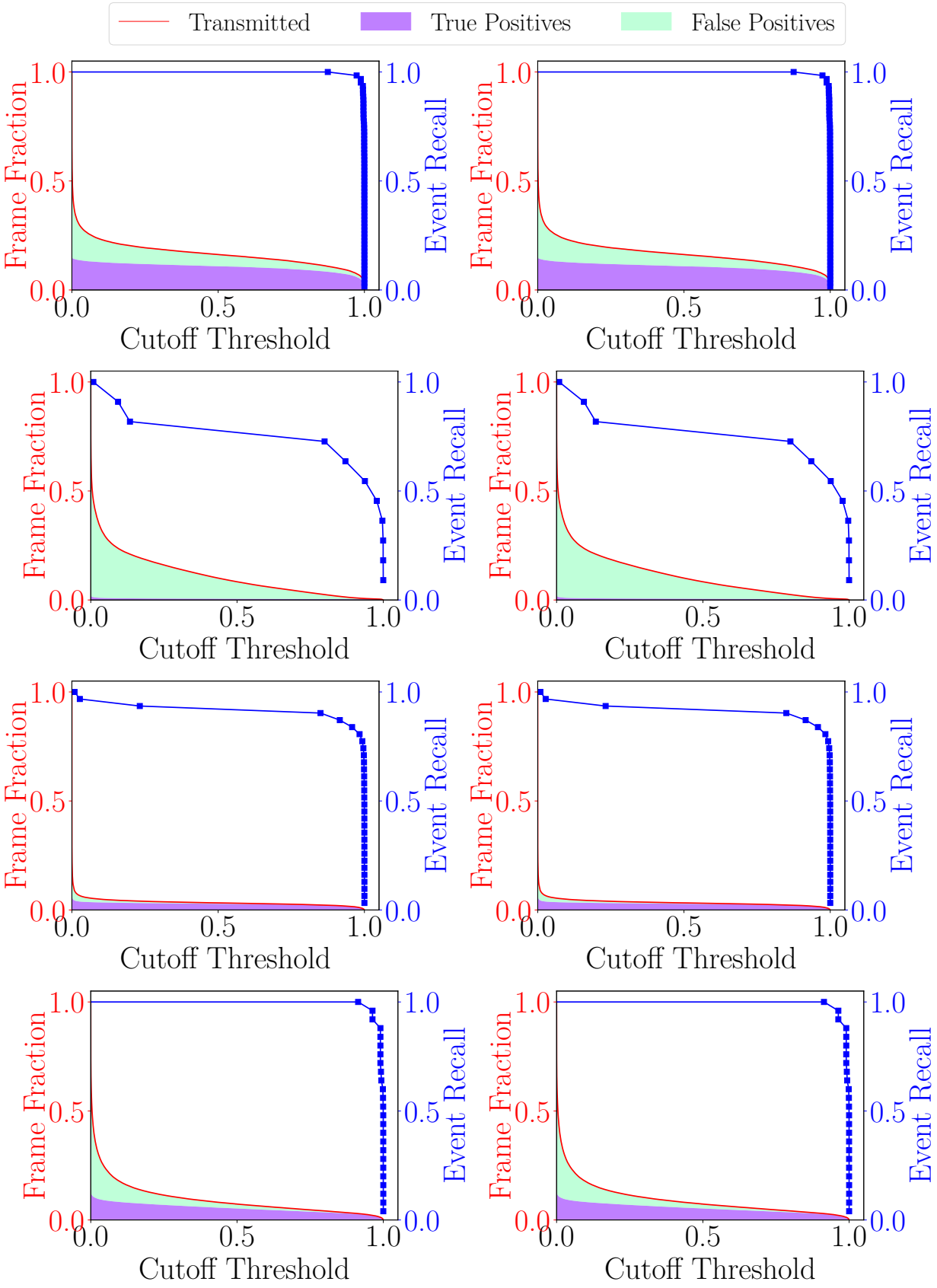


Figure 3.3: Where the Bandwidth Goes

	Task Total Events	Dete-cted 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 3.4: Recall, Event Latency and Bandwidth at Cutoff Threshold 0.5

3.1.4 Use of Sampling

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

Figure 3.5 shows the effects of sending a sample of frames from the drone, without any 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 3.6 shows results when running early discard on a sample of the frames. This shows that for the same recall, we can reduce the bandwidth consumed by another factor of 5 on average over sampling alone. This effective combination can reduce the average bandwidth consumed for our test videos to just a few hundred kilobits per second. Furthermore, more processing time is available per processed frame, allowing more sophisticated algorithms to be employed, or to allow smaller tiles to be used, improving accuracy of early discard.

One case where sampling is not an effective solution is when all frames containing an object need to be sent to the cloudlet for some form of activity or behavior analysis from a complete video sequence (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.

3.1.5 Effects of Video Encoding

One advantage of the DUMBDRONE 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

