Split Computing and Early Exit Done Right

Roger Iyengar, Qifei Dong, Chanh Nguyen,
Padmanabhan Pillai[†], Mahadev Satyanarayanan

[†]Intel Labs

March 2023
CMU-CS-23-102

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We make the case that a wide array of techniques for split computing (SC) and early exit (EE) exist beyond DNN-only approaches. Practitioners should consider all of these posiblities, and recognize the difficulty of modifying a complex DNN architecture. We offer a design strategy for edge-native applications, to help take advantage of split computing and early exit strategies. We used this strategy to successfully develop four wearable cognitive assistance applications, and demonstrated that some relatively simple SC and EE strategies offered a significant savings in bandwidth usage. Lastly, we showed that achieving the best possible accuracy for our applications require the use of edge computing.

This research was supported by the National Science Foundation (NSF) under grant number CNS-2106862. This work was done in the CMU Living Edge Lab, which has been supported by Intel, ARM, Vodafone, Deutsche Telekom, CableLabs, Crown Castle, InterDigital, Seagate, Microsoft, the VMware University Research Fund, and the Conklin Kistler family fund. Any opinions, findings, conclusions or recommendations expressed in this document are those of the authors and do not necessarily reflect the view(s) of their employers or the above funding sources.

Keywords: edge computing, cloud computing, mobile computing, pervasive computing, IoT, offload, offload shaping, augmented reality, edge AI, machine learning, wearable cognitive assistance, low latency

1 Introduction

How to partition a deep neural network (DNN) so that its lightweight *head* executes on a mobile device, and its heavyweight *tail* executes in the cloud or on an edge computing node (cloudlet) is a hot topic today [7, 25, 17, 23, 19, 4, 27]. The partitioning is called *split computing (SC)* in the machine learning (ML) literature. When combined with *early exit (EE),* where the head result indicates that tail execution is unnecessary, it can significantly lower network bandwidth demand and cloudlet load without serious degradation of accuracy for recognition/classification tasks in mobile computer vision.

This position paper observes that splitting such tasks into head and tail components is indeed valuable, but formulating the problem as DNN-partitioning is too narrow and rigid. In addition, such an approach requires mobile system designers to have deep ML skills, well beyond the basic ML skills that they typically possess. We observe that the idea of a cheap head, followed by an expensive tail has been a feature of compute offload for mobile devices from its earliest roots. Further, the idea of early exit without executing the tail has long been used by the mobile computing community under the names "early discard" and "offload shaping." That older framing offers more general and powerful optimizations, and subsumes DNN-focused EE. We provide experimental results that confirm the value of the broader problem formulation.

We conducted a case study for wearable cognitive assistance (WCA), an important emerging class of applications that are both bandwidth intensive and latency sensitive. Our study demonstrated the effectiveness of SC and EE strategies that do not require specialized ML skills to implement. In addition, we examined how well WCA applications perform when they are run entirely on a mobile device, without offloading any parts of the computation to a cloudlet.

2 Evolution of Key Concepts

The essence of SC, splitting a mobile application into a lightweight head and a heavyweight tail, has been integral to offloading since birth. Figure 1, reproduced from the 1997 Odyssey paper by Noble et al [31], shows the very first use of offloading to amplify the capabilities of a mobile device. The Janus speech recognition application was modified to operate in one of three modes in Odyssey. In one of the modes, a prelimary phase of speech processing was done locally (i.e., the "head"), and the extracted information was shipped to a remote server for the completion of the recognition process (i.e., the "tail"). For certain combinations of network bandwidth and device/server capabilities, this split offered lower end-to-end latency than fully local or fully remote execution. In 1999, Flinn et al [11] showed how SC could extend battery life. Abstracting and generalizing from these efforts, the concept of "cyber foraging" was introduced in 2001 [35]. Today, we use the term "of-

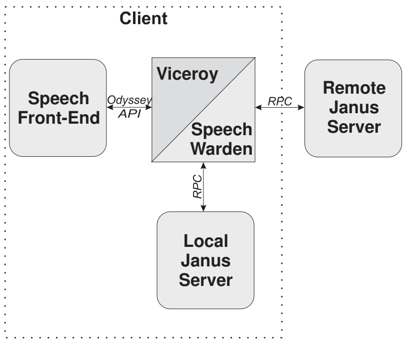

Figure 1: First Use of Offloading [31]

floading" instead of "cyber foraging." Building on this foundational work, the period from 2001-2008 saw vigorous research activity[1, 9, 10, 30, 14, 43, 32, 2, 22]. Flinn's survey [8] provides a detailed account of these efforts. Between 2009 and 2015, MAUI [6], Odessa [33], CloneCloud [5], and COMET [13] explored programming language and virtual machine support for offloading. In 2015, Hu et al [18] introduced offload shaping:

