An Empirical Study of Latency in an Emerging Class of Edge Computing Applications for Wearable Cognitive Assistance

Carnegie Mellon University and Intel Labs
Presented by Andrew Neth

Introduction

- Why edge computing?
  - Low latency
  - Portability tends to cost performance
- Resource-hungry applications that won’t run on mobile devices anytime soon
- Wearable devices that interface closely with the user
- Wearable cognitive assistance applications
  - Wearable devices stream sensor data to cloudlets
  - Cloudlet performs intensive data processing and sends guidance messages to wearable
  - Wearable presents feedback to user with (ideally) very low latency
  - “Bring[s] AI technologies within the inner loop of human cognition and interaction”

Goals

Latency is critical in these applications

- Where does the time go?
- Is (single-hop) LTE a viable substitute for Wi-Fi?
- How much cloudlet power do these applications need?
- Does specialized hardware (e.g. a GPU) help?
- What are the lower and upper bounds on acceptable latency?
- Is latency a problem that more processing power can solve?

Applications

“Pool”

Helps a novice pool player adjust aim.

Uses a “fractional aiming system” to calculate correct angle, giving continuous feedback as to how the user should adjust their angle.

Symbolic representation: positions of pocket, object ball, cue ball, and cue top/bottom

Guidance format: Continuous graphical display of left/right arrow or thumbs up

“Ping-pong”

Tells a novice player whether they should hit the ball to the left or the right based on which it thinks is more likely to beat the opponent.

Finds objects in the scene (ball, table, opponent) using optical-flow based motion detection

Symbolic representation: in-rally status, ball position, opponent position

Guidance format: Spoken message of “Left!” or “Right!”

“Workout”

~~Tells a novice player whether they should hit the ball to the left or the right based on which it thinks is more likely to beat the opponent.~~

~~Finds objects in the scene (ball, table, opponent) using optical-flow based motion detection~~

Symbolic representation: action, rep count

Guidance format: spoken rep count

“Face”

Recognizes a familiar face using a deep residual network, reminding the user of their name.

Can be used in conjunction with the Expression conversation aid application

Symbolic representation: name as ASCII

Guidance format: spoken name of recognized person

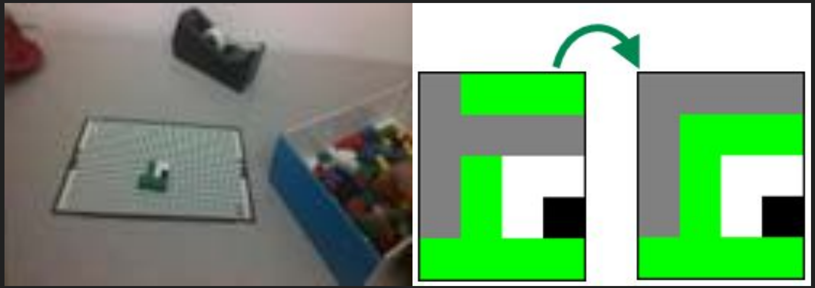
“Lego”

Guides a user through the process of assembling a (2D) Lego model.

Identifies the board using an easily-tracked pattern

Uses edge/color detection to locate bricks on the board

Uses weighted majority voting to determine the brick color within each block



Symbolic representation: matrix of brick color values within the grid of the board

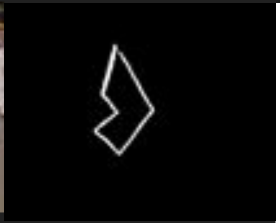
Guidance format: instructions— “put a 1x3 green piece on top”

“Draw”

Helps a user sketch an existing image

Built on an existing third-party app that displays similar feedback on a screen and for digital drawings

Now works with any (visible) drawing medium and displays the error alignment using Google Glass



Symbolic representation: matrix of brick color values within the grid of the board

Guidance format: instructions— “put a 1x3 green piece on top”

“Sandwich”

~~Helps a user sketch an existing image~~

~~Built on an existing third-party app that displays similar feedback on a screen and for digital drawings~~