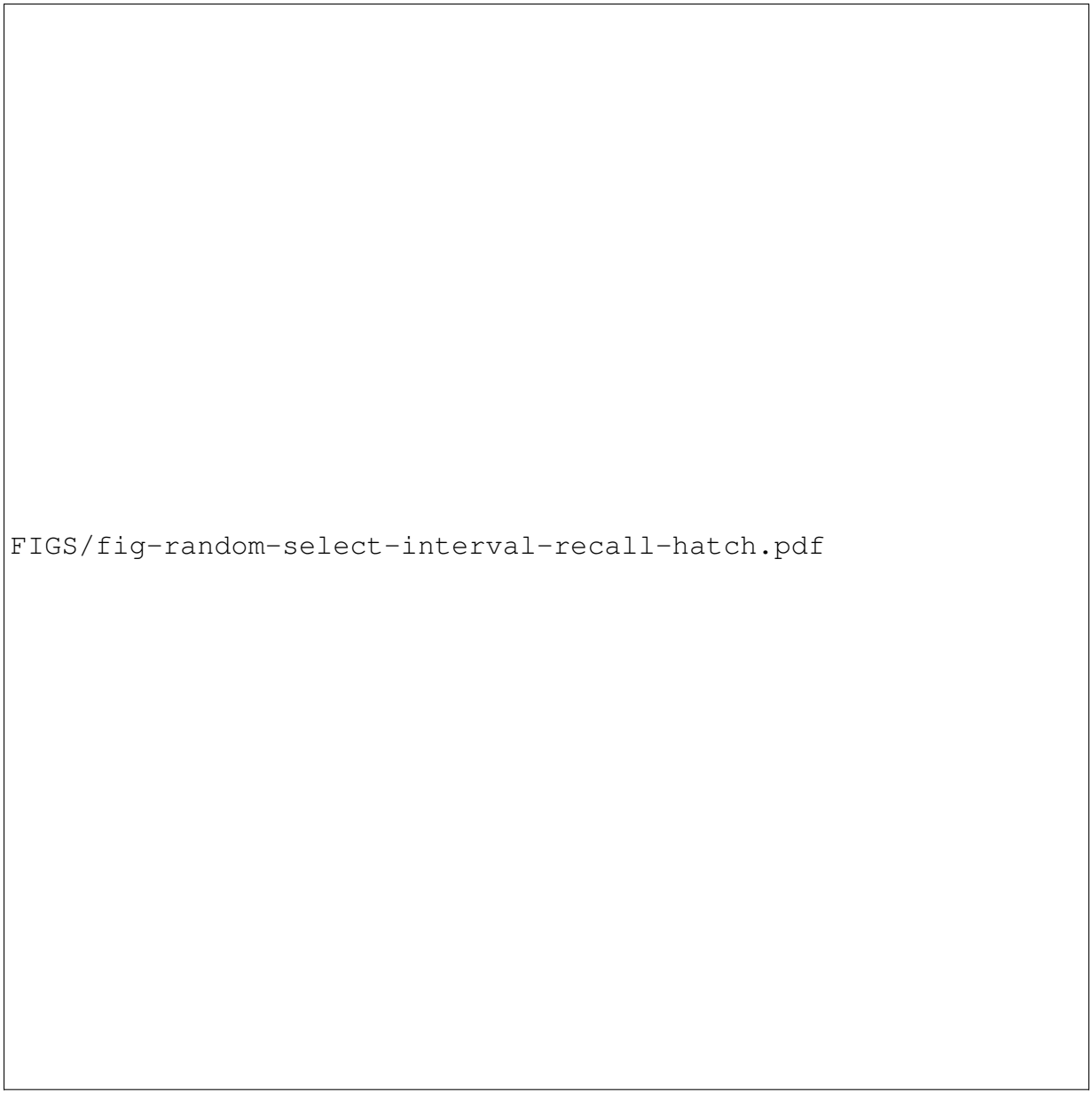
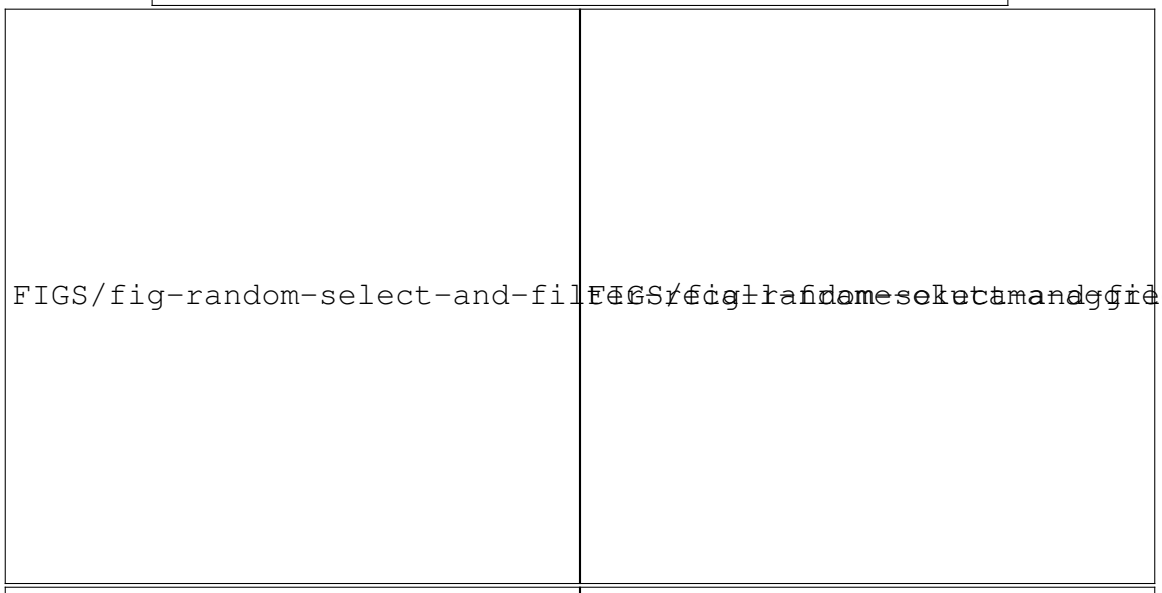
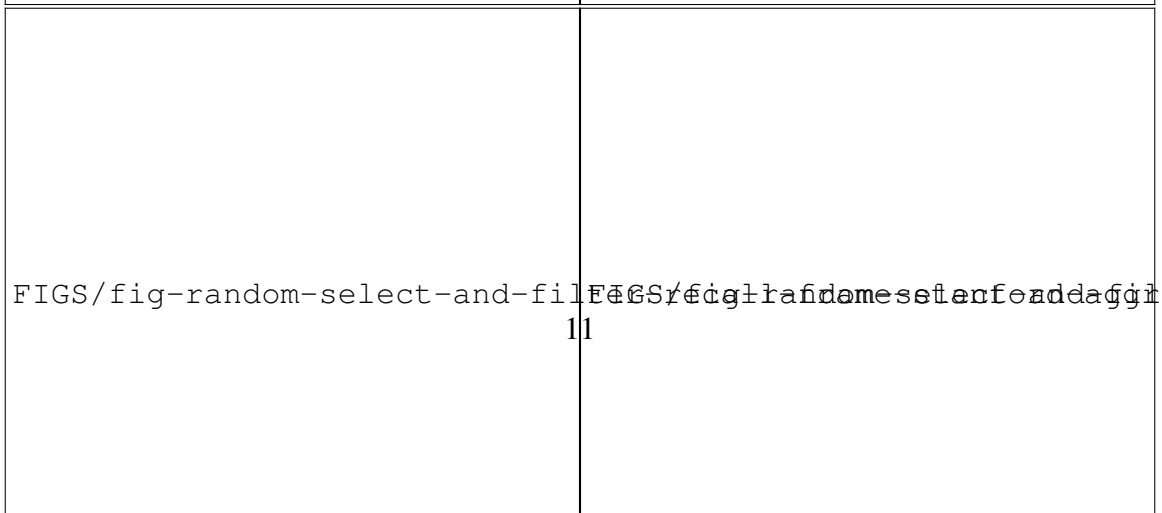


Figure 3.5: Event Recall at Different Sampling Intervals


FIGS/fig-recall-frame-aggregated-legend.pdf


FIGS/fig-random-select-and-fil
FIGS/fig-random-select-and-fil


FIGS/fig-random-select-and-fil
FIGS/fig-random-select-and-fil

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 3.7: Test Dataset Size With Different Encoding Settings

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 3.7 compares the size of frame sequences in JPEG and the encoded video file sizes. We see only about 3x difference in the data size for the medium quality. We can increase the compression (at the expense of quality) very easily, and are able to reduce the video data rate by another order of magnitude before quality degrades catastrophically.

However, this compression does affect analytics. Even at medium quality level, visible compression artifacts, blurring, and motion distortions begin to appear. Initial experiments analyzing compressed videos show that these distortions do have a negative impact on accuracy of analytics. Using average precision analysis, a standard method to evaluate accuracy, we 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.

3.2 JUST-IN-TIME-LEARNING Strategy To Improve Early Discard

3.2.1 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 EarlyDiscard 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.

3.2.2 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.

3.2.3 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 3.8 details the fraction of frames saved by JITL. The x-axis presents event recall. Y-axis represents the fraction of total frames. The blue region presents the achievable fraction of frames by EarlyDiscard. The orange region shows the additional savings using JITL. For T1, T3, and T4, at the highest event recall, JITL filters out more than 15% of remaining frames. This shows that JITL is effective at reducing the false positives thus improving the precision of the 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.

3.3 Evaluation

3.4 Discussion


FIGS/fig-jitl-legend.pdf


FIGS/fig-jitl-okutama-eventrecall-step.pdf
FIGS/fig-jitl-okutama-eventrecall-step.pdf


FIGS/fig-jitl-stanford-eventrecall-step.pdf
FIGS/fig-jitl-stanford-eventrecall-step.pdf


FIGS/fig-jitl-raft-eventrecall-step.pdf
FIGS/fig-jitl-raft-eventrecall-step.pdf

Chapter 4

Application-Aware Techniques to Reduce Offered Load

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 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. We embrace both approaches, and combine them using the system architecture shown in Figure **??**. We assume benevolent and collaborative clients in this paper, and leave the handling of malicious users to future work.

4.1.1 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

	Question	Example	Load-reduction Tech
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 sampli active and passive phase
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 f
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 f
4	Does the user have a pre-defined workspace in the scene?	Lego pieces are assembled on the lego board. Information outside the board can be safely ignored.	Focus processing atten 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 tion 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 fra vice before transmissio
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 fi 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 fra sion.
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 p tween Tier-3 and Tier-2

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

fed to a *policy module* running on the cloudlet. This module (described in detail in Section **??**) is guided by an external policy specification and determines how cloudlet resources should be allocated across competing Tier-3 applications. Such policies can factor in latency needs and fairness, or simple priorities (e.g., a blind person navigation assistant may get priority over an AR game).

A *planner module* on the Tier-3 device uses current resource utilization and predicted short-term processing demand to determine which workload reduction techniques (described in Section 4.1.3) should be applied to achieve best performance for the particular application given the resource allocations.

4.1.2 Adaptation Goals

For the applications of interest in this paper, 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.

4.1.3 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 (a medical training application), drilling a hole in bone can take several minutes to complete. During the drilling, no further guidance is provided after the initial instruction to drill. Inherently, these applications contain *active phases,* during which an application needs to sample and process video frames as fast as possible to provide timely guidance, and *passive phases,* during which the human user is busy performing the instructed step. During a passive phase, the application can be limited to sampling video frames at a low rate to determine when the user has completed or nearly completed the step, and may need guidance soon. Although durations of human operations need to be considered random variables, many have empirical 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.

4.1.4 Adaptation-Relevant Taxonomy

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

4.2 Adaptive Sampling

The processing demands and latency bounds of a WCA application can vary considerably during task execution because of human speed limitations. When the user is awaiting guidance, it is desirable to sample input at the highest rate to rapidly determine task state and thus minimize guidance latency. However, while the user is performing a task step, the application can stay in a passive state and sample at a lower rate. For a short period of time immediately after guidance is given, the sampling rate can be very low because it is not humanly possible to be done with the step. As more time elapses, the sampling rate has to increase because the user may be nearing completion of the step. Although this active-passive phase distinction is most characteristic of WCA applications that provide step-by-step task guidance (the blue cluster in the lower right of Figure 4.1), most WCA applications exhibit this behavior to some degree. As shown in the rest of this section, adaptive sampling rates can reduce processing load without impacting application latency or accuracy.

