Collaborative Acceleration for Mixed Reality

Kiron Lebeck
University of Washington*
kklebeck@cs.washington.edu

Eduardo Cuervo
Microsoft Research
cuervo@microsoft.com

Matthai Philipose
Microsoft Research
matthaip@microsoft.com

ABSTRACT

A new generation of augmented reality (AR) devices, such as the Microsoft HoloLens, promises a user experience known as *mixed reality* (MR) that is more seamless, immersive, and intelligent than earlier AR technologies. However, this new experience comes with high computational costs, including exceptionally low latency and high quality requirements. While this cost could be offset in part through offloading, we also observe an increasing availability of on-device, task-specific accelerators. In this paper, we propose *collaborative acceleration*, a collaborative technique that utilizes the unique hardware accelerated capabilities of an MR device, in conjunction with an edge node, to partition an application's core workflow according to the specific strengths of each device. To better understand the workloads of next generation MR applications, we implement a concrete MR app on the HoloLens: an assistive tool to visually aid users in manipulating physical objects. Through our prototype, we find that offloading a subset of the app's workload to an edge while also leveraging the strengths of the HoloLens delivers accurate enough results at a low latency. Our work provides an early glimpse into the system design challenges of MR, potentially the first "killer application" of edge offloading.

1 INTRODUCTION

A new generation of augmented reality (AR) devices, such as the Microsoft HoloLens [11], MagicLeap [10], and Meta2, promises a user experience known as *mixed reality* (MR) that is more seamless, immersive, and intelligent than that provided by earlier AR technologies such as the Google Glass. Where past AR experiences overlaid text or 2D annotations over a small portion of the user's field of view, MR provides overlays with the illusion of physical presence by generating a semantic understanding (e.g., the identities of objects and people) and detailed 3D geometric maps of the user's environment, along with finely rendered graphics anchored within this 3D environment. These immersive capabilities enable new applications [1] such as home planning, pain and phobia treatment, and *3D task-assistance*, or assistive tools to visually guide users through the manipulation of physical objects. However, these advancements come at high computational cost: the "maps, meshes and models" that comprise this workload are computationally expensive. In this paper, we seek to better understand the workload and hardware ecosystem surrounding MR technologies, and to cast some early light on their implications for MR systems design.

In order to better understand the workload, we implement a concrete application[1], a 3D task-assistance app to assist users with cleaning rooms, on the HoloLens. We identify five tasks as fundamental to such applications: specify the target configuration of a room, detect the objects in this space and estimate their 3D positions, recognize their orientations, track these objects as they move, and render overlays to indicate how these objects should be manipulated.

Architecturally, we believe the HoloLens is representative of MR devices that will be available in coming years. In particular, the battery constraints introduced by mobility mean that on-board computation will include a mix of custom acceleration and low-power processing. For example, the HoloLens provides hardware acceleration for estimating the geometry of the space surrounding it, along with a modest integrated GPU for rendering. Further, recent announcements indicate that the device will soon see an accelerator for Deep Neural Networks added to it, as for instance, in the Apple iPhone X. Finally, the device provides fast network connectivity that allows it to take advantage of resources in the cloud or nearby "edge" [18] devices, nodes with substantial amounts of computing resources placed in close proximity to mobile devices to augment their abilities.

Offloading to the cloud and edge has been a recurring theme in recent years when seeking to balance mobile constraints with application needs [2, 5, 8]. MR apps motivate yet another revisiting of this tradeoff for two reasons. First, they have exceptionally low latency and high quality requirements, rendering the latency to the cloud prohibitive (around 74ms [9]) and potentially straining both the latency limits of off-boarding even to nearby edge nodes, as well as the computational limits of on-board execution. Second, on-device task-specific accelerators have the potential to dramatically increase local computational capabilities, allowing the device to collaboratively share execution duties with the edge [4] rather than merely playing the role of "offload shaping" [6, 7]. Accelerators also raise the possibility that offloading to the edge may not be needed at all in the future.