~~Now works with any (visible) drawing medium and displays the error alignment using Google Glass~~

Symbolic representation: object structure— “lettuce on top of ham and bread”

Guidance format: instructions— “put a piece of bread on the lettuce”

Applications

Differences

- Different ML algorithms employed
- Different forms of guidance
- Different latency requirements
- Different levels of interactivity

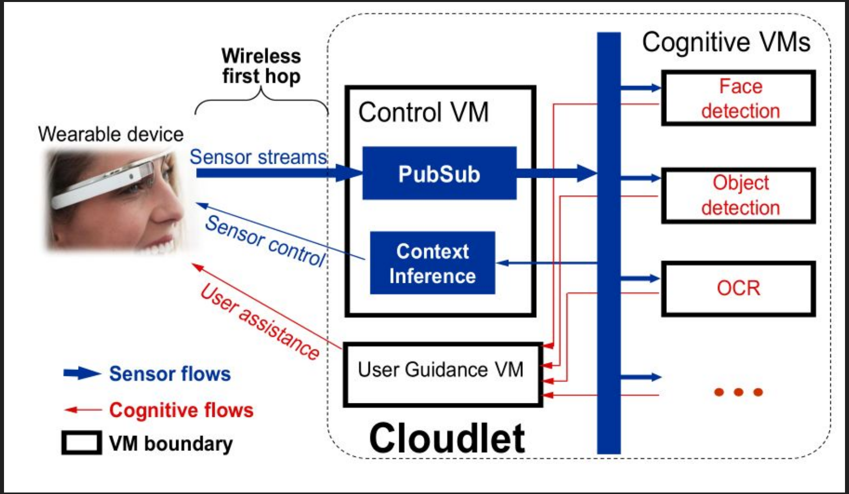
Similarities

- All based on visual input
- All first-person (except “Workout”)
- Shared logical structure
  - Sensory input analyzed to get symbolic representation
  - Symbolic representation further analyzed to generate guidance
  - Guidance sent to wearable and presented to user

The Gabriel Platform

The structural similarities across the apps allow them to all be implemented on top of the Gabriel offloading framework.

- Control VM receives sensor stream from device; shares with "cognitive engines"
- Cognitive VMs process sensor data to generate symbolic representation
- User guidance VM performs "phase 2" processing to generate guidance



Gabriel Architecture

Sensor stream is 640x360@≤15fps video

Guidance messages are JSON, usually <1kB

Specific Questions

- How much does edge computing affect end-to-end latencies of these applications?
- How does edge computing based on cellular/LTE compare with that based on WiFi?
- Does the choice of end-user device affect performance?
- How much can hardware accelerators and extra CPU cores in the back-end help?
- Short of devising revolutionary new algorithms, what can we do to improve cloud/cloudlet processing time?
- How hard do we need to work at reducing latency?

Testing

- Two backends: i7-3770 cloudlet, Amazon EC2 3.2xlarge cloud
- Five frontends: Nexus 6 (represents high-end wearable), Google Glass, Microsoft HoloLens, Vuzix M100, ODG R7
- Devices use 5 GHz 802.11n WiFi where possible and the best available option otherwise
- For consistency, devices send pre-recorded frames (but have the camera on) and are cooled with ice packs