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

We use the LEGO application 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

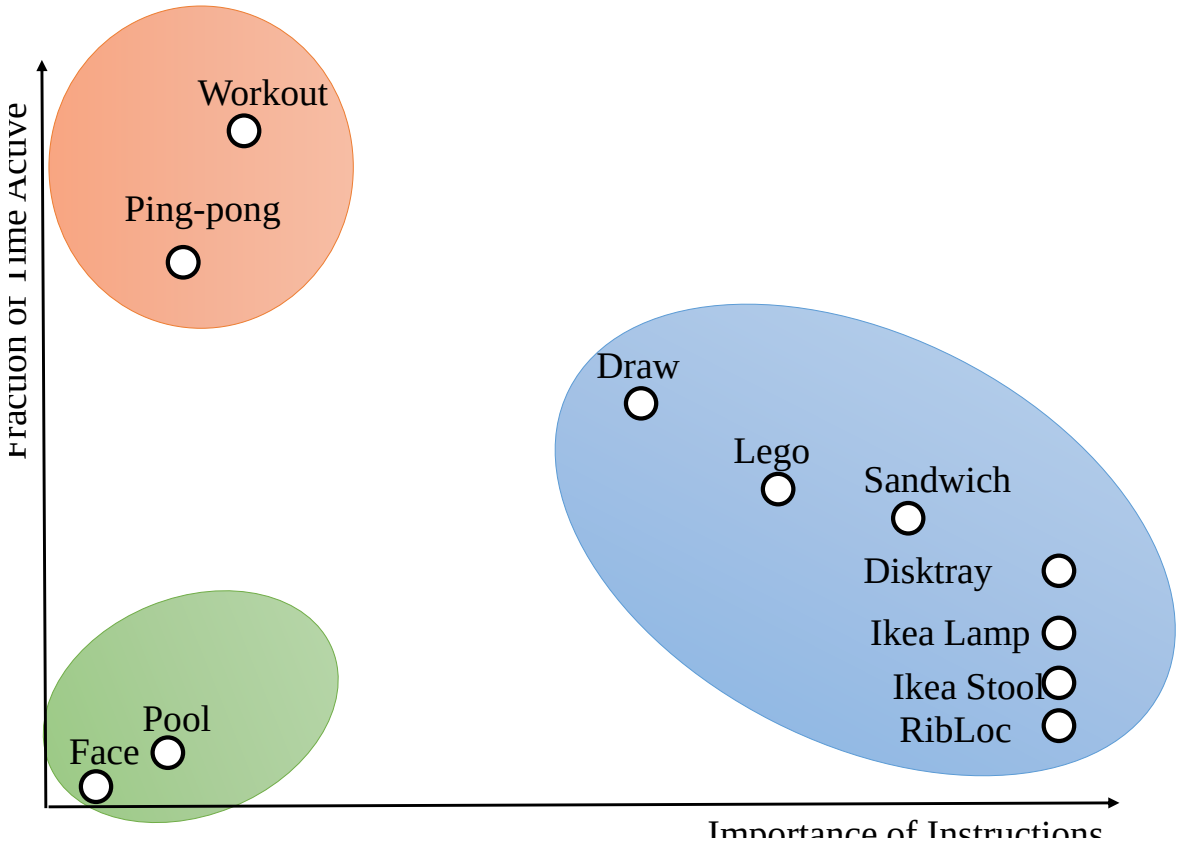


Figure 4.1: Design Space of WCA Applications

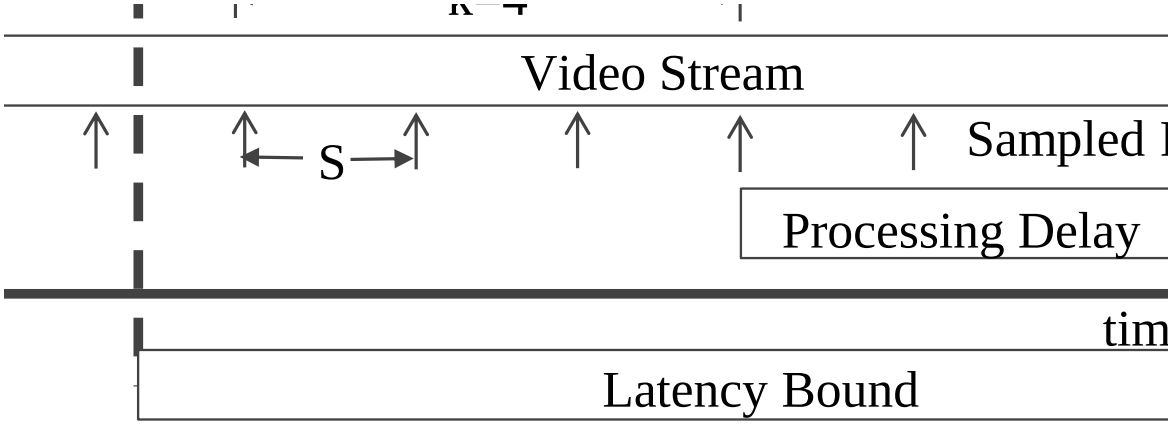


Figure 4.2: Dynamic Sampling Rate for LEGO

and assembling LEGO blocks. The length and sampling rate of a passive phase is provided by the application to the framework. We provide the following system model as an example of what can be provided. We collect five LEGO traces with 13739 frames as our evaluation dataset.

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

Lowest Sampling Rate in Passive Phase: The lowest sampling rate in passive phase still needs to meet application's latency requirement. Figure 4.2 shows the system model to calculate the largest sampling period S that still meets the latency bound. In particular,

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

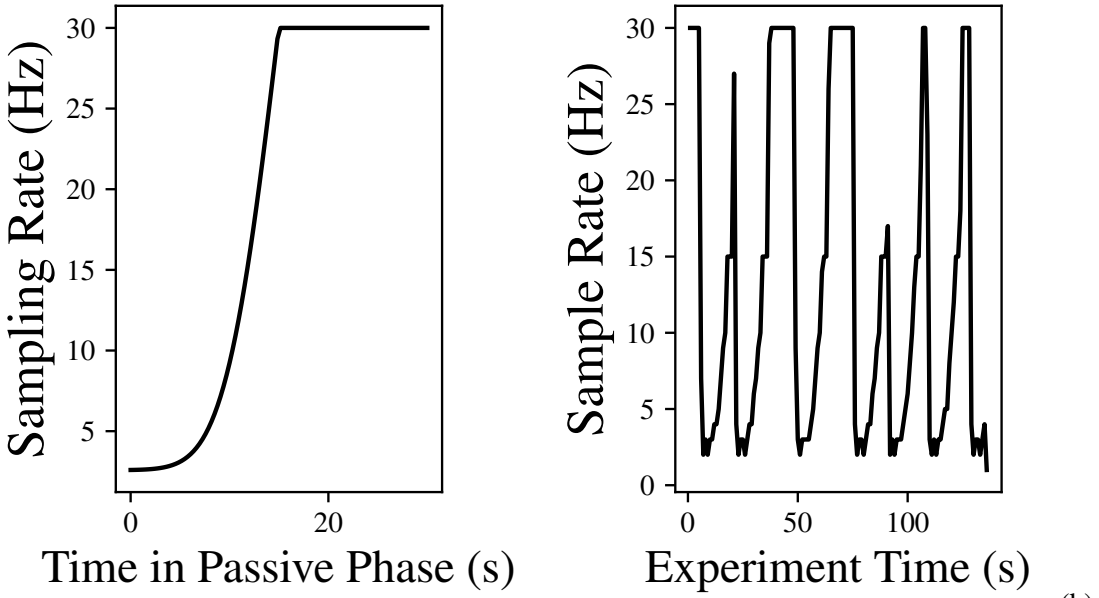
k represents the cumulative number of frames an event needs to be detected in order to be certain an event actually occurred. The LEGO application empirically sets this value to be 5.

Adaptation Algorithm: At the start of a passive phase, we set the sampling rate to the minimum calculated above. As time progresses, we gradually increase the sampling rate. The idea behind this is that the initial low sampling rates do not provide good latency, but this is acceptable, as the likelihood of an event is low. As the likelihood increases (based on the Gaussian distribution described earlier), we increase sampling rate to decrease latency when events are likely. Figure 4.3(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. α is a recovery factor which determines how quickly the sampling rate rebounds to active phase rate.

Figure 4.3(b) shows the sampling rate for a trace as the application runs. The video captures a user doing 7 steps of a LEGO assembly task. Each drop in sampling rate happens after an instruction has been delivered to the user. Table 4.2 shows the percentage of frames sampled and guidance latency comparing adaptive sampling with naive sampling at half frequency. Our adaptive sampling scheme requires processing fewer frames while achieving a lower guidance latency.



