# Near-Sensor Distributed DNN Processing for Augmented and Virtual Reality

Reid Pinkham, *Member, IEEE*, Andrew Berkovich, *Member, IEEE*, and Zhengya Zhang, *Senior Member, IEEE*

***Abstract*— Untethered Augmented and Virtual Reality (AR/VR) devices are an emerging compute platform with unique opportunities and challenges. AR/VR devices use an array of sensors, including multiple cameras, to understand their surroundings and provide the user with an immersive experience. To deliver the functionality and performance, AR/VR devices rely on state-of-the-art algorithms including Deep Neural Networks (DNNs). These algorithms must operate in real time, and it presents a computational challenge for a mobile system. The emergence of on-sensor compute provides a possible solution to increase the processing capabilities of an AR/VR platform. In this work, we explore how to optimally map DNN models on an AR/VR compute platform that consists of an on-sensor processor and an edge processor to minimize energy and latency. We explore properties of popular DNN models, and the ideal network split locations, processor sizes, caching strategies, and the interactions between these design choices using a new Distributed Algorithm Simulator (DAS). Based on this study, we develop the basic principles on network split, parameter caching, and two-processor balancing to achieve near-optimal system designs. We show the addition of on-sensor processing to the existing Quest 2 VR platform can reduce MobileNetV3 inference energy by 64.6%. Finally, we demonstrate in a representative AR/VR platform how the minimum-energy configuration changes under the practical design constraints of memory size and silicon area, as well as the impact of future memory technologies.**

***Index Terms*— Computer architecture, Augmented Reality, Virtual Reality, systems architecture, image processing, neural networks.**

## I. Introduction

WEARABLE Augmented Reality and Virtual Reality (AR/VR) systems present a unique hardware architectural design challenge. They must process inputs from an array of sensors, including multiple cameras, microphones, and inertial measurement units. The data streams coming from these sensors must be analyzed in real time to provide users with low-latency human-machine interfaces and immersive visual experiences [1]. Furthermore, these devices operate under strict power budgets imposed by thermal and form factor constraints (e.g. battery size).

Manuscript received April 29, 2021; revised August 27, 2021; accepted October 4, 2021. Date of publication October 18, 2021; date of current version December 13, 2021. This work was supported by Facebook. This article was recommended by Guest Editor I. Partin-Vaisband. *(Corresponding author: Reid Pinkham.)*

Reid Pinkham and Zhengya Zhang are with the Department of Electrical and Computer Engineering, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: pinkhamr@umich.edu; zhengya@umich.edu).

Andrew Berkovich is with Facebook Reality Labs, Redmond, WA 98052 USA (e-mail: andrew.berkovich@fb.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JETCAS.2021.3121259.

Digital Object Identifier 10.1109/JETCAS.2021.3121259

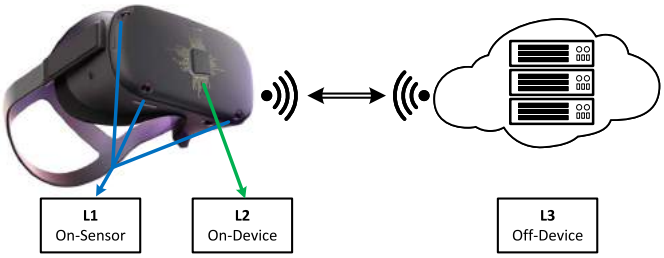



Fig. 1. A typical AR/VR system with three levels of compute representing on-sensor processing, mobile SoC-based edge processing, and PC- or cloud-backed off-device processing.

Future AR/VR hardware platforms may be composed of multiple levels of processing, possibly including an intelligent image sensor with a small, stacked on-sensor processor,[1] a nearby, off-sensor but on-device edge processor, and off-device processing that can be harnessed through wireless links on-demand to a nearby PC or remote cloud resources. This setup creates a distributed computing paradigm as shown in Fig. 1, where the processing can be spread across three layers.