Nevertheless, we posit that the edge still has a strong, albeit altered role in the future of mixed reality. Although we

*This work was conducted while the first author was an intern at Microsoft.

[1] A video of our prototype can be found at https://youtu.be/bWTNzg6y5hY

believe that on-device localization and mapping will be feasible going forward, both AI workloads (e.g., executing Deep Neural Networks) and high quality graphics will benefit from edge assistance. In particular, even if certain AI primitives (e.g., hand pose estimation or object detection) run on board, each application could have its own custom AI requirements that may not all be accommodated locally. Similarly, although high fidelity graphics from a sufficiently powerful on-board GPU would allow edge-free operation, truly immersive or seamless graphics will likely require edge-class capabilities in the foreseeable future.

In this paper, we propose collaborative acceleration, a technique that utilizes the unique custom hardware accelerated capabilities of a mixed reality device in collaboration with an edge device to partition an application's core workflow according to the specific strengths of each device. Through a prototype application on the HoloLens, we identify five key primitives for collaborative acceleration that leverage the strengths of the HoloLens's on-device capabilities. Our work provides an early glimpse into system design for mixed reality. Although the MR setting requires renegotiating the systems contract between device and edge significantly, the tangible benefits from edge acceleration may also make it the first "killer application" of edge offloading.

2 BACKGROUND

We begin with important background context on the HoloLens and the increasing prevalence of hardware accelerators before diving into the details of collaborative acceleration.

2.1 HoloLens

The HoloLens is an untethered head-mounted display that presents visual overlays on see-through LCOS waveguide displays. The device can track its pose within the user's environment, without external hardware, using computer vision algorithms, four grayscale cameras, and an inertial measurement unit (IMU). It also employs a depth sensor to perform spatial mapping, or the creation of a 3D mesh representing the user's surroundings, and it provides developers with access to a general purpose 2.4 megapixel RGB camera and a four-microphone array. The headset features an Intel Cherry Trail CPU/GPU SoC [19], 8 MB of RAM, and a custom accelerator chip that we discuss further below.

2.2 Hardware Accelerators

Devices like the HoloLens must execute computationally intensive tasks while remaining lightweight, responsive, and power efficient. This tension has driven the development of custom hardware accelerators; for example, the HoloLens includes a set of 28 custom 24nm Tensilica DSP cores, called the Holographic Processing Unit (HPU) [19]. The HPU speeds up operations such as low-latency rendering and spatial mapping [12], and future versions will also accelerate neural networks [13]. Other organizations have begun employing hardware accelerators for MR as well. For example, Apple's iPhoneX includes a neural network accelerator they call the Neural Engine, and Google's Tango phone platform runs atop devices with Qualcomm-built hardware support for 3D depth perception, analogous to HoloLens spatial mapping. While the latency requirements for traditional phone-based MR may not be as strict as they are on head-mounted displays, phones can be easily used as MR headsets by employing adapters like mergecube. Other major players such as Huawei [15] and Intel (Movidius) also have neural network hardware accelerators of their own, suggesting an industry-wide trend towards hardware support for MR.

3 DESIGN AND IMPLEMENTATION

We next explore the potential benefit of applying collaborative acceleration to MR applications, beginning with a motivating scenario to guide our inquiry.

Motivating Scenario: 3D Task Assistance. To more concretely explore the potential role of collaborative acceleration in MR, we focus on the domain of *3D task assistance*, activities in which the user may benefit from precise visual guides to help them complete a task (e.g., assembling furniture or cooking). Specifically, we focus on meticulous room cleaning, commonly performed in hotels or cruise ships. Looking to deliver the best experiences to their guests, interior designers carefully plan the layout and placement of items within guests' rooms. Before new guests arrive, cleaning staff must ensure that the rooms are appropriately configured according to the designer's specifications; for example, by wiping, moping, or dusting surfaces, or by correctly positioning items of interest in specific locations. An AR cleaning assistance app could help cleaning staff complete these objectives by providing helpful visual cues. In our model, a room designer specifies a correct room configuration with visual markers a priori, demarcating surfaces that should be cleaned and target locations for individual items to be placed. The application then retrieves this configuration for cleaning staff, providing visual guidelines to identify out-of-place objects and their target locations, as well as uncleaned surfaces.