"... we show that it is sometimes valuable to perform additional cheap computation, not part of the original pipeline, on the mobile device in order to modify the offloading workload. We call this offload shaping. We show that offload shaping can be applied at many different levels of abstraction using a variety of techniques, and that it can produce significant reduction in resource demand with little loss of application-level fidelity or responsiveness."

The "cheap computation" mentioned above can take many forms including blur detection using vision algorithms [37, 38], IMU-based blur detection, and perceptual hashing [44, 29, 21] to detect nearly similar frames for deduplication. Although the head algorithms explored by Hu et al did not include DNNs, the 2018 extensions of this concept by Wang et al [41] did use DNNs at the head to implement early discard of video frames. The head DNN is cheap but not very accurate; the DNN at the cloudlet (tail) is far more accurate. Together, the cascaded pair of DNNs achieve good precision and recall. They effectively implement SC and EE without the algorithmic challenges of splitting a DNN or designing a bottleneck. In 2019, Wang et al [42] showed how these single-user SC and EE concepts could be leveraged to improve the multi-user scalability of edge-native applications.

The DNN-centric focus of SC for mobile devices began in 2017, with Kang et al [20]. That work did not include the EE concept. A different group of researchers had already described the EE concept for DNNs in 2016, without explicit mention of its relevance to mobile computing [40]. *Couper,* a 2019 system by Hsu et al [17], showed how production DNNs could be optimally sliced for SC in different hardware settings. No architectural additions or modifications beyond splitting were made to the DNNs. In 2019, Eshratifar et al [7] and Matsubara et al [25] independently pointed out that SC was especially valuable when combined with EE for processing data captured on mobile devices. A 2022 survey [27] highlights the large volume of work in this space since 2019.

The fact that the code executed on device (i.e., the head) is not part of the original pipeline may seem to be a weakness of offload shaping/early discard relative to DNN-only SC with EE. However, this apparent weakness is offset by the need to create a bottleneck in DNN-only approaches. The code that implements the bottleneck is not part of the original DNN; it is created solely to make SC work effectively and represents extra work relative to the original pipeline.

3 The Limits of DNN-only SC and EE

From this 25-year long evolution of SC and EE concepts, the take-away message can be distilled as follows:

- SC is a valuable performance optimization for offload from mobile devices. It has withstood the test of time, and is likely to remain highly relevant well into the future. The head may or may not involve a DNN.

- EE is also likely to remain important for the foreseeable future. It can take many forms, depending on the specific algorithm and the capabilities of the mobile device. The head code determining whether tail execution is necessary may or may not involve a DNN.

To these firmly-grounded observations of the past, we add a third that is forward-facing and possibly controversial:

- *DNN-only SC/EE is a dead end for mobile computing.* It relies critically on ML expertise, a resource that is scarce and growing ever scarcer. Scaling up or scaling out SC/EE into widespread everyday use will be a failure if it is restricted to DNN-only approaches.

Modifing DNN internals is not for the faint of heart. Structural changes to the internals of a DNN requires a much higher level of ML skill from those needed to re-train an off-the-shelf DNN via transfer learning, or to use such a DNN for inferencing. That DNN internals can be modified successfully by ML experts is not in question. What is doubtful is whether developers with basic ML skills can accomplish similar feats. As the ML experts themselves state [26]:

> "...the complex structure of CNN-based object detectors poses unique challenges in designing effective splitting approaches. ... it is not possible to reduce the inference time of CNN-based object detectors by naive splitting methods without altering the models' architecture. This is due to the designs of the early layers of the models, which amplify the input data size."

Prophetically, the first paper to describe DNN-only SC [20] used the system name "Neurosurgeon." Indeed, DNN-only SC/EE is like brain surgery — it is not wise for even a well-trained general surgeon to attempt it! Since ML expertise is a chronically scarce resource, it is self-defeating for mobile system designers to become critically reliant on it. A viable alternative, described in the next section, is readily available.

