OpenRTiST: End-to-End Benchmarking for Edge Computing

Shilpa George, Thomas Eiszler, Roger Iyengar, Haithem Turki, Ziqiang Feng, Junjue Wang, and Mahadev Satyanarayanan
Carnegie Mellon University

Padmanabhan Pillai
Intel Labs

***Abstract*—The growth of edge computing depends on large-scale deployments of edge infrastructure. Benchmarking applications are needed to compare the performance across different edge deployments and against device-only and cloud-only implementations. In this article, we present OpenRTiST, an open-source application that is simultaneously compute-intensive, bandwidth-hungry, and latency-sensitive. It implements a form of augmented reality that lets you "see the world through the eyes of an artist." We compare end-to-end application latency over varying network conditions and measure performance across a variety of edge platforms. OpenRTiST is designed to be easily deployed and has been used to showcase the benefits of edge computing.**

For edge computing to become an everyday reality, there is a need to quantitatively characterize the end-to-end path from a mobile or IoT device to its currently associated edge computing node or cloudlet[1]. This is especially true in the case of *edge-native applications* that require low latency and/or bandwidth scalability of edge-offloaded computation[2]. The end-to-end path includes many components: (a) capture, preprocessing, and encoding of image frames at the device; (b) network transmission to the cloudlet; (c) processing at the cloudlet; (d) network transmission of the result; and (e) rendering/display at the device. The term *motion-to-photon latency* is used to refer to the total delay experienced along this entire path. It is directly correlated with the subjective quality of user experience (QoE) for latency-sensitive applications such as augmented reality (AR). Today,

Digital Object Identifier 10.1109/MPRV.2020.3028781
Date of publication 19 October 2020; date of current version 9 November 2020.

Published by the IEEE Computer Society

there does not exist any widely used benchmark for this class of applications. The performance bottleneck on the end-to-end path determines system throughput.

Given the many variables on the end-to-end path and their complex inter-relationships, we do not yet know how to analytically predict the expected behavior of a specific AR/gaming application running on a specific device in a given networking and cloudlet setting. Experimentation with the actual application in the specific edge computing setting (i.e., using the real device, cloudlet, and network) is the only sure way to obtain rigorous results. However, in many cases, the application software may still be under development and may not be available in final form for many months. The ability to use an existing edge-native application as a stand-in for the real application is valuable in these circumstances. Parameterization of the stand-in application to generate different levels of bandwidth demand and cloudlet processing demand is also valuable because it enables sensitivity analysis in the face of uncertainty.

Toward these goals, we have created and open-sourced *OpenRTiST,* an edge-native interactive application that performs a computationally expensive real-time transformation of a live video feed. Similar to other AR applications, OpenRTiST has a high network load in both directions. Thus, OpenRTiST is simultaneously compute-intensive, bandwidth-hungry, and latency-sensitive. This trifecta of attributes offers the most compelling rationale for edge computing [3]. OpenRTiST is simple to understand and easy to deploy, yet offers visceral impact and intuitive insights into edge computing. Most importantly, it can be used as a workload generator in controlled benchmarks to measure motion-to-photon latency, estimate scalability, and identify performance bottlenecks.

The core functionality of OpenRTiST is *style transfer,* a well-known computer vision technique for transforming images [4,5]. Offline machine learning is used on a reference image (such as a famous painting) to extract its stylistic essence, and embody it in a deep neural network (DNN). Through runtime inferencing on this DNN, the scene captured by a video camera is transformed in real time to acquire the stylistic attributes of