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

Figure 4.3: Adaptive Sampling Rate

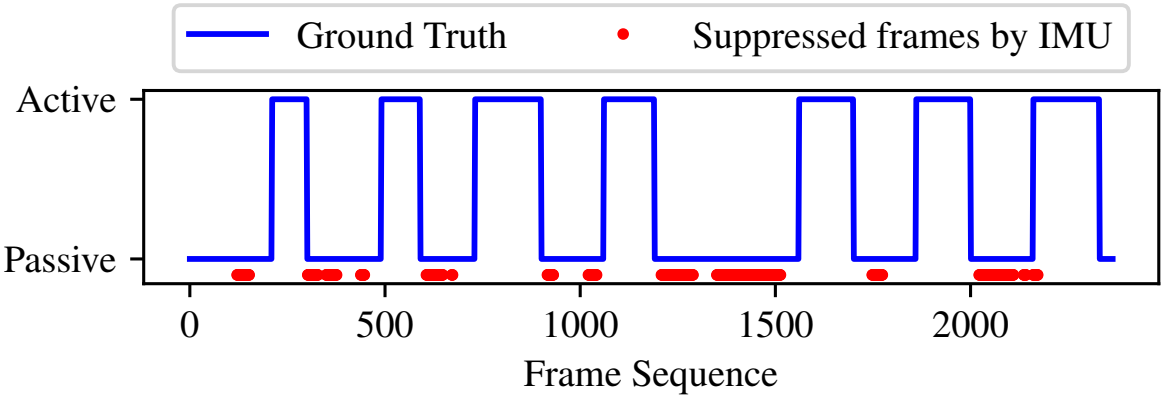
Trace	Sample Half Freq	Adaptive Sampling
1	50%	25%
2	50%	28%
3	50%	30%
4	50%	30%
5	50%	43%

(a) Percentage of Frames Sampled

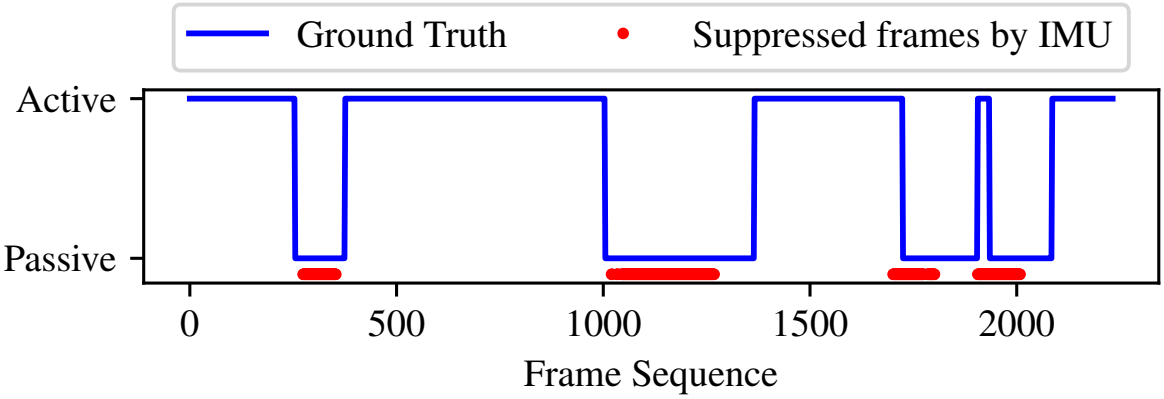
	Guidance Delay (frames±stddev)
Sample Half Freq	7.6 ± 6.9
Adaptive Sampling	5.9 ± 8.2

(b) Guidance Latency

Table 4.2: Frames Sampled and Guidance Latency



(a) LEGO



(b) PING PONG

Figure 4.4: Accuracy of IMU-based Frame Suppression

	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

4.3 IMU-based Approaches: Passive Phase Suppression + Evaluation of Image Quality

In many applications, passive phases can often be associated with the user's head movement. We illustrate with two applications here. In LEGO, during the passive phase, which begins after the user receives the next instruction, a user typically turns away from the LEGO board and starts searching for the next brick to use in a parts box. During this period, the computer vision algorithm would detect no meaningful task states if the frames are transmitted. In PING PONG, 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 4.4(a) and (b) show an example trace from LEGO and PING PONG, respectively. Human-labeled ground truth indicating passive and active phases is shown in blue. The red dots indicate predictions of passive phase frames based on the IMU readings; these frames are suppressed at the client and not transmitted. Note that in both traces, the suppressed frames also form streaks. In other words, a number of frames in a row can be suppressed. As a result, the saving we gain from IMU

is orthogonal to that from adaptive sampling.

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

4.4 Evaluation

4.5 Discussion

Chapter 5

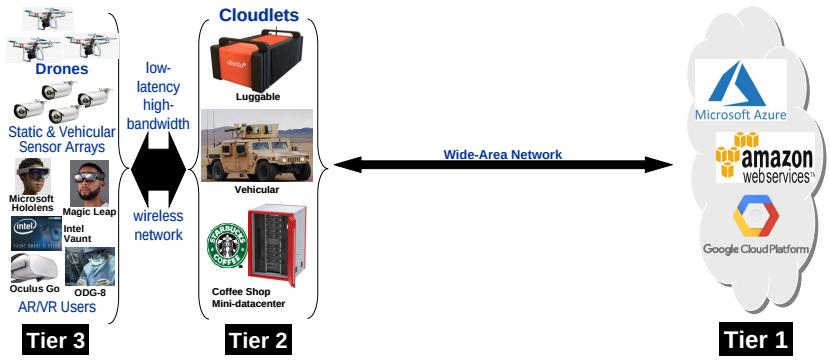
Cloudlet Resource Management for Graceful Degradation of Service

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 and its wireless network. 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 our focus.

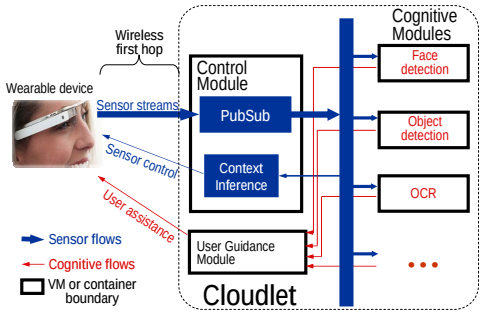
Our approach to scalability is driven by the following observation. Since compute resources and wireless capacity at the edge cannot be increased on demand, the only paths to scalability are (a) to reduce offered load, or (b) to reduce queueing delays through improved end-to-end scheduling. Otherwise, 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 and the wireless channel 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 5.1, scalability at the left is achieved very differently from scalability at the right. The relationship between Tier-3 and Tier-2 is *non-workload-conserving*, while that between Tier-1 and other tiers is workload-conserving.

With rare exceptions, reducing offered load is only possible with application assistance. Scalability at the edge is thus only achievable 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



From left to right, this tiered model represents a hierarchy of increasing physical size, compute power, energy usage, and elasticity. Tier-3 represents IoT and mobile devices; Tier-2 represents cloudlets; and Tier-1 represents the cloud. We use "Tier-2" and "cloudlet" interchangeably in the paper. We also use "Tier-3" to mean "mobile or IoT device."

Figure 5.1: Tiered Model of Computing



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

Figure 5.2: Gabriel Platform

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 [**?**].

Since the techniques for reducing offered workload are application-specific, we focus on a specific class of edge-native applications to validate our ideas. Our choice is a class of applications called *Wearable Cognitive Assistance* (WCA) applications [**?**]. They are perceived to be "killer apps" for edge computing because (a) they transmit large volumes of video data to the cloudlet; (b) they have stringent end-to-end latency requirements; and (c) they make substantial compute demands of the cloudlet, often requiring high-end GPUs.

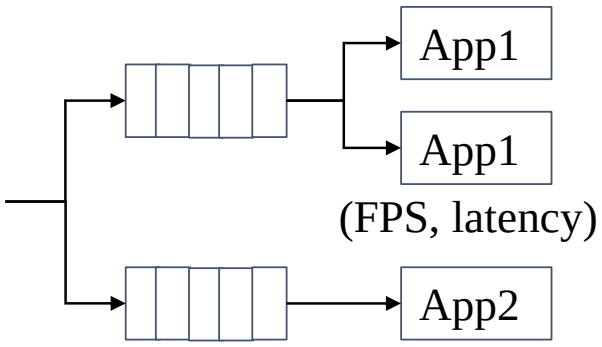
We leverage unique characteristics of WCA applications to reduce offered load through graceful degradation and improved resource allocation. Our contributions are as follows:

- An architectural framework for WCA that enables graceful degradation under heavy load.
- An adaptation taxonomy of WCA applications, and techniques for workload reduction.
- A cloudlet resource allocation scheme based on degradation heuristics and external policies.
- A prototype implementation of the above.
- Experimental results showing up to 40% reduction in offered load and graceful degradation in oversubscribed edge systems.

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 pro-



Only request flow is shown.