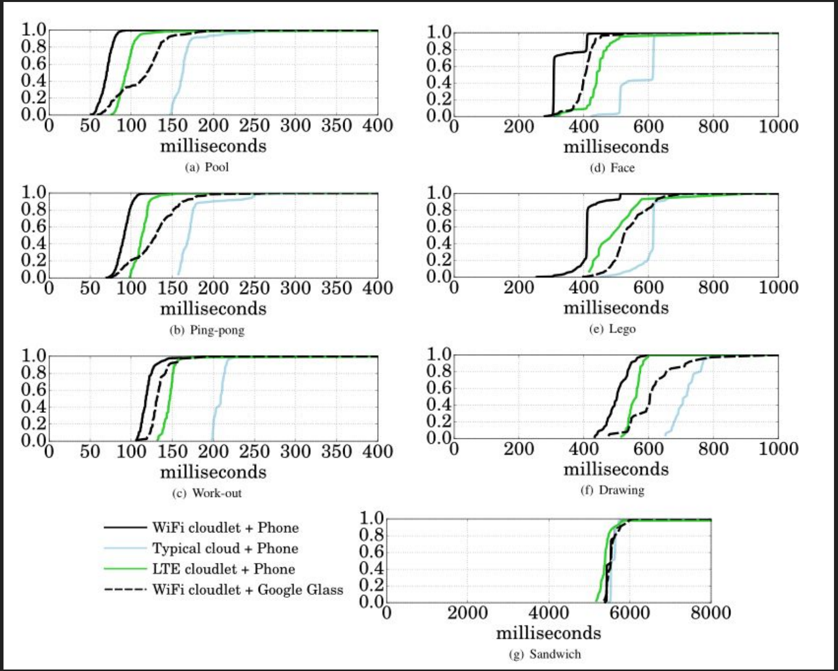
Data

The “WiFi cloudlet + Phone” configuration is generally the fastest out of those tested

Variation in CDF line shape is mostly due to variation in frame compression time

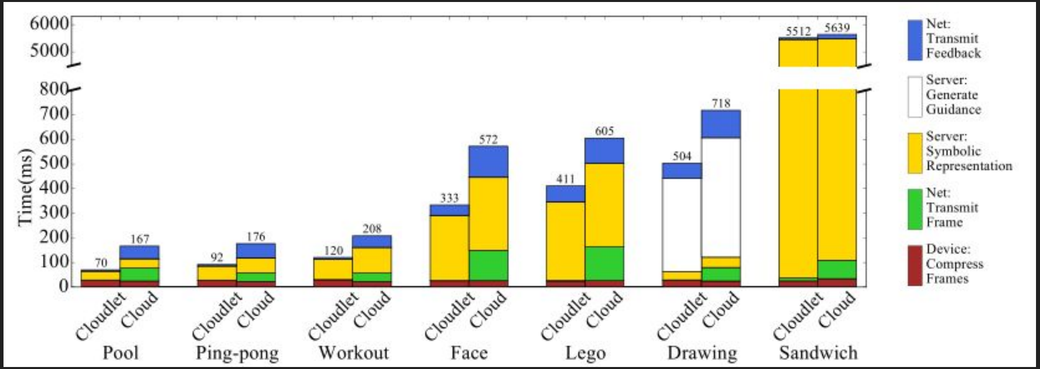
Massive variations between applications: “Sandwich” almost 100x slower than “Pool”

Using AWS-West over a local cloudlet adds significant (for most applications) latency: >2x for “Pool”, but negligible for “Sandwich”



the cloud is slow

Latency Breakdown



Primary conclusion: Cloudlets show significant reduction of transmission times and overall latency advantage

Compute time is almost unchanged between cloudlet & cloud

“Drawing” is the only case where phase 2 computation is non-trivial (it uses complex third-party software to generate guidance)

The high latency of “Sandwich” is basically all from phase 1 computation (it uses an expensive deep neural network)

Latency Breakdown: WiFi vs. 4G LTE

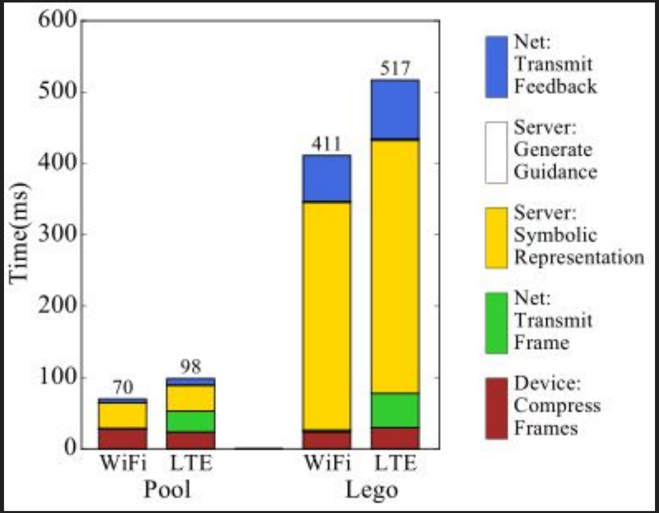
What if we had an LTE-connected cloudlet instead of using WiFi?