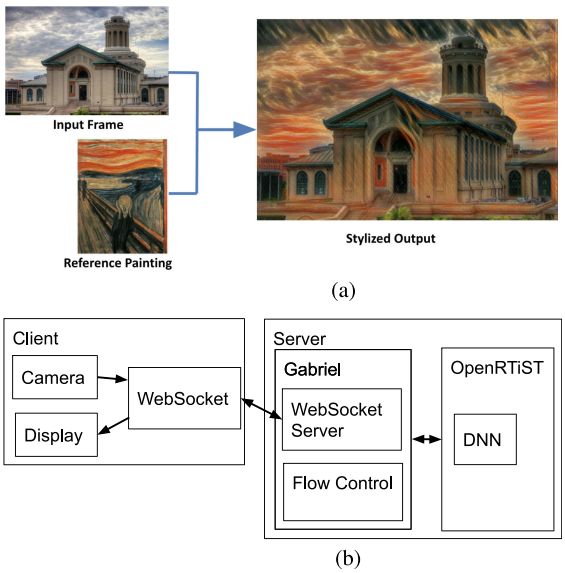


Figure 1. Example and architecture of OpenRTiST. (a) OpenRTiST example. (b) System architecture.

the learned image [see example in Figure 1(a)]. This transformed output can be displayed on a big screen for easy viewing by large groups of people, on a smartphone for personal viewing, or on a head-up display to give an immersive experience. Image resolution and frame rate can be easily-tuned to vary the network and compute load generated by OpenRTiST. In the rest of this article, we describe the design, implementation, and evaluation of OpenRTiST.

SYSTEM ARCHITECTURE

Overview

OpenRTiST, which stands for Open Real-Time Style Transfer, uses neural style transfer (NST) to transform a live video stream by "painting" it in the style of a reference painting. NST is a computer vision technique first proposed by Gatys *et al.* [4] that leverages a DNN to first extract the stylistic essence of a *reference* image and then blend it with a *content* image to create an output image that looks like the content image painted in the style of the reference image. Our implementation is based on an efficient NST implementation and training methodology described by Johnson *et al.*[5] and made available as an open-source PyTorch module. The DNN consists of 16 convolutional layers, and requires approximately 23.7×10^9 operations to compute the

stylized output for a 320×240 image. Since OpenRTiST applies style transfer to a live stream of frames instead of individual images, we modified the training algorithm to preserve the consistency of appearance across frames and reduce flickering effects.

To execute OpenRTiST in real time in a mobile context, we leverage the Gabriel framework, which is designed to minimize the latency of computational offloading in the context of wearable cognitive assistance applications[6]. Figure 1 (b) shows the system architecture of OpenRTiST. Frames captured by the client device are transmitted to the server. The server runs this frame through the DNN and sends the resulting frame back to the client to render on the user's display.

Our server runs two POSIX processes. The Gabriel process runs a websocket server to communicate with clients, and it manages flow control. The OpenRTiST engine process implements the style transfer component, using a transformer DNN. Users can control the resolution of images, which will affect the amount of computation done by the transformer DNN.

OpenRTiST's attributes make it a good workload to benchmark the performance of edge platforms for AR-like applications. First of all, the application is computationally demanding and prohibitively expensive to run in a mobile context without computational offload. As both input and processed images are transferred across the network, it is bandwidth-intensive on both the uplink and downlink. Furthermore, as a real-time application, OpenRTiST is latency-sensitive: Any lag in displaying the processed results is obvious to a human observer.

Supporting Edge and Device Heterogeneity

Cloudlets can span a broad spectrum of form factors with varying capabilities, from rack servers with powerful discrete GPUs to nettops, such as Intel NUC. To support DNN inference on a variety of platforms, OpenRTiST uses the PyTorch[7] framework by default. This framework has been well optimized for discrete Nvidia GPUs, and recent versions take advantage of the Intel Math Kernel Library for Deep Neural Networks (MKL-DNN) to efficiently utilize vector operations on CPUs.

Alternatively, OpenRTiST can utilize the Intel OpenVino toolkit[8]. The toolkit enables heterogeneous execution of DNNs across a wide variety of Intel hardware such as CPUs, integrated GPUs (iGPU), Movidius VPUs, and some FPGAs, although, to date, we have focused our testing on CPUs and iGPUs.