Prototype Overview. Our prototype, shown in Figure 1, addresses the object placement portion of the room cleaning scenario described above. The app consists of a *designer* mode and a *cleaner* mode. In designer mode, the user specifies target locations for objects of interest with a 3D cursor and voice commands. First, the user focuses their gaze and says 'target' followed by the name of an object (e.g., 'target mouse'), creating a blue-tinted overlay of the object at the

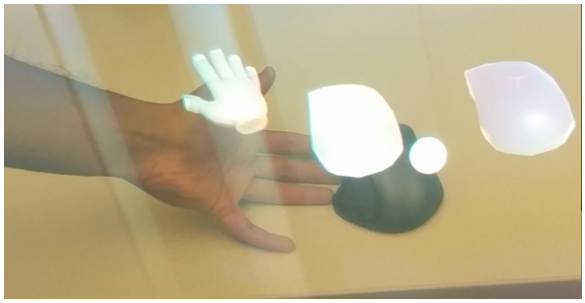


Figure 1: Our prototype. The user is moving a mouse to the right towards a target.

cursor's location. The user repeats this procedure for every desired object. In cleaner mode, the app displays red-tinted overlays on top of the relevant real-world objects, in addition to the blue target overlays. As the user moves a physical object, the corresponding red overlay follows it. Once the object reaches its correct target location, the blue overlay disappears and the red overlay turns green.

Through our prototype development, we identify 5 operations critical to the core workflow, and that we believe generalize more broadly to 3D task assistance problems. While we have only implemented 3 so far, we find that all 5 benefit from collaborative acceleration by taking advantage of the HoloLens's specialized hardware, as we describe below.

3.1 Define Target Locations

To support designer mode as described above, the application must be able to place targets within the user's 3D environment in response to user requests. We first observe the following: target placement is a latency-tolerant operation. Thus, when the user executes a voice command (e.g., "target scissors"), the application can offload the audio to more powerful cloud or edge servers to perform speech recognition[2]. Once the command is processed and returned to the HoloLens, the next challenge is determining where to apply the result of the operation, i.e., where to place the target. Fortunately, the HoloLens comes equipped with hardware support for efficiently generating a 3D geometric understanding of the user's environment, referred to as *spatial mapping* [12]. We determine the position of the user's gaze by tracing a ray orthogonal to the device's camera perspective, taking the first 3D point of intersection between that ray and the spatial map and placing the target overlay at this location.

3.2 Obtain 3D Positions of Objects

Once targets have been placed, the application must determine the positions of relevant real-world objects in 3D space. In our case, this includes the objects that the designer would like placed in specific locations. This operation can be divided into three sub-operations: (1) detecting and determining the 2D positions of all objects currently in the user's view, (2) obtaining a 3D position for each object, and (3) compensating for motion of the headset since the detection operation was performed. We implement (1) and (2).

1. 2D Object Detection. To identify the objects in the user's view, we obtain video frames using the HoloLens camera capture API and send them over Wi-Fi to our edge node to be processed. Our edge node uses YOLO [14][3], a state-of-the-art object detection framework, to process each frame and return information about detected objects to the HoloLens, including 2D bounding boxes for each object.

2. 2D-to-3D Position Mapping. We again leverage the HoloLens's spatial mapping capabilities, this time to determine the depth of detected objects, using a heuristic similar to our target placement method. We trace a ray orthogonal to the device's camera emanating from the 2D location of the detected object within the device's camera feed and take the intersection of this ray with the spatial map as our object's 3D position. Due to noisy results from YOLO, we take a sliding window of detection events across consecutive frames, determining that an object is present only when a sufficient proportion of the past N frames contain detections that map to a cluster of nearby 3D positions. As we show in Section 4, spatial map-based depth estimation is highly accurate.