Figure 5.3: Resource Allocation System Model

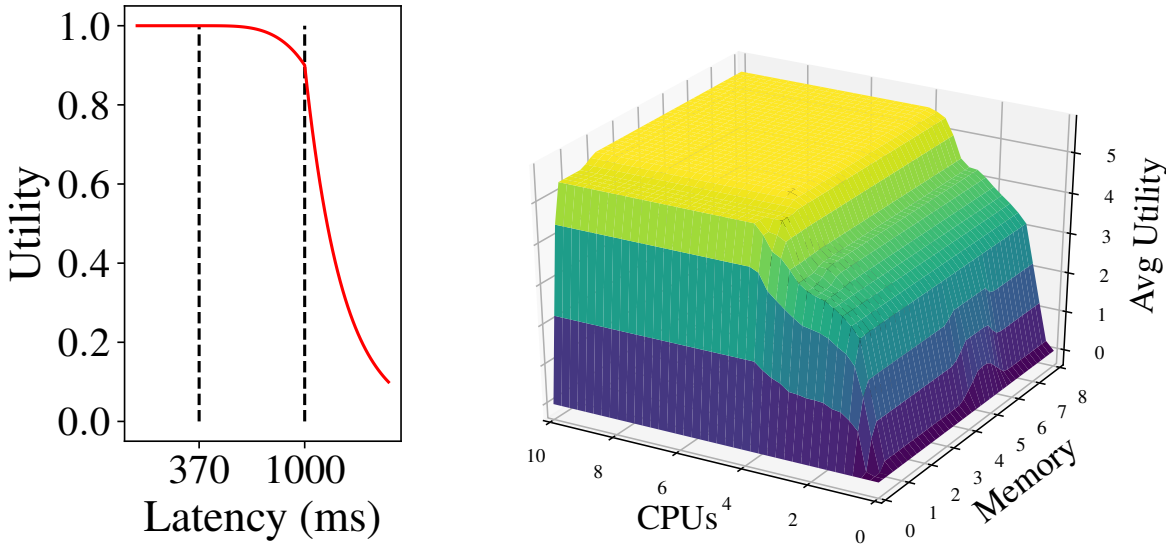
cessing delays, hence exceeding acceptable latency thresholds. On the other hand, when using short queues, accurate estimations of throughput, processing, and networking delay are vital to enable efficient use of cloudlet resources. However, sophisticated computer vision processing represents a highly 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 on this information, we construct offline application profiles that map multidimensional resource allocations to application QoS metrics. At runtime, we calculate a resource allocation plan to maximize a system-wide metric (e.g., total utility, fairness) specified by cloudlet owner. We choose to consider the allocation problem per application rather than per client in order to leverage statistical multiplexing among clients and multi-user optimizations (e.g., cache sharing) in an application.

5.1.1 System Model

Figure 5.3 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



(a) Utility For FACE

(b) Profile for FACE

Figure 5.4: FACE Application Utility and Profile

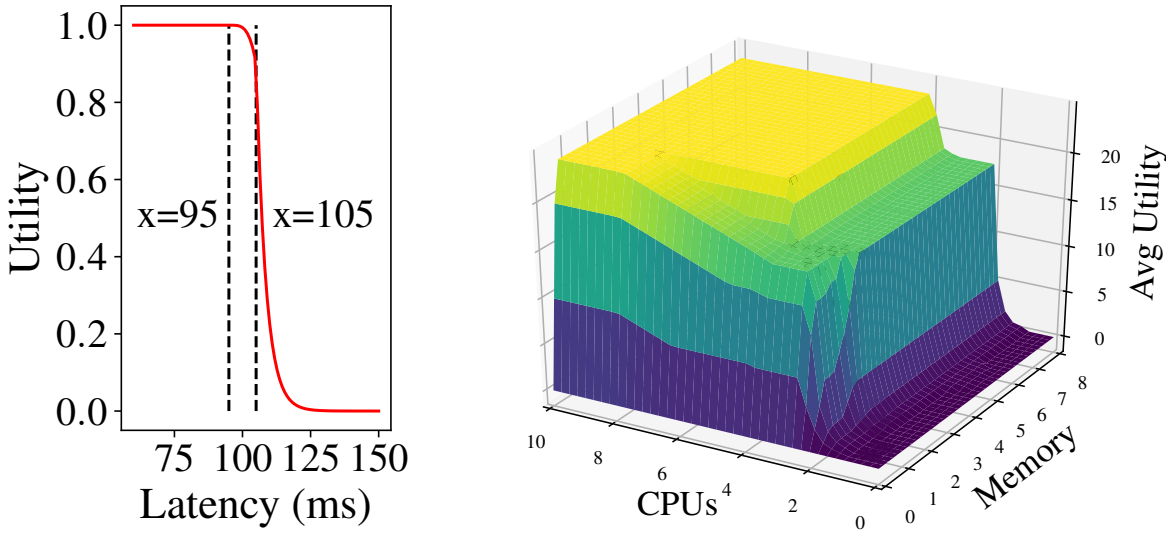
quality of response and its QoS characteristics (e.g., end-to-end latency). The total utility of a system is the sum of all individual utilities. A common limitation of a utility-based approach is the difficulty of creating these functions. One way to ease such burden is to position an application in the taxonomy described in Section 4.1.4 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.

5.1.2 Application Utility and Profiles

We build application profiles offline in order to estimate latency and throughput at runtime. First, we ask developers to provide a utility function that maps QoS metrics to application experience. Figure 5.4(a) and Figure 5.5(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 re-



(a) Utility For POOL

(b) Profile for POOL

Figure 5.5: POOL Application Utility and Profile

sources by utility are tractable. First, we assume utility values across different applications are comparable. Furthermore, we assume utility is received on a per-frame basis, with values that are normalized between 0 and 1. Each frame that is sent, accurately processed, and replied within its latency bound receives 1, so a client running at 30 FPS under ideal conditions can receive a maximum utility of 30 per second. This clearly ignores variable utility of processing particular frames (e.g., differences between active and passive phases), but simplifies construction of profiles and modeling for resource allocation; we leave the complexities of variable utility to future work. Figure 5.4(b) and Figure 5.5(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 $a \in \{\text{FACE, LEGO, PING PONG, POOL, } \ldots\}$, 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:

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

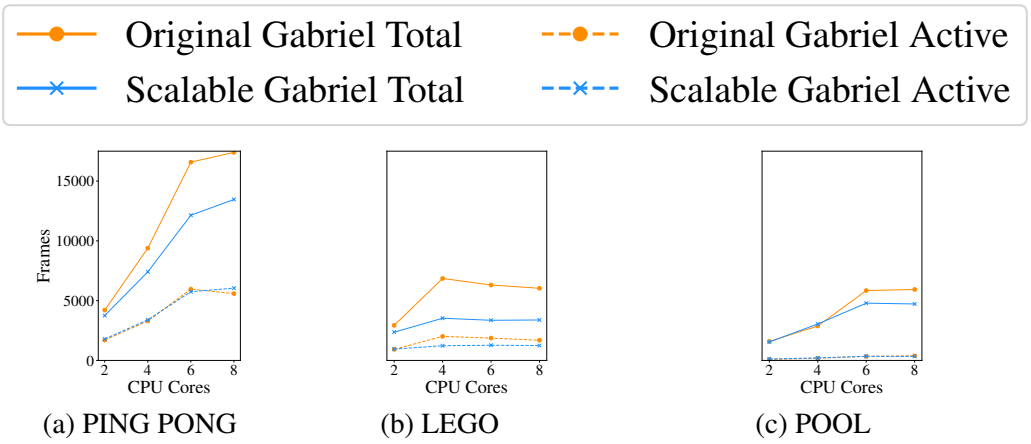
In above, c_a is the number of mobile clients requesting service from application a. The total resource vector of the cloudlet is $\hat{\mathbf{r}}$. For each application a, we determine how many instances to launch — k_a, and allocate resource vector $\mathbf{r}_a$ to each of them. A tunable knob γ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

Solving the above optimization problem is computationally difficult. We thus use an iterative greedy allocation algorithm as 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 5.6: 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

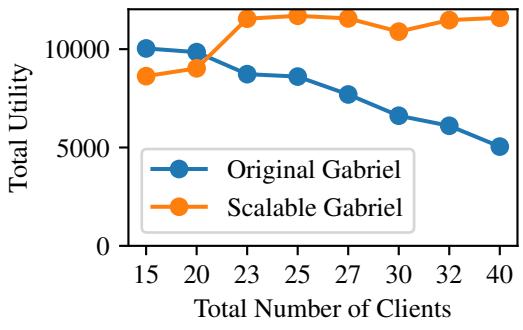


Figure 5.7: Total Utility with Increasing Contention

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 5.6 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 5.1. We use offline generated application profiles discussed in Section **??** to optimize for total system utility. Figure 5.7 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 5.8 and Figure 5.9 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 5.10 and Fig 5.11 show per client frame latency and FPS achieved. The first thing to notice is that concurrently utilizing

both sets of techniques does not cause conflicts. In fact, they appear to be complementary and latencies remain in better control than using resource allocation alone.