Likewise, on the client-side, OpenRTiST can run on various target devices. An Android client supports phones, tablets, and some wearable devices. A Python client can run on most nonmobile systems, enabling large, multiviewer demos on projected or flat-panel displays.

Performance Metrics

The performance of an edge-native application is dependent on several variables such as the client device, the network quality, and the computational power at the server.

There are three types of latency used in this article: round-trip time (RTT), end-to-end (E2E) latency, and motion-to-photon (MTP) latency.

These terms are defined as follows:

- RTT: Just the round-trip network latency, nothing else;
- E2E: RTT + server compute time;
- MTP: device capture + E2E + device display.

In the "Evaluation" section, we evaluate the performance of OpenRTiST for different client and server configurations over different network topologies by capturing metrics, such as bandwidth utilization, E2E, and MTP latencies.

EVALUATION

Experimental Setup

We investigate the performance of OpenRTiST on different edge and cloud platforms over WiFi and LTE networks. Table 1 summarizes the hardware configuration of the platforms used in the experiments. We use four different cloudlets to evaluate edge performance. Cloudlet-GPU1 and Cloudlet-GPU2 are two similarly powerful servers with discrete GPUs. We also include Cloudlet-iGPU, a small form factor edge server based on an Intel NUC containing an integrated GPU as well as a Cloudlet-CPU server-grade machine with no graphics card.

To evaluate the performance of offloading to the cloud, we use two instances with different network latency characteristics in Azure East US and

Table 1. Hardware Used in Experiments and Performance on 240p Frame.

Type	Name	CPU	GPU	Processing time (ms)	E2E latency (ms)
Edge	Cloudlet-GPU1 (PyTorch)	2× Intel Xeon Gold 6144, 8 cores @ 3.50GHz	NVIDIA Tesla V100	10	24 (4)
Edge	Cloudlet-GPU2 (PyTorch)	2× Intel Xeon E5-2699v3, 18 cores @ 2.30GHz	NVIDIA GTX 1080 Ti	21	39 (9)
Edge	Cloudlet-iGPU (OpenVino)	Intel Core i7-6770HQ, 4 cores @ 2.60GHz	Integrated	36 (1)	75 (8)
Edge	Cloudlet-CPU (OpenVino)	2× Intel Xeon Gold 5115, 16 cores @ 2.40GHz	-	77 (6)	119 (14)
On-device	Essential Phone	8 core Kryo 280 @ 2.45GHz	-	3887 (551)	3887 (551)
Cloud	Azure East US	Intel Xeon E5-2690v4 6 cores @ 2.60GHz	NVIDIA Tesla P100	13	58 (27)
Cloud	Azure West US	Intel Xeon E5-2690v4 6 cores @ 2.60GHz	NVIDIA Tesla P100	13	105 (72)

Azure West US, both with NVIDIA Tesla P100 GPUs. The NVIDIA Tesla V100 GPU on Cloudlet-GPU1 has better inference performance characteristics as compared to the Pascal GPUs utilized by the cloud instances. In contrast, the NVIDIA GTX 1080 Ti GPU on Cloudlet-GPU2 is approximately 30% slower than those used by the cloud instances. In the "Network Effects on End-to-End Latency" section, we see that lower network latency to the edge results in higher throughput using Cloudlet-GPU2 as compared to more powerful cloud servers.

The workload of OpenRTiST can be modified by configuring the frame resolution and/or the frame rate of capture. We tested the performance of OpenRTiST for video feeds with a 4:3 aspect ratio at 30fps and varying resolutions of 240p, 360p, 480p, and 720p.

Performance on Edge

In this section, we explore the performance of running OpenRTiST on the four edge platforms mentioned in the "Experimental Setup" section. We run OpenRTiST using the PyTorch framework on Cloudlet-GPU1 and Cloudlet-GPU2 and we utilize OpenVino on Cloudlet-iGPU and Cloudlet-CPU. We also run OpenRTiST on an Essential smartphone (PH-1) to measure the performance of OpenRTiST in a device-only setting.