3.3 Track Objects in Motion

Once an object is detected, it must be tracked to allow the app to update any assistive overlays. The position of an object relative to the user can change as a result of two actions: (1) movement of the user, or (2) movement of the object itself.

User Movement. To account for user movement, we utilize the sophisticated inside-out tracking [17] capabilities of the HoloLens using a technique we call *object pinning*. Once an overlay is placed, the HoloLens can compensate for user motion to render it such that it appears stationary, as though it is "pinned" to a real-world location. By performing this computation locally, the app can update overlays smoothly with low latency and reduce the server's load by avoiding the need for continuous object detection.

Object Movement. For our room cleaning scenario, we make a simplifying assumption that objects may only be repositioned by the user. Thus, we use the following procedure to update overlays in response to object movement: when the user's hand is detected near an object with an overlay atop it, the overlay becomes "unpinned", updating

[2]In the next version of HoloLens, the HPU will support on-device acceleration of speech recognition.

[3]Specifically, we modify the Darknet implementation of YOLO (https://pjreddie.com/darknet/yolo/)

its position in real time as the user moves the corresponding physical object. When the user's hand moves away from the object, the overlay is pinned once more.

Our current prototype continuously streams video to the edge to perform both hand and other object detection. However, there is again an opportunity to again partition work between the HoloLens and the edge. For example, the HoloLens could employ a lightweight DNN to perform hand detection locally, to reduce communication overhead to the edge when all objects are in a pinned state. Furthermore, YOLO does not leverage the temporal locality of video streams, treating each individual frame independently. Thus, once an object of interest has been detected, the HoloLens could leverage its video encoder accelerator to extract motion information as part of the video encoding process, tracking objects locally at low cost and reducing the frequency and number of objects that need to be detected by the edge.

3.4 Determine the Orientation of Objects

In addition to an object's 3D position, it is often also necessary to know it's 3D orientation. For example, in our scenario, the direction that a piece of furniture (e.g., a sofa) is facing matters. The same principle applies to other domains, such as wood working or maintenance, in which instructions and overlays heavily depend on the orientation of physical objects. In increasing order of computational requirements (and precision of pose estimation), the options include two-dimensional matching by comparing to a set of template images of the object, using classifiers such as neural networks on RGB (and optionally depth) data to directly regress the pose of the object, and finally using a fine-grained 3D matching technique such as Iterated Closest Point (ICP) for fine adjustments [20]. While template matching can easily be done on (vector accelerated) device CPUs for coarse results, pose regression and ICP are best done off board.

3.5 Render High-Quality Overlays

The final step of any MR app is presenting overlays that blend with a view of the physical world to provide assistance or entertainment. In our case, rendering a realistic representation of target objects increases immersion and improves the user experience. HoloLens already supports 3D experiences at 60 FPS, but due to the fundamental volume, thermal, and power gaps between desktop and mobile GPU components, an edge node will be able to deliver richer and more life-like graphics than the mobile device for the foreseeable future. Offload rendering has been extensively studied in recent years [3, 4, 8, 16], and we envision collaborative acceleration taking advantage of existing techniques to deliver high-quality visuals for critical objects, while performing local rendering on minor or background objects.

4 PRELIMINARY EVALUATION

We next present a preliminary evaluation of collaborative acceleration applied to our MR app through a series of microbenchmarks on the HoloLens, modeled after the primitive operations described above. Specifically, we ask:

(1) Is HoloLens spatial mapping accurate enough for estimating the depth of objects?
(2) How expensive is performing object detection on a device like HoloLens vs. on an edge node?
(3) How effective is our pinning technique at reducing the latency of rendering updates when the user moves?

4.1 Methodology

We used a HoloLens running Windows Holographic 10 as our main client to explore questions (1) and (3). However, to our knowledge, there are no readily available DNN-based object detection implementations compatible with the HoloLens. Thus, to simulate the HoloLens's computational capabilities when profiling object detection (question (2)), we used an Intel Compute Stick 1st generation STCK1A32WFC powered by an Intel Cherry Trail Atom (similar to the CPU powering the HoloLens), with 2GB of RAM and running Ubuntu Linux 16.04. For our edge node, we used a HP Z420 workstation with 16GB of RAM and a Nvidia GTX 980Ti GPU also running Ubuntu Linux 16.04. The edge node uses Linux Network Emulation (netem) to insert queuing delays and control the RTT between the HoloLens and the edge node.