One of the largest computational challenges for AR/VR devices are the image processing pipelines, specifically Deep Neural Network (DNN) processing [2]. Image sensors can capture a large amount of data in real time. For example, four VGA resolution sensors operating at 30fps produce >4MB/sec. Transferring and processing such large amounts of data is costly (in terms of system power and energy). Moving processing closer to the data source (e.g. image sensor) can yield improvements in overall energy efficiency of the system [3]. For this reason, the wearable AR/VR platform is an excellent candidate for on- or near-sensor processing.

Previous work has shown that it is feasible and advantageous to run small DNN workloads entirely on the sensor in the context of AR/VR devices [1], [4]. However, the amount of processing available on a sensor is limited by both the silicon area available (stacked processor die needs to match the footprint of the sensor die to some degree), as well as the sensor's energy budget. With larger workloads, any processing not performed on sensor is handled by an edge processor. The edge processor is also limited by the wearable device's

[1]Strictly speaking, a processor die stacked on a sensor to provide processing is near-sensor processing, but to differentiate it from other types of off-sensor processing, we call it on-sensor processing as it is stacked "on" the sensor.

energy envelope, but its size and energy are not as constrained as the on-sensor processor. On today's devices, this edge processor is often a mobile SoC like the Qualcomm Snapdragon 865 [5]. Ultimately, the remaining processing will fall back to an off-device PC or the cloud that are assumed not to be limited by energy, but data must be sent off-device via a wireless link, and extra latency will be incurred.

Given the opportunity to distribute processing to three layers, the problem arises as to how to divide the workload to best utilize each layer of processing in order to optimize for the device's performance and energy consumption. Many DNN accelerators have been developed in the past, but most work was focused on a single-processor system [6]–[17] which is not applicable to this work. Splitting DNN processing has been considered in the past [18]–[20], but they only address the split between the edge and the cloud, where the wireless communication dominates and the latency is at a much longer time scale than what is considered in this work. Architecture simulation infrastructure was developed for analyzing accelerator architectures [21], but it focused on how fine-grained architecture details impact single-processor performance, and does not consider system level performance.

In this work, we study the visual system and the associated DNN workloads which are the toughest bottlenecks in wearable AR/VR platforms [2]. The majority of visual system computation must happen in real time, and thus puts a heavy reliance on the first two levels of processing (on-sensor and edge processing) to reduce latency. Therefore we focus on the workload partition, and hardware design and optimization in the first two levels of processing. The contributions of this work are as follows.

- *Model partition*: we show the distinctive features of early and late layers of popular DNN models in terms of weight and activation sizes and reuse, and use it to guide the network partition and workload division between an on-sensor processor and an edge processor.
- *Workload split*: we introduce and analyze a method of splitting a DNN workload across an asymmetric multi-processor compute platform.
- *Simulation tool*: we create the Distributed Algorithm Simulator (DAS) to perform the design space exploration, extract insights on how to partition a network, and make system design choices.
- *Split processing design principles*: we develop the basic principles in making on-sensor processor and edge processor design choices to optimize the overall system performance and efficiency. The key insights are: 1) the on-sensor processing needs to meet a minimum capacity to justify an efficient split processing, 2) the optimal caching strategy should be determined based on maximizing a size-reuse product metric, and 3) balancing the latency between the on-sensor and the edge processor is essential for ensuring high system performance and efficiency. By following these basic principles, we show that near-optimal design solutions can be made without an exhaustive search.
- *AR/VR platform design constraints*: we demonstrate practical memory size and silicon area constraints for an AR/VR platform and how they affect the design choices. We show how future improvements in memory technology can enable even more possibilities for on-sensor processing in both small resource-constrained and high performance use cases.

In Section II we give an overview of the common DNN workloads for the AR/VR visual system. In Section III we detail the AR/VR system model used for this work. In Section IV we analyze the trade-offs in the system design, and develop a strategy to quickly find a low-energy design given system constraints. In Section V we explore some practical constraints of an on-sensor system and how they impact system design. Finally, in Section VI we provide details of DAS that we used to perform our analysis.