With help from Vodafone Research, set up a low-power local LTE network for testing

Conclusion: 4G LTE is worse than WiFi (expected), but still not too bad

5G also holds promise[2017] for further improvements

As before, the significance of the added latency depends on how much latency the application already has



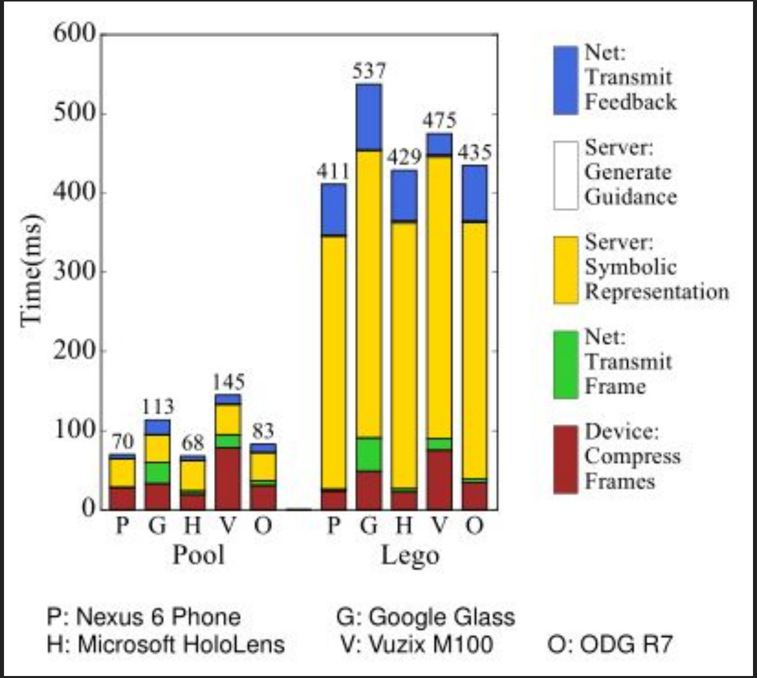
Latency Breakdown: Frontend Hardware

Even though we're offloading, wearable device performance still matters

Some compress more slowly than others

Google Glass, with 2.4 GHz 802.11b/g, transmits more slowly, and the Vuzix device is also somewhat slow there

Yet again, difference is more meaningful for "Pool" than for "Lego"

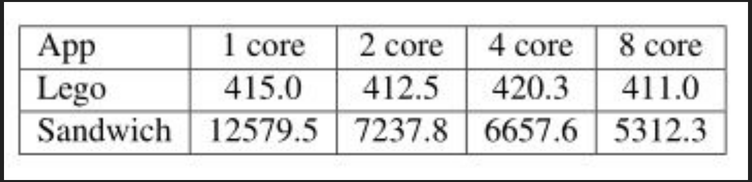


Can we make the yellow bar smaller?

What if we used more cores?

Multithreading, our old friend

Sometimes helpful, sometimes not



App	1 core	2 core	4 core	8 core
Lego	415.0	412.5	420.3	411.0
Sandwich	12579.5	7237.8	6657.6	5312.3

Ability to leverage CPU parallelism depends on algorithm: “Sandwich” sees reasonable benefits (albeit with diminishing returns), but “Lego” does not