4.2 Depth Estimation Accuracy

Our first and second primitives (defining target locations and obtaining the 3D positions of detected objects) each require a depth reading to identify some position in 3D space. Fortunately, we observe that the spatial map generated by the HoloLens's HPU and depth sensor provides applications with an efficient mechanism for querying the geometry of the user's environment and estimating depth. However, if these depth estimates are not accurate enough, the application's overlays may be misplaced, leading to a poor user experience.

We profiled the accuracy of these depth estimates as follows. Using our room cleaning app in designer mode (Section 3), one researcher wearing a HoloLens stood in front of a table, looking down at it directly from above. They then defined the target locations of different objects (mouse, keyboard, and scissors), creating a hologram of each object at the points where their gaze intersected the spatial map. Since the HoloLens was oriented to face directly down, our depth estimation calculated the vertical distance between the HoloLens and the table. In this case, a perfect estimate would place the holograms exactly above the table. We physically marked the boundaries of the holograms on the table while still looking from above. We then oriented the HoloLens

parallel to the surface of the table and, using the marked boundaries to position a ruler, we measured the distance between the hologram boundaries and the table. We repeated the experiment 10 times for each object.

Our results (Figure 2a) show that the estimation error was consistently under 1 cm — accurate enough for overlay placement in our room cleaning application and similar apps where sub-centimeter guidance is not required. Applications where such precision is necessary (e.g. carpentry or surgery) can either use refined versions of our pinning algorithm (e.g. tracing multiple rays) or offload the estimation of the objects requiring additional precision to the edge.

4.3 The Cost of Object Detection

We argue that real-time object detection is too computationally expensive to be handled by a mobile device like the HoloLens. To verify this claim, we modeled the time it would take for the HoloLens to run the YOLO object detection framework on each frame of a video stream by using an Intel Compute Stick, power by a CPU similar to the one contained in the HoloLens. We focus on the HoloLens's CPU rather than its GPU under the assumption that the GPU is already devoted to the critical task of delivering a consistent stream of 3D overlays at 60 fps. We compared the performance of the Compute Stick to the performance of our Nvidia GPU-based edge node (Figure 2b). We used two different neural network model configurations for YOLO — YOLO and TinyYOLO, the latter being a more lightweight version for weaker devices, and we measured the performance on individual images as well as on a continuous video stream. Both the images and the video stream were recorded directly from the HoloLens using a native resolution (896x504) while looking at a table with the objects used in our app (mice, scissors, keyboards). We can easily observe that the performance of object detection using the Compute Stick is insufficient, taking several seconds per frame on TinyYOLO and almost a minute on full YOLO. We can also observe that the performance of YOLO on the edge node is several orders of magnitude faster, and indeed fast enough to deliver smooth animation updates (> 15 fps). While we would prefer detection rates of 30 fps or higher, more powerful GPUs or fine tuned DNN models would make this possible.

4.4 Pinning Effectiveness

Rather than continuously re-detecting an already-detected object as the user moves, pinning allows the HoloLens to internally track the user's movement, and thus to track the object's position relative to the user and display an accurate overlay with little delay. To understand the nature of this latency reduction, we generated a modified version of our room cleaning app with pinning disabled (*no-pinning*) for comparison against our original version (*pinning*). For each configuration, we measured the delay from the time a real-world object comes into the HoloLens's field of view to the time that an overlay is presented for that object. To perform these measurements, we used an iPhone 8 camera set to slow motion at 240 fps, with the camera lens placed directly behind the HoloLens's right eye lens, allowing it to capture both the overlays and the real world[4]. We present our results in Figure 2c, which shows that when using pinning, the HoloLens is capable of displaying overlays as fast as its display permits (240 fps divided among the 4 color channels that conform the image), resulting in a constant 4ms for all of our measurements. The overlay latency of no-pinning, however, varied greatly depending on both the network latency and the number of frames that YOLO had to process before passing our confidence threshold (Section 3). While this is in part an implementation artifact, it is evident that for any offloaded non-pinning implementation there will always be additional latency on top of the display refresh rate caused by network latency and object detection.