Table 1 summarizes the results, showing mean and standard deviation for processing time as well as E2E latency for an input feed of 240p. Running OpenRTiST on a reasonably performing smartphone takes about 4 s of processing per frame, making it prohibitive to run without offloading. Running OpenRTiST on CPU-only devices is also problematic, with the mean E2E latency surpassing 100 ms on Cloudlet-CPU. Low latencies can only be reliably achieved on edge devices with hardware acceleration, such as a discrete or integrated GPU.

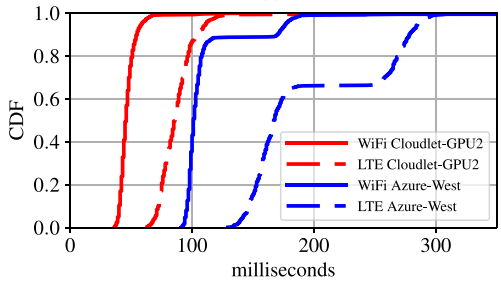
Network Effects on End-to-End Latency

The E2E latency as observed by the user is affected by both the computational processing time on the server as well as the network conditions between the client and the server. In this section, we evaluate the performance of OpenRTiST over different network technologies (WiFi and LTE) as the network distance is increased. Figure 2 (a) measures OpenRTiST's performance for a 240p resolution video feed running on a cloudlet instance located near the client (Cloudlet-GPU2) as well as against two increasingly remote cloud instances, Azure East US and Azure West US. For the user device, we use an Essential Phone (PH-1) configured to capture 320 × 240 resolution frames.

To reach the cloudlet over cellular data, we use an experimental LTE network, indicative of future 5G CBRS Band 48 networks. It is comprised of three small cells and a virtualized RAN that are connected via optical fiber. This allows us to be a single hop away from the computational

	Using WiFi		Using LTE	
	Median	90%	Median	90%
Cloudlet-GPU2	44	55	86	104
Azure-East	59	87	140	171
Azure-West	101	170	167	278

(a)



(b)

Figure 2. E2E latency over WiFi versus LTE. (a) E2E latency of OpenRTiST in milliseconds. (b) Cumulative Distributive Function of E2E latency.

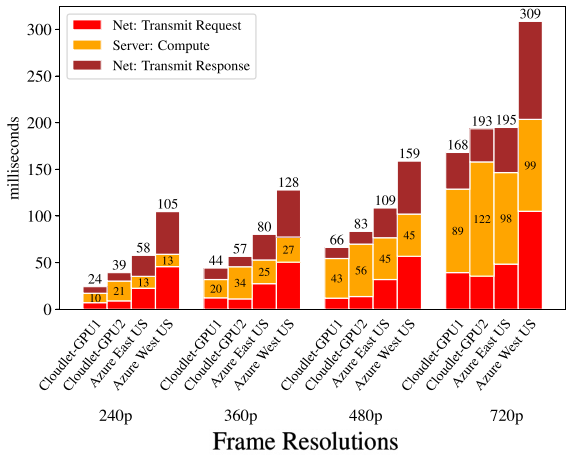


Figure 3. E2E latency breakdown over WiFi.

resources in our lab. The network operates in the 3.5GHz spectrum (band 42), and only a few phones such as an Essential Phone (PH-1) can connect to it. We use a commercial 4G LTE network (T-Mobile) to reach the Azure cloud datacenter.

Figure 2(b) illustrates how the higher latencies of LTE cause the response time CDF curves to shift to the right of the results achieved over WiFi. In both cases, offloading to Cloudlet-GPU2 provides the best response times despite its GPU being less powerful than those of the cloud instances. The lower network latency to the cloudlet offsets its slower frame processing capabilities to give better response time.