4 Design Strategy

- *Make it work on the cloudlet first:* Construct the tail assuming a null head. The tail is often a single DNN. However, it may sometimes involve a DNN cascade. For example, an object detector may find relevant regions of a frame, and a fine-grain classifier may perform precise classification of each detected object. The important point is to design the tail to achieve the desired accuracy and speed, assuming that the input data is available locally. Since the tail executes entirely on the cloudlet, all issues pertaining to mobile device limitations and networking limitations can be ignored at this stage. The execution of this tail represents the best case from a performance point of view. It assumes that the head contributes zero frame capture and encoding latency, and that the network has zero latency and infinite bandwidth. Optimize the tail until the desired speed and accuracy are achieved.
- *Create the lightest possible head:* Construct a minimal head that captures live data, followed by encoding and transmission to the cloudlet. Avoid EE optimizations of any kind. Ensure that the worst-case end-to-end latency requirement is met. Until you can meet this requirement, don't bother trying to add EE. It will only improve the average case, not the worst case.

- *Enrich the head:* Find the lightest-weight head optimization that achieves significant EE for the test data. In many cases, this may not involve a DNN. It may use one of the simple mechanisms described for offload shaping by Hu et al[18]. In other cases, as in Wang et al [41], it may involve a cheap but weak DNN to perform EE. Ensure that adding the optimization still meets worst-case end-to-end latency. Explore other optimizations in order of increasing cost, resulting in a cascade. If the mobile device is powerful enough to run part of a cascaded tail, then moving that part to the device would achieve SC with EE. In the example above of a cascaded object detector and fine-grain classifer, the object detector could be moved from the tail to the head. On those frames where no object is detected, no transmission to the cloudlet is necessary. On frames where an object is detected, only a cropped region around it needs to be transmitted. It some cases, the move from tail to head may also involve using a different DNN. For example, it may be profitable to use a less accurate but cheaper object detector whose threshold is set to preserve recall at the cost of lower precision. This trades off device cycles and energy for network bandwidth and cloudlet cycles. In general, the design phase of enriching the head will be inherently iterative in character, because it depends critically on experimental measurements. What works well and what doesn't will depend critically on the use case, input data characteristics, device and network attributes, the specific optimizations, etc.

This design strategy achieves SC and EE, but involves no DNN surgery. It only relies on systems skills and basic ML skills that the typical mobile system designer is likely to possess. Mobile devices evolve quickly, so the third phase (enriching the head) will have to be experimentally re-validated on each new version of the device.

Although not explicitly mentioned above, energy efficiency is a key metric on mobile devices. In many cases, adding EE may improve both average case end-to-end latency and energy efficiency. In other cases, it may improve one but not the other. Adding SC may also impact on-device energy usage. The additional local compute will consume more energy, but if the transmitted result is compressed significantly there may be a net win. Only rigorous experimental exploration on the actual device, network and cloudlet can provide credible insights in this space.

5 Case Study

We offer a specific use case to exemplify the design strategy presented in §4. This case study examines Wearable cognitive assistance (WCA) applications, a well published genre of edge native applications, originally introduced in 2014 and since extensively studied [15, 3, 42].

WCA applications provide assistance with real world tasks to users who are wearing smart glasses. These applications consume large amounts of bandwidth while being latency sensitive, which makes them a compelling use case for edge computing. Here, we focus on an important subclass of WCA applications that provide step-by-step guidance through physical assembly tasks. These applications carry out a back and forth process with the user. First, the application gives the user an instruction for completing a step of the task. The user then completes this step, and then the application gives the user the next instruction. The camera on the glasses captures image frames of a user's progress through the task. The

Name	Description
Stirling	Assemble a heat engine from metal parts
Meccano	Build a model bike from metal parts
Toyplane	Build a model helicopter from 3D printed plastic parts
Sanitizer	Assemble a sanitizer for a smartphone from metal and plastic parts

Table 1: The WCA applications used for our study.

application processes these frames to determine when a task step has been completed.

We conducted experiments with four distinct WCA applications, which are described in Table 1. All of these applications required computer vision models, in order to determine the step of the task that is shown in an image frame. Training these computer vision models required capturing training images that depicted each step of the task. We labeled each training image with a bounding box, indicating the region of the image that will be modified once a user completes the next task step. We also assigned a class label to each image, which indicates the step of the task that is shown. In addition to the training sets, we collected test sets of images that we used to evaluate our models. The test images were manually assigned class labels, but we did not label them with bounding boxes.