The previous plots consider per request latencies. The ultimate goal of our work is to maintain user experience as much as possible and degrade it gracefully when overloaded. For WCA applications, the key measure of user experience is guidance latency, the time between the occurrence of an event and the delivery of corresponding guidance. Figure 5.12 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 5.13. 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 explored [**?** **?** **?**]. 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

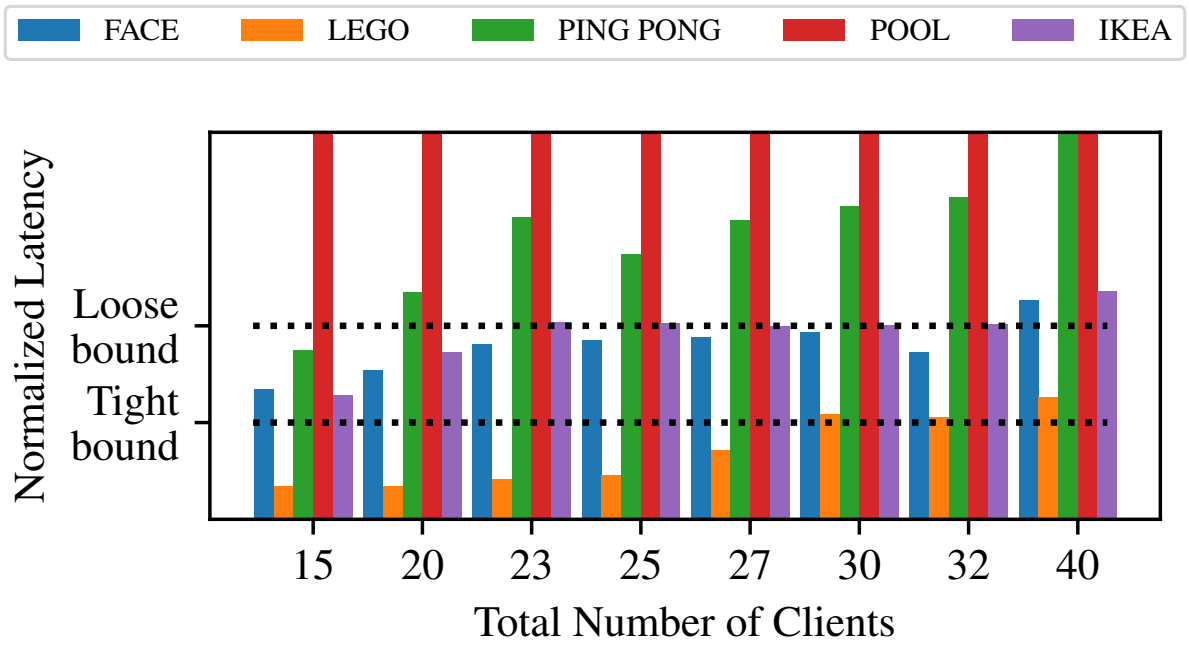
More than a decade ago, the emergence of cloud computing led to the realization that applications had to be written in a certain way to take full advantage of elasticity of the cloud. This led to the concept of "cloud-native applications" whose scale-out capabilities are well matched to the cloud, as 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.

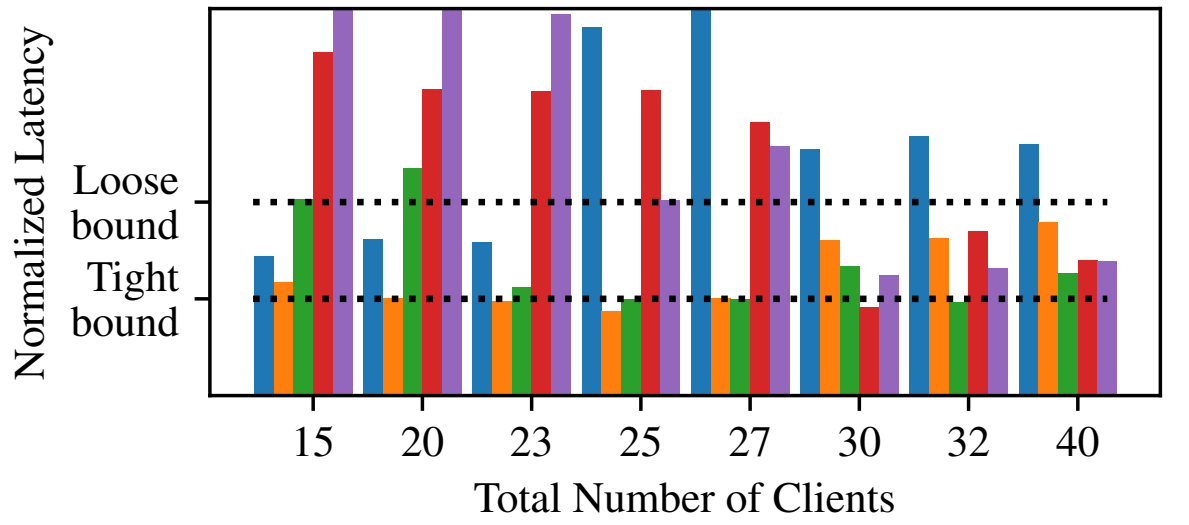
The is the first work to show that cloud-native implementation strategies that focus primarily on dynamic scale-out are unlikely to be effective for scalability in edge computing. Instead, edge-native applications need to adapt their network and cloudlet resource demand to system load. As the total number 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.