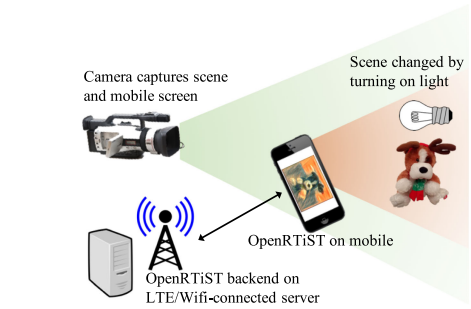
Payload Size

The network and compute load of OpenRTiST can be adjusted by tuning the resolution and framerate of the video feed. In this section, we explore how image quality affects E2E latency and network bandwidth. Table 2 summarizes the average bandwidth utilization and E2E latency of various image resolutions for OpenRTiST's backend running on Cloudlet-GPU1. The video feed is captured at 30fps. Prior knowledge of the offered bandwidth of an application for different image quality can help adapt the payload in the presence of congestion or to support scalability.

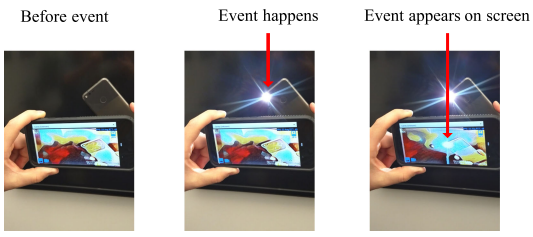
Figure 3 shows the E2E latency of OpenRTiST for video feeds with a 4:3 aspect ratio across varying resolutions of 240p, 360p, 480p, 720p, and multiple offloading edge and cloud platforms. For a 240p resolution video feed, the response time at the slower Cloudlet-GPU2 is 1.5× faster than the Azure East server and almost 3× faster than the Azure West server. As noted in the previous section, the lower network latency to the cloudlet gives the best E2E performance even when requiring longer server-side processing.

Table 2. Bandwidth and E2E Latency on Cloudlet-GPU1.

Resolution	Avg. request size	Avg. response size	BW to/from Cloudlet	E2E latency (ms)	
	(kB)	(kB)	(Mb/s)	Median	90%
240p	9.89	11.56	1.98 / 2.32	23.0	26.0
360p	16.22	23.02	2.97 / 4.22	44.0	48.4
480p	23.11	37.87	3.86 / 6.32	65.6	72.1
720p	40.83	80.11	4.36 / 8.56	165.0	186.7



(a)



(b)

No-Op	Cloudlet-GPU2	Azure East US	Azure West US
100 ± 10	156 ± 25	172 ± 35	372 ± 54

Numbers are mean ± stdev in milliseconds.

(c)

Figure 4. Setup and results of MTP latency.
(a) Setup for measuring motion-to-photon latency.
(b) Example frames from observer camera.
(c) Benchmarked motion-to-photon latency.

Motion-to-Photon Latency

A more relevant metric than E2E latency for AR or gaming applications is MTP latency. This metric is defined as the elapsed time between a user movement and when the response to that action is rendered on the screen. High MTP latency leads to the user's perception that the transformed video is delayed.

MTP latency differs from E2E latency in that camera and display lag are also included. A slow camera and display can hurt MTP latency (and hence user experience) even when edge computing is used to achieve low E2E latency. In fact, the results shown in Figure 4(c) illustrate precisely this point.

We measured OpenRTiST's MTP latency using the mobile client, a flashlight, and a separate observer camera, as shown in Figure 4(a). The observer camera simultaneously captures the flashlight both directly and through OpenRTiST's display on the mobile phone. As shown in Figure 4(b), we analyze the captured video to determine when the light appears and when it is captured on the client's display. As the observer camera captures at 60 fps, the granularity of measurement is 17 ms.

Figure 4(c) shows the MTP latency results when OpenRTiST is offloaded to different edge/cloud locations over WiFi. Note that we also report the No-Op latency, where the scene is captured and simply displayed by the mobile phone. This indicates the lower bound on the latency on the mobile device, due to the camera's exposure time, sensor and OS overhead, and rendering/display delay. This high baseline shows how the application QoE is dependent on the display device and the need for low-latency displays.