5.1 Cloudlet Only Computation

The first step in our design strategy is developing a version of the application that runs entirely on the cloudlet. This eliminates the need for many of the performance engineering steps required to run some of these computations on the mobile device. In addition, we can avoid doing any mobile app development or worrying about network transmission until after we have a working prototype. We test our application with pre-recorded images and videos, at this stage.

These applications detect the presence of the partially-completed completed assembly after each step i. We can use image classification for this, but also object detection DNNs, particularly for cases where the object may not be dominant in the view. We trained and tested multiple classifiers and detectors for our applications. We also tried cascades of an object detector to find likely regions of interest (ROIs), followed by classification to confirm the presence of the object at those ROIs (similar to the approach in [12]). Table 2 shows the accuracy of the best models for each of our applications. The object detectors (EfficientDet [39] and Faster R-CNN [34]) outperformed the image classifiers (Resnet 50 [16] and Fast MPN-COV [23]), even though our test images had relatively clean backgrounds. However, the cascaded pairs worked best for all of our applications.

The cascade of Faster R-CNN with Fast MPN-COV performed best for all applications except Stirling, where Effi-cientDet-Lite2 with Resnet 50 worked better. This highlights

Application	Best model or cascade pair	Accuracy
Stirling	EfficientDet-Lite2 & Resnet 50	91.0%
Meccano	Faster R-CNN & Fast MPN-COV	84.5%
Toyplane	Faster R-CNN & Fast MPN-COV	92.9%
Sanitizer	Faster R-CNN & Fast MPN-COV	81.9%

Accuracy is the fraction of images correctly associated with the assembly "step."

Table 2: The model or cascade pair that achieved the highest accuracy for each application.

the need for WCA application developers to determine the models that work best for their specific application.

5.2 Adding a Thin Mobile Client

After successfully implementing the image processing on the cloudlet, we developed a mobile client that transmits JPEG images from the mobile device to the cloudlet. This client sends all images, without cropping them. We tested the application using a Glass Enterprise Edition 2 as the mobile device, and a cloudlet with an Intel Intel Xeon E5–2699 CPU and a GeForce GTX 1080 Ti GPU. The ping time between the mobile device and the cloudlet was under 0.5 ms.

The average end-to-end round trip time required to transmit a 1920 * 1080 pixel image to the cloudlet, process it using our Faster R-CNN and Fast MPN-COV pipeline, and receive the result back on the mobile device was 398 ms. This is well within the 600 ms (tight) or 2700 ms (loose) bounds of acceptable latency for such interactive applications as determined in a recent user study [3]. Energy consumption on the Google Glass is moderate, averaging 0.57 J per frame (measured using Android's PowerManager class) to transmit frames and wait for results. The main drawback to the thin client approach is the significant amount of bandwidth consumed transmitting every single frame to the cloudlet. The thin client approach also puts a significant strain on the cloudlet, requiring it to carry out expensive computations for each frame that the client sends.

5.3 Offload Shaping

Hu et al [18] suggested running cheap computations on the mobile device in order to avoid sending certain frames to the cloudlet. We first experimented with computing perceptual hash values to detect duplicate frames. When a new frame's perceptual hash value is similar to the last frame that was sent, the client does not send the new frame to the cloudlet. We tuned the perceptual hash threshold, so that we never increased the amount of time that a user has to wait for the application to give the next task instruction.

We also considered modifying the application so that it only sends frames to the cloudlet after a user has indicated that they have completed a step. The user makes a thumbs up gesture to indicate that they believe a step is complete. The application then begins transmitting frames to the cloudlet, and gives the user the next instruction if a step has in

	Time (ms)	Energy (J)
Perceptual Hash	94	0.40
Thumbs Up	102	0.51

Table 3: Cost of deciding whether a frame should be offloaded

	Stirling	Meccano	Toyplane	Sanitizer
Perc. Hash	76.8%	38.4%	41.2%	51.0%
Thumbs Up	46.3%	73.8%	54.9%	65.0%

Table 4: The percentage reduction in frames achieved by offload shaping.

fact been completed. After the next instruction is given, the application waits for a thumbs up gesture again. We detected thumbs up gestures using MediaPipe, which can be run on Google Glass [24]. The application can therefore detect thumbs up gestures locally, and then it only has to send frames to the cloudlet after the user has indicated that a step has been completed.