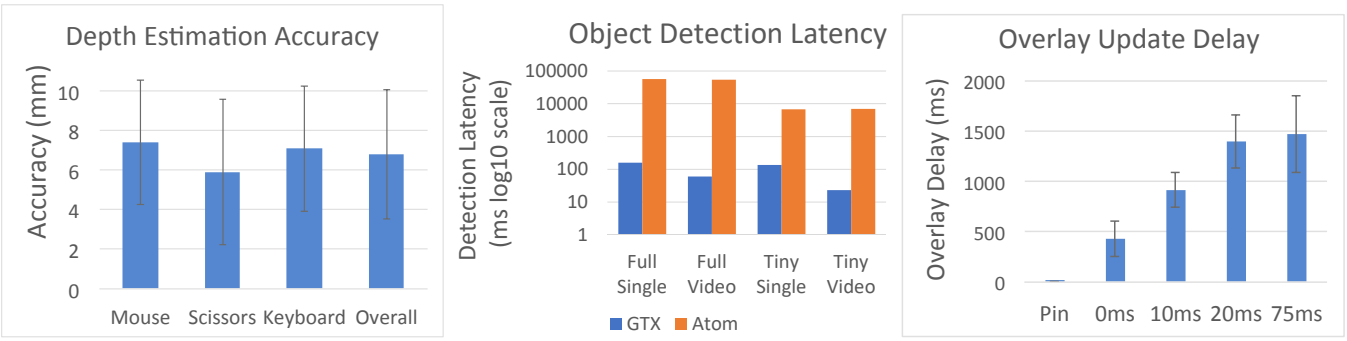
5 DISCUSSION AND FUTURE WORK

With an eye towards future hardware-accelerated MR ecosystems, we identify challenges opportunities for future work.

5.1 Device Heterogeneity and Accelerators

AR devices are diverse, with each presenting its own balance of cost, power consumption, and on-board capabilities. This diversity is often reflected in a heterogeneous set of CPUs, GPUs, and specialized accelerators. Furthermore, as devices evolve, we expect to see an increasing variety of accelerator-enabled operations. This trend raises the following questions: (1) How should collaborative acceleration adapt to heterogenous devices, and (2) once more and more operations become fully accelerated, what role will collaborative acceleration have? In the future, we plan to address both of these questions — the first, through an abstraction of the device's capabilities, along with application latency and accuracy requirements, that would allow a collaborative acceleration offload engine to determine which tasks or sub-tasks can be accelerated. With regards to the second question, we plan to build on top of our application to better understand the kinds of application-specific AI or rendering requirements that may not be fully implemented in hardware due to their less general applicability, and to characterize the opportunities they present to be collaboratively accelerated.

[4]We opted for this setup because the HoloLens does not allow recording the mixed reality feed (containing video of the physical world and any holograms present) while the RGB camera is being simultaneously used by an application (in our case, to stream to the edge for processing).



(a) Depth Estimation Accuracy *(b) YOLO Object Detection Latency* *(c) Pinning Effectiveness*

Figure 2: The results of our performance evaluation.

5.2 Variable Load and Availability

We have assumed implicitly that an application has the entire device and edge to itself. In practice, however, multiple applications will run in each location. Further, due to contention and environmental factors, the network between device and edge may vary in availability. Two fundamental questions therefore present themselves. First, how can the system help assure a graceful degradation in user experience as resources become scarce? For instance, although this is in the end an application-level concern, mechanisms for prioritization, feedback and conditional execution of components could lessen the load on the application developer. Second, what abstractions can the system provide to ensure that accelerators and other resources are used optimally across applications? Efficiently virtualizing accelerators such as GPUs across applications is poorly understood.