The MTP latency of 100 ms even for a No-Op mutes the benefit of edge computing. If a much better device were to be used (leading to a smaller value than 100 ms for the No-Op latency), the benefit of edge computing would be magnified.

REAL-WORLD DEMONSTRATIONS

OpenRTiST is a good application workload to benchmark edge deployments. Also, a non-technical audience can gain first-hand experience with edge computing using the application as it is both fun and easy to understand. The ability to compare a live video stream transformed on a distant cloud and the same video stream transformed at the edge results in an immediate appreciation of the value of edge computing. OpenRTiST's back-end is encapsulated in a container [9] and the client is publicly available on the Google PlayStore [10]. This easy onboarding makes OpenRTiST highly conducive to testing new edge infrastructures. Below are some examples of how OpenRTiST has been used to showcase edge computing outside CMU. Figure 5 shows some of the scenes captured during the demonstrations.

At the Edge Computing Congress in London (2019), InterDigital used OpenRTiST to demonstrate its edge emulation platform called AdvantEDGE. Figure 5(b) shows their booth during the conference. They used OpenRTiST to test the platform's location service and evaluate the impact of network latency on user experience.

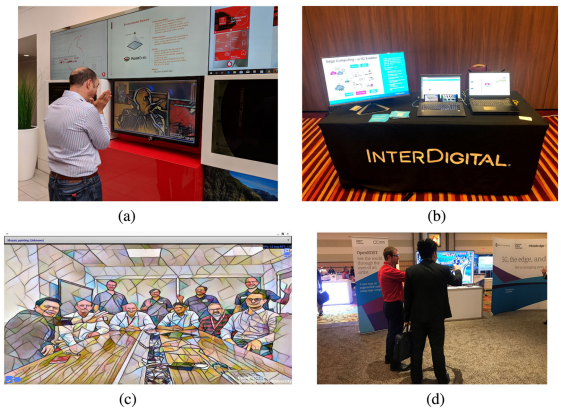


Figure 5. Demonstrations of OpenRTiST. (a) OpenRTIST at Vodafone, Newbury, U.K. (b) Interdigital Demo in London. (c) Event attendees in Ottawa. (d) ConnectX in Orlando, FL, USA.

In partnership with Crown Castle, America's largest provider of shared communications infrastructure, we demonstrated OpenRTiST in May 2019 at the ConnectX Expo [11]. The show floor of this 5G infrastructure conference in Orlando, FL, was a great place to display the power of edge computing. Hundreds of attendees over a 2-day period witnessed what edge computing could do in this setting [see Figure 5 (c) and (d)].

OpenRTiST was also demonstrated as an exhibit in the lobby of Vodafone Group's research lab in Newbury, U.K. To quote Guenter Klas, Senior Manager R&D at Vodafone Group: "Vodafone tested OpenRTiST at our HQ in Newbury, UK to bring edge computing to life for employees and business partners, in a deliberately non-technical, more engaging way. Art turned out to be a good channel for this purpose. The importance of real-time edge computing can be well conveyed with OpenRTiST."

RELATED WORK

The Yahoo! Cloud Serving Benchmark [12] and DCBench [13] contain standardized sets of data-intensive workloads, designed to profile cloud services and parts of datacenters. DAWNBench [14] and MLPerf [15] compare the speed and accuracy of machine learning models with different parameters and hardware configurations. EdgeBench [16] and Defog [17] compare the performance of running workloads at the edge with running them in the cloud. Both benchmarks measure network throughput, latency, and computational resources used. EdgeBench and Defog both include neural network inference workloads. However, their experiments were not conducted using hardware accelerators. Our experiments were run using a GPU.

The edge applications [16,17] can be mainly classified as follows:

1) scalar/sensor applications: where the inputs are scalar representing sensor data.
2) audio/text applications: where an audio input stream is transcribed to text.
3) image applications: which take images as input and returns scalar results.