We recorded videos using a Google glass, of a user completing each of the tasks and making a thumbs up gesture in between each step. We then processed these videos using both strategies. The code for all of these measurements was run using a Google Glass. Profiling information for both strategies is listed in Table 3, and the reductions in frames sent to the cloudlet are listed in Table 4.

5.4 Running DNNs on the mobile device

Resnet 50 and EfficientDet can both be run on Android devices using TensorFlow Lite. This enables us to carry out some or all of the computation on the mobile device instead of the cloudlet. Inference time for a given DNN is going to be higher on a mobile device than a cloudlet. For a given latency bound, a cloudlet can run a more accurate DNN than a mobile device can. Therefore, we can run DNNs on the mobile device instead of the cloudlet, in order to reduce bandwidth and cloudlet resource consumption at the cost of accuracy.

We experimented with running all computations on the Google Glass, as well as splitting the cascade described in Section 5.1 across the mobile device and the cloudlet. In particular, we ran the object detector on the mobile device and the image classifier on the cloudlet. This allowed us to use Fast MPN-COV, which is not supported by PyTorch Mobile[1]. It also presented a huge bandwidth savings compared to sending every frame in full. Our application can crop images around the region of interest detected by the object detector, and just send this cropped region to the server. In addition, the application can avoid sending images that the detector does not find any instances of the object being assembled.

We examined all models and cascades of models that can be run on a Google Glass within [3]'s tight and loose latency bounds. Table 5 lists the best possible accuracy when running all computations on the cloudlet (mobile only) and running the object detector on the Google

[1]https://github.com/pytorch/pytorch/issues/22329

	Stirling	Meccano	Toyplane	Sanitizer
Mobile, tight	85.1%	75.2%	69.8%	87.9%
Mobile, loose	91.0%	75.2%	70.1%	89.1%
Split, tight	85.1%	82.0%	77.2%	87.9%
Split, loose	91.0%	82.0%	77.7%	89.1%

The split implementations achieve accuracy much closer to the best-case fully-offloaded versions from § 5.1 than the mobile-only implementations.

Table 5: Best accuracy achieved for mobile-only and split computing implementations

Stirling	Meccano	Toyplane	Sanitizer
79.2%	52.3%	86.8%	94.2%

Table 6: Bandwidth saved by sending only bounding boxes from EfficientDet-Lite0

Glass and the image classifier on the cloudlet (split). The bandwidth saved by transmitting detected crops instead of full images is presented in Table 6. The Google Glass consumed an average of 0.71 J to process one frame without offloading, and it consumed an average of 0.51 J to process one frame with the split implementation.

5.5 The Problem with DNN-only approaches

Matsubara et al [28] offer architectures to split the computation for a standalone object detector or a standalone image classifier, between a mobile device and a cloudlet. However, the output from the split object detector just contains the bounding box coordinates and class labels for detected objects. There is no way for the server to obtain a crop from the original image based on this output. In order to apply one of the above split architectures to our cascade of models we face two choices. One choice is for our application to send the entire image to the cloudlet along with the input to the tail of the object detector. The other choice is for it to send the bounding box coordinates back to the mobile device, and request the mobile device to send the cropped image in some form to run the classifier. The first choice eliminates all of the bandwidth savings that split computing offers. The second choice requires a second round trip to the mobile device, which increases end-to-end latency.

All of the strategies presented in this section offer significant savings in bandwidth, and none of them require specialized ML knowledge to implement. We suggest that developers follow our design strategy, and consider a broad range of EE and SC options, rather than focusing entirely on splitting computation using a single DNN.

6 Conclusion

While we commend and appreciate the recent interest in split computing in the ML community, we believe a purely ML approach is overly limiting. It leaves out many non-ML based optimization opportunities, and requires expertise in DNN architecture design. In contrast, we propose (and demonstrate with a case study) a design strategy tailored to ML practitioners and system integrators rather than DNN researchers. It leverages off-the-shelf DNNs, application-specific training, various non-DNN early discard techniques, but needs no new DNN architectures or network surgery. We believe this is the most practical approach to producing *edge-native* applications on mobile devices — i.e., those that are simultaneously compute-, bandwidth-, and latency-sensitive [36].

References

[1] Balan, R., Flinn, J., Satyanarayanan, M., Sinnamohideen, S., and Yang, H. The Case for Cyber Foraging. In *Proceedings of the 10th ACM SIGOPS European Workshop* (Saint-Emilion, France, September 2002).