6 CONCLUSION

In this paper, we present collaborative acceleration, a technique that utilizes the unique hardware accelerated capabilities of MR devices in conjunction with the raw computational power of edge nodes. Through our prototype implementation of a 3D-task assistance app, we built three collaboratively accelerated techniques and proposed another two. We showed that for our evaluated primitives, it is possible to deliver sufficient accuracy with much lower latency than full offload or local execution. In future work, we intend to understand the power and performance characteristics of our primitives, especially in light of hardware acceleration trends and multi-application workloads.

REFERENCES

[1] Cuervo, E. Beyond reality: Head-mounted displays for mobile systems researchers. *GetMobile: Mobile Comp. and Comm. 21*, 2 (Aug. 2017).

[2] Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P. Maui: Making smartphones last longer with code offload. In *MobiSys 2010* (2010).

[3] Cuervo, E., and Chu, D. Poster: Mobile virtual reality for head-mounted displays with interactive streaming video and likelihood-based foveation. In *Proceedings of MobiSys 2016* (2016), pp. 130–130.

[4] Cuervo, E., Wolman, A., Cox, L. P., Lebeck, K., Razeen, A., Saroiu, S., and Musuvathi, M. Kahawai: High-quality mobile gaming using gpu offload. In *MobiSys* (May 2015).

[5] Gordon, M. S., Jamshidi, D. A., Mahlke, S., Mao, Z. M., and Chen, X. Comet: Code offload by migrating execution transparently. In *OSDI'12* (Berkeley, CA, USA, 2012), USENIX Association.

[6] Ha, K., Chen, Z., Hu, W., Richter, W., Pillai, P., and Satyanarayanan, M. Towards wearable cognitive assistance. In *MobiSys '14* (New York, NY, USA, 2014), ACM.

[7] Hu, W., Amos, B., Chen, Z., Ha, K., Richter, W., Pillai, P., Gilbert, B., Harkes, J., and Satyanarayanan, M. The case for offload shaping. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications* (New York, NY, USA, 2015), HotMobile '15.

[8] Lee, K., Chu, D., Cuervo, E., Kopf, J., Degtyarev, Y., Grizan, S., Wolman, A., and Flinn, J. Outatime: Using speculation to enable low-latency continuous interaction for cloud gaming. In *MobiSys'15*.

[9] Li, A., Yang, X., Kandula, S., and Zhang, M. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement* (2010), ACM, pp. 1–14.

[10] MagicLeap. Magic leap. https://www.magicleap.com/, 2017.

[11] Microsoft. Microsoft hololens. https://www.microsoft.com/microsoft-hololens/en-us, Apr. 2016.

[12] Microsoft. Spatial mapping. https://developer.microsoft.com/en-us/windows/mixed-reality/spatial_mapping, 2017.

[13] MSR-blog. Second version of hololens hpu will incorporate ai coprocessor for implementing dnns.

[14] Redmon, J., and Farhadi, A. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242* (2016).

[15] Reichert, C. Huawei unveils kirin 970 chipset with ai. http://www.zdnet.com/article/huawei-unveils-kirin-970-chipset-with-ai/, 2017.

[16] Reinert, B., Kopf, J., Ritschel, T., Cuervo, E., Chu, D., and Seidel, H.-P. Proxy-guided image-based rendering for mobile devices. *Computer Graphics Forum 35*, 7 (2016), 353–362.

[17] RoadToVR. Microsoft hololens inside out tracking is game changing for ar and vr.

[18] Satyanarayanan, M. The emergence of edge computing. *Computer 50*, 1 (2017), 30–39.

[19] Williams, C. Microsoft hololens secret sauce: A 28nm customized 24-core dsp engine built by tsmc. http://www.theregister.co.uk/2016/08/22/microsoft_hololens_hpu/, Oct. 2017.

[20] Wong, J. M., et al. Segicp: Integrated deep semantic segmentation and pose estimation. *CoRR* (2017).