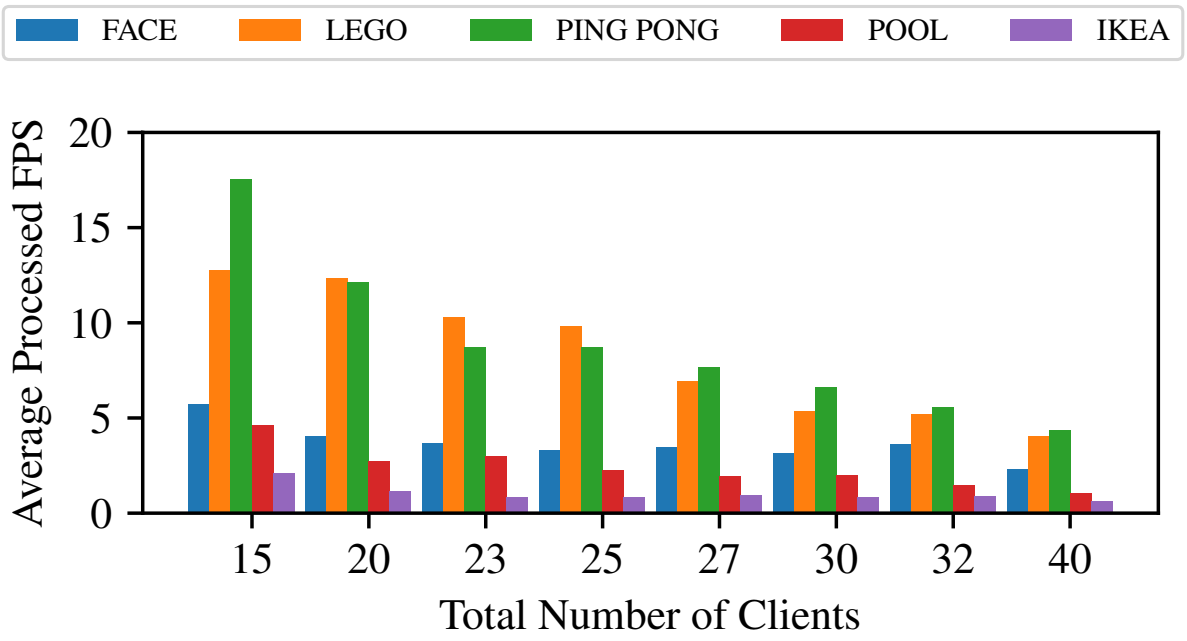
(a) Original Gabriel



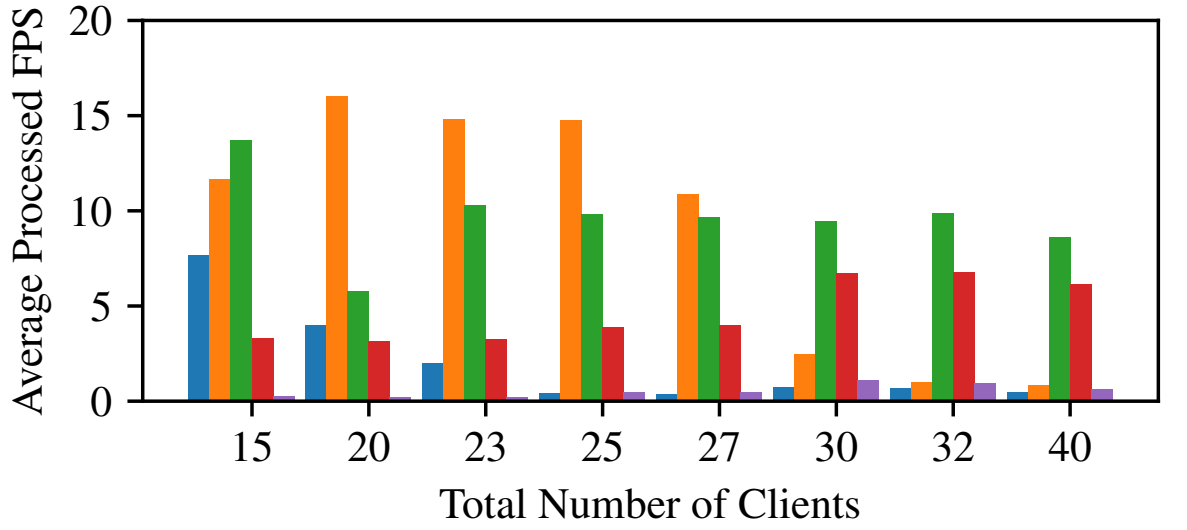
(b) Scalable Gabriel

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

Figure 5.8: Normalized 90%-tile Response Latency

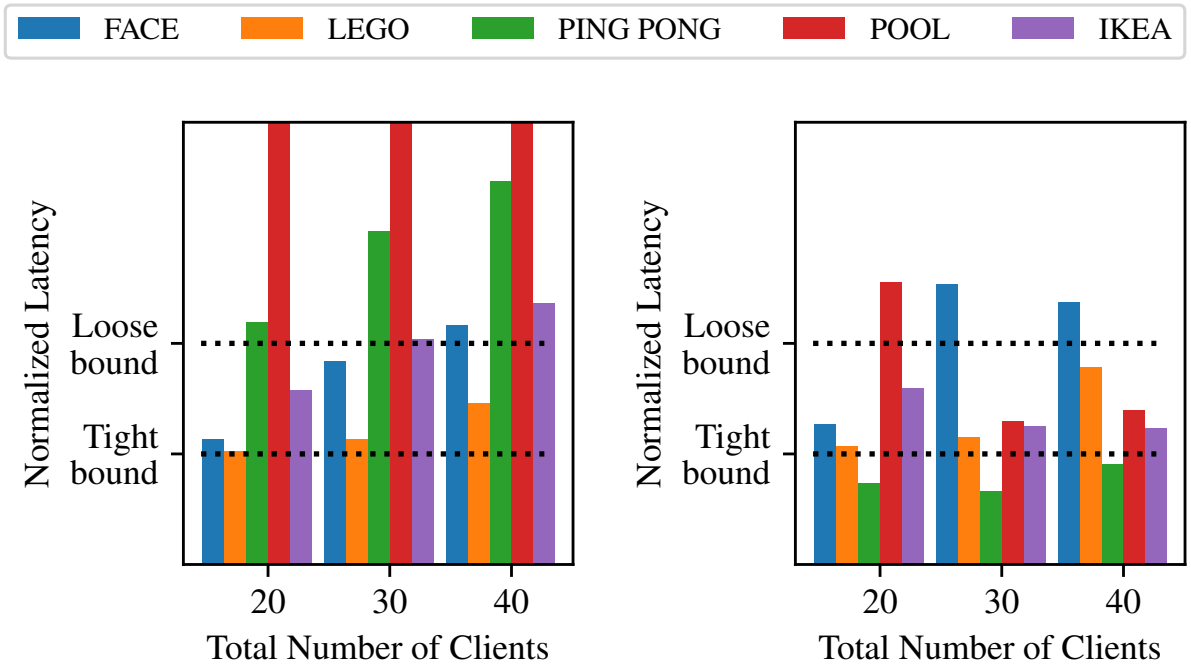


(a) Original Gabriel



(b) Scalable Gabriel

Figure 5.9: Average Processed Frames Per Second Per Client

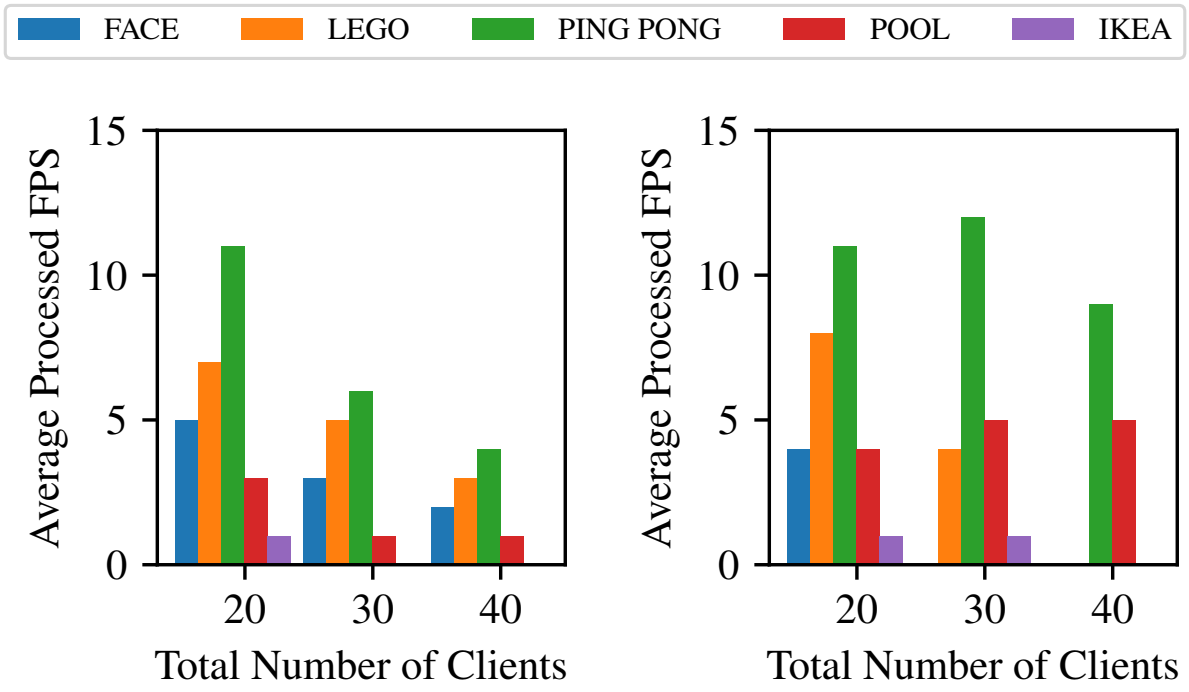


(a) Original Gabriel

(b) Scalable Gabriel

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

Figure 5.10: Normalized 90%-tile Response Latency



(a) Original Gabriel

(b) Scalable Gabriel

Figure 5.11: Processed Frames Per Second Per Application

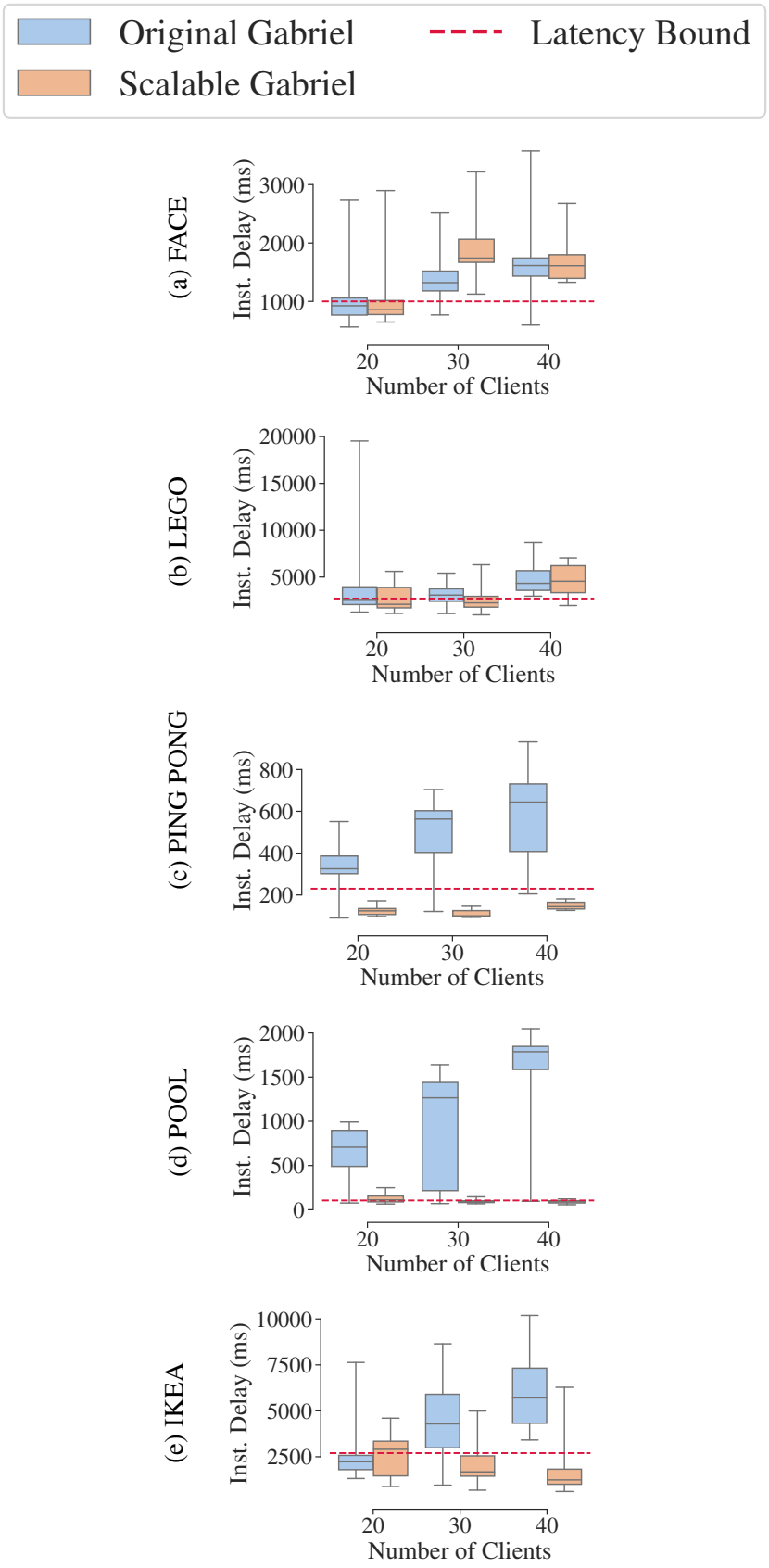


Figure 5.12: Guidance Latency

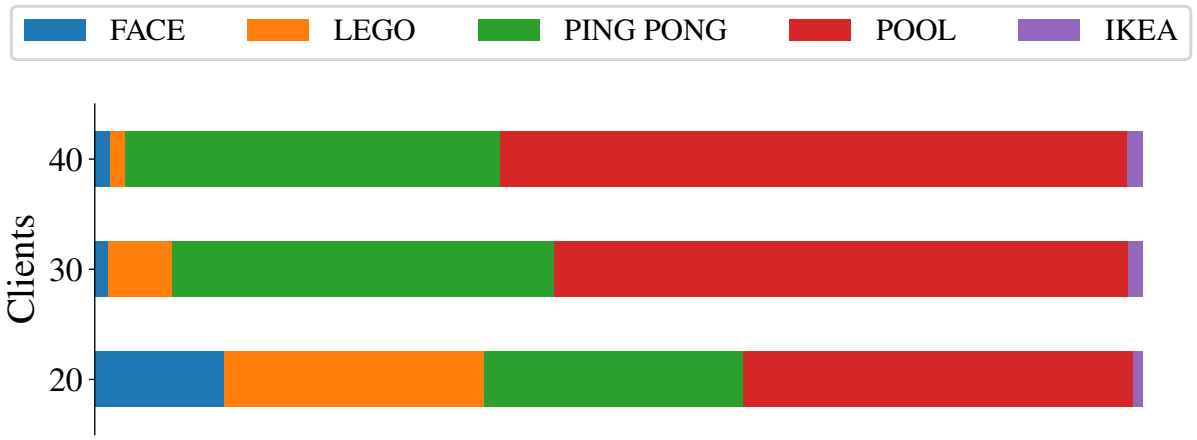


Figure 5.13: Fraction of Cloudlet Processing Allocated

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 Cloudlet Gateway

7.2 Enabling GPU Usage for Cloudlets

Many edge workloads require a GPU to meet latency requirements. However, sharing and virtualizing GPU devices for computation remain a challenge. In below, I will describe our group's setup to allow multi-tenancy on GPUs.

7.2.1 Architecture

The figure below represents the architecture. We adopt a containers on top of virtual machines approach to allow GPU sharing. The GPU device is dedicated to a particular VM through GPU-passthrough on the host.

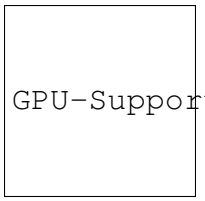


Figure 7.1: Architecture

7.2.2 Setup

GPU-Passthrough to a libvirt VM

The setup process is automated through Ansible. Please see repo. Use following command to set up GPU passthrough.

[] ansible-playbook -i hosts-gpu-passthrough gpu-passthrough-playbook.yml

Container Access to GPU

nvidia-docker enables containers to access GPU easily. See repo for installation.

7.2.3 Performance Overhead

In all of our benchmark measurements, the overheads introduced by virtualization are between 0% and 2.3%.

We used two benchmarks to measure the overhead introduced by VM and container virtuation. The first benchmark DeepBench is compute-intensive. In particular, we focused on convolution operation — the core workload of convolutional neural networks. The second benchmark BandwidthTest evaluates the data transfer bandwidth between the host and the GPU.

HW & SW Setup

The GPU in test is NVIDIA Tesla GTX 1080 Ti GPU.

- Max GPU Clock: 1911 MHz(Graphics), 5505 MHz(Memory)
- Default Computing mode

The software in bare-metal, VM, and container-inside-VM is kept the same.

- Ubuntu 16.04
- linux kernel 4.4.0-130
- 396.37 NVIDIA driver + cuda 9.0 + cudnn 7.1.4.18

The VM is created using qemu-kvm 2.6.2 and libvirt. GPU passthrough is achieved through vfio. The container-inside-VM is created using nvidia-docker 2.0.3 and docker 18.03.1.