[2] Balan, R., Gergle, D., Satyanarayanan, M., and Herbsleb, J. Simplifying Cyber Foraging for Mobile Devices. In *International Conference on Mobile Systems Applications and Services* (San Juan, Puerto Rico, June 2007).

[3] Chen, Z., Hu, W., Wang, J., Zhao, S., Amos, B., Wu, G., Ha, K., Elgazzar, K., Pillai, P., Klatzky, R., Siewiorek, D., and Satyanarayanan, M. An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance. In *Symposium on Edge Computing* (San Jose, California, 2017).

[4] Choi, H., and Bajic, I. V. Deep Feature Compression for Collaborative Object Detection. In *International Conference on Image Processing (ICIP)* (2018).

[5] Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., and Patti, A. CloneCloud: Elastic Execution between Mobile Device and Cloud. In *EuroSys 2011* (Salzburg, Switzerland, April 2011).

[6] 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 *International Conference on Mobile Systems, Applications, and Services* (San Francisco, CA, June 2010).

[7] Eshratifar, A. E., Esmaili, A., and Pedram, M. BottleNet: A Deep Learning Architecture for Intelligent Mobile Cloud Computing Services. In *Symposium on Low Power Electronics and Design (ISLPED)* (2019).

[8] Flinn, J. *Cyber Foraging: Bridging Mobile and Cloud Computing via Opportunistic Offload.* Morgan & Claypool Publishers, 2012.

[9] Flinn, J., Narayanan, D., and Satyanarayanan, M. Self-Tuned Remote Execution for Pervasive Computing. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems* (Schloss Elmau, Germany, May 2001).

[10] Flinn, J., Park, S., and Satyanarayanan, M. Balancing Performance, Energy Conservation and Application Quality in Pervasive Computing. In *Proceedings of the 22nd International Conference on Distributed Computing Systems* (Vienna, Austria, July 2002).

[11] Flinn, J., and Satyanarayanan, M. Energy-aware Adaptation for Mobile Applications. In *Symposium on Operating Systems and Principles* (Kiawah Island, SC, December 1999).

[12] Gebru, T., Krause, J., Wang, Y., Chen, D., Deng, J., and Fei-Fei, L. Fine-grained car detection for visual census estimation. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence* (2017), p. 45024508.

[13] Gordon, M. S., Jamshidi, D. A., Mahlke, S., Mao, Z. M., and Chen, X. COMET: Code Offload by Migrating Execution Transparently. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)* (Hollywood, CA, October 2012).

[14] Goyal, S., and Carter, J. A Lightweight Secure Cyber Foraging Infrastructure for Resource-constrained Devices. In *Proceedings of the 6th IEEE Workshop on Mobile Computing Systems and Applications* (2004).

[15] Ha, K., Chen, Z., Hu, W., Richter, W., Pillai, P., and Satyanarayanan, M. Towards Wearable Cognitive Assistance. In *International Conference on Mobile Systems, Applications, and Services* (Bretton Woods, NH, June 2014).

[16] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

[17] Hsu, K.-J., Bhardwaj, K., and Gavrilovska, A. Couper: DNN Model Slicing for Visual Analytics Containers at the Edge. In *Symposium on Edge Computing* (Arlington, VA, 2019).

[18] 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 *HotMobile* (Santa Fe, NM, 2015).

[19] Jeong, H.-J., Jeong, I., Lee, H.-J., and Moon, S.-M. Computation Offloading for Machine Learning Web Apps in the Edge Server Environment. In *International Conference on Distributed Computing Systems (ICDCS)* (2018).

[20] Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., and Tang, L. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In *ASPLOS '17: Architectural Support for Programming Languages and Operating Systems* (Xi'an, China, April 2017).

[21] Kozat, S. S., Venkatesan, R., and Mihçak, M. K. Robust perceptual image hashing via matrix invariants. In *ICIP'04: International Conference on Image Processing* (2004), vol. 5.

[22] Kristensen, M. D. Execution Plans for Cyber Foraging. In *MobMid '08: Workshop on Mobile Middleware* (Leuven, Belgium, 2008).

[23] Li, G., Liu, L., Wang, X., Dong, X., Zhao, P., and Feng, X. Auto-tuning Neural Network Quantization Framework for Collaborative Inference Between the Cloud and Edge. In *Int. Conference on Artiicial Neural Networks* (2018).