OpenRTiST is interactive, latency-sensitive, and it represents the first AR workload used for benchmarking edge deployments that we are aware of. Most mobile applications download significantly more data than they upload, whereas OpenRTiST is bandwidth-intensive in both directions.

In this article, we consider the full end-to-end path of OpenRTiST. This starts with capturing an image from the user's camera and ends with displaying the transformed version of this image, as shown in the "Motion-to-Photon Latency" section. We also benchmark the compute performance, end-to-end latency, and effective network bandwidth used for different resolution settings.

CONCLUSION

We have built and demonstrated OpenRTiST, an application that is compute-intensive, bandwidth-hungry, and latency-sensitive. These attributes make it an excellent application to perform end-to-end benchmarking of edge platforms. We have designed it to be easily deployed in a variety of contexts and useful in demonstrating the value of edge computing in a visually interesting and easily understood manner. OpenRTiST has been adopted in several industry demonstrations of edge technologies. Our measurements show that offload is necessary in mobile contexts and that cloudlets can significantly outperform cloud deployments. Furthermore, motion-to-photon latency is significantly limited by internal delays of the mobile client itself. We believe improved low-latency cameras and displays will be required on future mobile devices to enable low-latency interactive applications, such as AR.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their feedback, which helped to improve this article. This work was supported in part by the Defense Advanced Research Projects Agency under Contract HR001117C0051, in part by the National Science Foundation under Grant CNS-1518865, in part by Intel, in part by Vodafone, in part by Deutsche Telekom, in part by Crown Castle, in part by InterDigital, in part by Seagate, in part by Microsoft, in part by VMware, and in part by the Conklin Kistler family fund. The work of Roger Iyengar was supported by an NSF Graduate Research Fellowship under Grant DGE1252522 and Grant DGE1745016. Any opinions, findings, conclusions, or recommendations expressed in this article are those of the authors and do not necessarily reflect the view(s) of their employers or the above funding sources.

REFERENCES

1. M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct.–Dec. 2009.
2. M. Satyanarayanan, G. Klas, M. Silva, and S. Mangiante, "The seminal role of edge-native applications," in *Proc. IEEE Int. Conf. Edge Comput.*, Milan, Italy, Jul. 2019, pp. 33–40.
3. K. Ha *et al.*, "The impact of mobile multimedia applications on data center consolidation," in *Proc. IEEE Int. Conf. Cloud Eng.*, San Francisco, CA, Mar. 2013, pp. 166–176.
4. L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2414–2423.
5. J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 694–711.
6. K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proc. Annu. Int. Conf. Mobile Syst., Appl., Services*, 2014, pp. 1–9.
7. A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Adv. Neural Inf. Process. Syst.*, pp. 8026–8037, 2019.
8. Intel, "Openvino toolkit," 2019. [Online]. Available: https://software.intel.com/en-us/openvino-toolkit
9. D. Hub, "cmusatyalab/openrtist," 2018. [Online]. Available: https://hub.docker.com/r/cmusatyalab/openrtist
10. Application on Google Play, "OpenRTiST." [Online]. Available: https://play.google.com/store/apps/details?id=edu.cmu.cs.openrtist
11. O. E. C. Initiative, "ConnectX 2019," May 2019. [Online]. Available: https://www.openedgecomputing.org/connectx-2019/
12. B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proc. 1st ACM Symp. Cloud Comput.*, 2010, pp. 143–154.
13. Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo, "Characterizing data analysis workloads in data centers," in *Proc. IEEE Int. Symp. Workload Characterization*, 2013, pp. 66–76.
14. C. Coleman *et al.*, "DAWNBench: An end-to-end deep learning benchmark and competition," *Training*, vol. 100, no. 101, 2017, Art. no. 102.
15. V. J. Reddi *et al.*, "MLPerf inference benchmark," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Architecture*, 2020, pp. 446–459.
16. A. Das, S. Patterson, and M. Wittie, "EdgeBench: Benchmarking edge computing platforms," in *Proc. IEEE/ACM Int. Conf. Utility Cloud Comput. Companion*, 2018, pp. 175–180.
17. J. McChesney, N. Wang, A. Tanwer, E. de Lara, and B. Varghese, "DeFog: Fog computing benchmarks," in *Proc. 4th ACM/IEEE Symp. Edge Comput.*, 2019, pp. 47–58.