Even with eight cores, “Sandwich” latency remains above any acceptable maximum

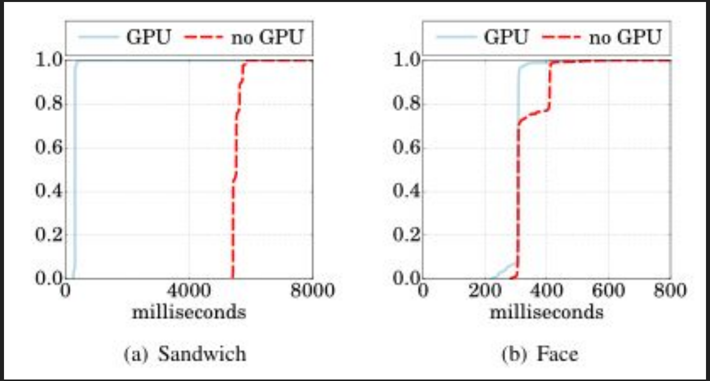
What if we used more coresmore coresmore coresmore cores more coresmore coresmore coresmore cores more coresmore coresmore coresmore cores more coresmore coresmore coresmore cores?

Some algorithms benefit greatly from GPU acceleration

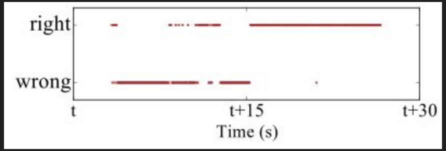
“Sandwich” and “Face” both use GPU-optimized NN libraries

“Sandwich” sees massive improvements, but “Face” barely changes

Benefits are application-dependent, so figure out if it makes sense for your application before making an investment



good algorithsm are slow [sic]



The computer vision community is largely focused on accuracy, performance be damned

Former state-of-the-art algorithms may be less accurate but also less demanding

Algorithm inaccuracies aren’t noise: if an algorithm’s results are good, they’re likely to keep being good under consistent conditions

Devise a “Black-box multi-algorithm approach that compares faster algorithms to more accurate ones

The Algorithm Competition

Key insight: algorithm accuracy exhibits temporal locality

Algorithm mistakes depend on input characteristics, not random events

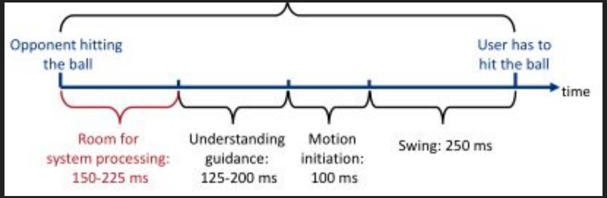
Run a fast but less-reliable algorithm and a slow but robust algorithm concurrently & compare results

If the fast algorithm agrees with the robust one for long enough, start to trust it and use its output. Until then, wait for known-good results

If fast algorithm's output deviates, temporarily stop trusting it and revert to just checking its trustworthiness

Works very well: Improves "Face", "Lego", and "Sandwich" by about ⅔ each

Acceptable Latency


Opponent hitting the ball
User has to hit the ball
time
Room for system processing: 150-225 ms
Understanding guidance: 125-200 ms
Motion initiation: 100 ms
Swing: 250 ms

Human perception is very fuzzy, but we can still establish approximate lower/upper bounds

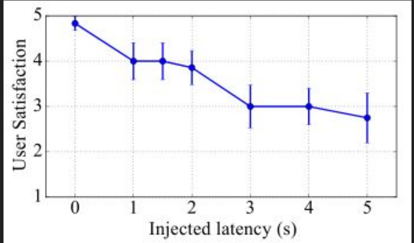
“Tight bound”: don’t bother trying to do any better than this

“Loose bound”: any slower than this threshold is perceptibly slow

For continuous feedback like “Pool”, response times of 100±5ms are perceived as instantaneous, so that’s the loose bound

For the guidance model of “Ping-pong”, loose bound is somewhere around 225ms

Acceptable Instructional Latency?



User satisfaction: 5=satisfied, 1=faster response desired

What are the bounds for tasks like “Lego” and “Sandwich”?

There’s not much existing research about this— past almost-relevant research has mostly been about devices responding to user input, not prompting of user action once the user is ready for a prompt

“Wizard of Oz” experiment with “Lego” application: human expert recognizes step completion, but artificial latency is injected before guidance is presented

2.7 second experimental bound

bottom text