Convolution Kernels — Compute

We used floating point general matrix multiplication from this benchmark. The benchmark is invoked with

[] ./conv$_b enchinferenctfloat$

The benchmark uses *cudnnFindConvolutionForwardAlgorithm* in cudnn to determine the convolution algorithm to use at runtime. In our experiments, such dynamic algorithm selection results in large variance of execution time as different algorithms are used across different runs. It is unclear why cudnn would select different algorithms even when convolution parameters are kept the same. To obtain reproducible results, we manuallly fixed the convolutional algorithm to be CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM in the code. See here for more on what the algorithm does.

We used the same convolutional kernels as described in DeepBench "Server Inference Setup" for comparison.

Convolution Experiment Parameters [c]@llllll@

Experiment	Input Size	Filter Size	# of Filters	Padding (h, w)	Stride (h, w)
1	W = 341, H = 79, C = 32, N = 4	R = 5, S = 10	32	0,0	2,22
	W = 224, H = 224, C = 3, N = 1	R = 7, S = 7	64	3, 3	2, 23
	W = 56, H = 56, C = 256, N = 1	R = 1, S = 1	128	0, 0	2, 24
	W = 7, H = 7, C = 512, N = 2	R = 1, S = 1	2048	0, 0	1, 1

Convolution Speed [c]@ccccc@

Virtualization	Exp 1 (us)	Exp 2 (us)	Exp 3 (us)	Exp 4 (us)
bare-metal	381 +- 9	44 +- 1	39 +- 1	68 +- 1
VM	384 +- 11	45 +- 1	39 +- 1	67 +- 1
Container inside VM	386 +- 9	45 +- 1	39 +- 1	68 +- 2

Bandwidth Test — Bandwidth

We benchmarked memory bandwidth between the host and the GPU device using bandwidthtest.cu. You can learn more about pinned transfer bandwdith here.

Pinned Transfer Bandwidth [c]@ccc@

Virtualization	Host To Device Bandwidth (GB/s)	Device to Host Bandwidth (GB/s)
bare-metal	6.17 +- 0.00	6.67 +- 0.01
VM	6.13 +- 0.00	6.66 +- 0.00
Container inside VM	6.58 +- 0.02	6.12 +- 0.0

Experiments Data

Complete experiment results are in `data` directory.

- baremetal-conv, vm-conv, container-conv: Convolution kernel benchmark results for bare-metal, vm, and container-inside-VM.
- run.sh: Convolution kernel result summary script
- bandwidth-test: BandwithTest benchmark results for bare-metal, vm, and container-inside-VM.
- conv-dynamic-algorithm: Unmodified convolution kernel benchmark results, which select convolution algorithms at runtime.
- gemm-test: GEMM kernel benchmark results using DeepBench. Note that the data has high variance since not enough runs are executed.
- vm-ssd-*: Object detection benchmark results on VM using cvutils. Note that this benchmark includes extra processing time on CPU as well. It should not be used for measuring virtualization overhead.

7.3 Gabriel Deployment

7.4 Discussion

Chapter 8

Conclusion and Future Work