Shilpa (Anna) George is working on interactive systems at the edge under the guidance of Prof. M. Satyanarayanan. Her research interest lies in edge computing and computer vision. She received the B.Tech. degree in electrical and computer engineering from the Indian Institute of Technology Kharagpur, Kharagpur, India, and the M.S. degree in electrical and computer engineering from Carnegie Mellon University (CMU), Pittsburgh, PA, USA. She is currently working toward the Ph.D. degree with in Computer Science Department, CMU. Contact her at shilpag@cs.cmu.edu.

Thomas Eiszler is currently a senior research scientist with the Computer Science Department, Carnegie Mellon University (CMU), Pittsburgh, PA, USA. As a part of the Living Edge Lab, his research focus revolves around systems and applications designed to leverage edge computing. Prior to his research career, he worked on designing and implementing systems in the healthcare and mobile industries. He received the B.S. degree in computer science from Pennsylvania State University, State College, PA, USA. Find him on Github at https://github.com/teiszler. Contact him at teiszler@andrew.cmu.edu.

Roger Iyengar is supported by a Graduate Research Fellowship from the National Science Foundation. His research interests include mobile systems and edge computing. He is currently working toward the Ph.D. degree in computer science at Carnegie Mellon University, Pittsburgh, PA, USA. Contact him at raiyenga@cs.cmu.edu.

Haithem Turki's current research interests lie at the intersection of edge computing and machine learning. He received the B.S. and M.S. degrees in computer science from Stanford University, Stanford, CA, USA. He is currently working toward the Ph.D. degree in computer science at Carnegie Mellon University, Pittsburgh, PA, USA. Contact him at hturki@cs.cmu.edu.

Ziqiang Feng's research interests include machine learning, edge computing, and distributed systems. He received the B.S. and M.Phil. degrees from the Hong Kong Polytechnic University, Hong Kong. He is currently working toward the Ph.D. degree in the Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA. Contact him at zf@cs.cmu.edu.

Junjue Wang's research interest lies at the intersection of edge computing, computer vision, and human–computer interaction. He received the B.S. degree in computer science from the University of Wisconsin–Madison, Madison, WI, USA, the M.S. degree in computer science from Carnegie Mellon University (CMU), Pittsburgh, PA, USA, and the Ph.D. degree in computer science from CMU, advised by Prof. M. Satyanarayanan. Contact him at junjuew@alumni.cmu.edu.

Padmanabhan (Babu) Pillai is currently a senior research scientist with Intel Labs, Santa Clara, CA, USA. His research interests include the use of edge computing to support immersive interactive experiences, and perform real-time understanding of the world. He received the B.S. degree in electrical and compter engineering from Carnegie Mellon University, Pittsburgh, PA, USA, and the M.S. and Ph.D. degrees in computer science from the University of Michigan, Ann Arbor, MI, USA. Contact him at padmanabhan.s.pillai@intel.com.

Mahadev (Satya) Satyanarayanan is currently the Carnegie Group University professor in computer science at Carnegie Mellon University (CMU), Pittsburgh, PA, USA. His multidecade research career has focused on the challenges of performance, scalability, availability, and trust in information systems that reach from the cloud to the mobile edge of the Internet. In the course of this work, he pioneered many advances in distributed systems, mobile computing, pervasive computing, the Internet of Things, and, most recently, edge/fog computing. He received the Ph.D. degree in computer science from CMU. He is a Fellow of the ACM and IEEE. Contact him at satya@cs.cmu.edu.