[24] Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., Zhang, F., Chang, C.-L., Yong, M., Lee, J., Chang, W.-T., Hua, W., Georg, M., and Grundmann, M. Mediapipe: A framework for perceiving and processing reality. In *Workshop on Computer Vision for AR/VR at IEEE Computer Vision and Pattern Recognition (CVPR)* (2019).

[25] Matsubara, Y., Baidya, S., Callegaro, D., Levorato, M., and Singh, S. Distilled Split Deep Neural Networks for Edge-Assisted Real-Time Systems. In *MobiCom Workshop on Hot Topics in Video Analytics and Intelligent Edges* (2019).

[26] Matsubara, Y., and Levorato, M. Split Computing for Complex Object Detectors: Challenges and Preliminary Results. In *Workshop on Embedded and Mobile Deep Learning (EMDL)* (September 2020).

[27] Matsubara, Y., Levorato, M., and Restuccia, F. Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges. *ACM Computing Surveys* (March 2022).

[28] Matsubara, Y., Yang, R., Levorato, M., and Mandt, S. Supervised compression for resource-constrained edge computing systems. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)* (January 2022), pp. 2685–2695.

[29] Monga, V., and Evans, B. Perceptual image hashing via feature points: performance evaluation and tradeoffs. *IEEE Transactions on Image Processing 15*, 11 (2006).

[30] Narayanan, D., and Satyanarayanan, M. Predictive Resource Management for Wearable Computing. In *International Conference on Mobile Systems, Applications, and Services* (San Francisco, CA, May 2003).

[31] Noble, B., Satyanarayanan, M., Narayanan, D., Tilton, J., Flinn, J., and Walker, K. Agile Application-Aware Adaptation for Mobility. In *Symposium on Operating Systems Principles* (Saint-Malo, France, October 1997).

[32] Ok, M., Seo, J.-W., and Park, M.-s. A Distributed Resource Furnishing to Offload Resource-Constrained Devices in Cyber Foraging Toward Pervasive Computing. In *Network-Based Information Systems*, T. Enokido, L. Barolli, and M. Takizawa, Eds., vol. 4658 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2007.

[33] Ra, M., Sheth, A., Mummert, L., Pillai, P., Wetherall, D., and Govindan, R. Odessa: Enabling Interactive Perception Applications on Mobile Devices. In *International Conference on Mobile Systems, Applications, and Services* (Bethesda, MD, June 2011).

[34] Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems* (2015).

[35] Satyanarayanan, M. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications 8*, 4 (2001).

[36] Satyanarayanan, M., Klas, G., Silva, M., and Mangiante, S. The Seminal Role of Edge-Native Applications. In *International Conference on Edge Computing (EDGE)* (Milan, Italy, 2019).

[37] Sobel, I., and Feldman, G. A 3x3 isotropic gradient operator for image processing. A talk at the Stanford Artificial Intelligence Project, 1968.

[38] Szeliski, R. *Computer Vision: Algorithms and Applications.* Springer, 2010.

[39] Tan, M., Pang, R., and Le, Q. V. Efficientdet: Scalable and efficient object detection. In *Computer Vision and Pattern Recognition (CVPR)* (June 2020).

[40] Teerapittayanon, S., McDanel, B., and Kung, H. BranchyNet: Fast inference via early exiting from deep neural networks. In *Proceedings of the 2016 23rd International Conference on Pattern Recognition (ICPR)* (2016).

[41] Wang, J., Feng, Z., Chen, Z., George, S., Bala, M., Pillai, P., Yang, S.-W., and Satyanarayanan, M. Bandwidth-efficient Live Video Analytics for Drones via Edge Computing. In *Symposium on Edge Computing (SEC)* (2018).

[42] Wang, J., Feng, Z., George, S., Iyengar, R., Pillai, P., and Satyanarayanan, M. Towards Scalable Edge-Native Applications. In *Symposium on Edge Computing (SEC)* (Washington, DC, November 2019).

[43] Ya-Yunn, S., and Flinn, J. Slingshot: Deploying Stateful Services in Wireless Hotspots. In *International Conference on Mobile systems, Applications, and Services* (2005).

[44] Zauner, C. *Implementation and benchmarking of perceptual image hash functions.* PhD thesis, University of Applied Sciences Hagenberg, Austria, 2010.