## II. Review of DNN Workload

A wearable AR/VR device must perform many visual tasks in real time, including video rendering, Simultaneous Location And Mapping (SLAM), scene segmentation, eye tracking, stereo depth estimation, and keypoint detection. This creates a very large burden on the device's computational system. All of these algorithms must run seamlessly in real time for the device to function properly.

Conventional AR/VR device designs [22]–[24] use an on-device edge processor to perform the real-time processing needed. However, the possibility of having some processing at the sensor level can alleviate some of this burden. Since an AR/VR device can have many image sensors, even a modest amount of processing per sensor can dramatically increase the computational capability of the system.

In addition to distributing processing, moving some computation to a processor that is directly stacked on a sensor can have the effect of reducing the data movement and possibly compressing data to reduce the energy required to move data to the edge processor that is further away. In a modern computation system, the energy cost associated with moving data off chip can be orders of magnitude more than processing that data [25]. Hence it is worthwhile to perform processing on the sensor to take advantage of the reduced data movement and a lower energy consumption.

Some applicable AR/VR workloads have a compressing property, such as object identification and object tracking. The input to these algorithms is a full image or video, while the output is a small set of data points. An example is DNN for object recognition, e.g., popular DNNs such as ResNet-50 [26], MobileNetV3 [27] and SqueezeNet [28]. It is common to use pre-trained versions of these networks as the front-end latent feature extractor to feed to more elaborate algorithms, such as the Detectron2 framework [29]. Such common use cases make DNNs good candidates for on-sensor processing.

However, due to size limitations of on-sensor processing, it is not feasible to run an entire DNN at the sensor level. Previous near-sensor processors from the tinyML community, such as SCAMP-5 [30], rely on extremely lightweight processing algorithms for successful operation. These sensors offer limited performance and flexibility, and are often limited to toy-workloads. Consequently, state-of-the-art DNN backbone

networks need to be split between an on-sensor processor and an edge processor. AR/VR devices offer a good representative application to evaluate this split-processing paradigm.

### A. DNN Model Characteristics

There are many common DNN architectures which have shown great success in the field of image recognition. AlexNet [31] demonstrated that a DNN of relatively few layers could be used to successfully perform complex image recognition. Even higher accuracy was achieved with much larger networks such as VGG16 [32] and deeper networks such as ResNet [26] and Inception [33]. These networks showed much higher accuracy than AlexNet, but all three have a larger parameter size, a higher computational load, or both, which prevent them from being widely adopted for mobile and power limited applications. Networks such as MobileNet [34] and SqueezeNet [28] showed that DNNs could be adapted for use in mobile and power limited applications with comparative accuracy to the previous larger DNNs. Since AlexNet, it has been shown that high precision computation is not needed for inference. With judicious quantization and retraining, DNNs can use INT8 for inference computation and maintain all or nearly all of the original FP32 performance [35].

All of these successful DNNs share some common architectural features. In general, the number of channels in each layer increases throughout the network, while the height and width of the activations decrease. Additionally, many networks use one or more fully connected layers at the end to produce the final category-by-category predictions. These common characteristics are helpful for understanding the challenges and gaining design insights. In the next three sections, we discuss some of these common characteristics of the CNNs in greater detail and how they pertain to a hardware architecture design. We use ResNet-50 [26] and MobileNetV3 [27] as example DNN workloads.

### B. Activation Size and Network Compression

A general trend for visual system DNNs is that they compress the intermediate representation size throughout the network. This can be seen as distilling the redundant information contained in the image at each layer until the DNN has an understanding of the scene to make a prediction.

Fig. 2 shows how initially the size of the intermediate representation (activation) is expanded from the input image size before beginning to reduce in size. The consistent spiking nature of the plot results from the modular block composition of the network, with each block containing multiple convolutional layers. For both networks, each block expands the representation before compressing it again.

In a two-processor (an on-sensor processor and an on-device edge processor) system, the communication between the first and second parts of the system can consume a significant amount of energy. A relevant point to look at in these networks is when the intermediate activation size is compressed to a point which is smaller than the original image, referred to as the *compression point*. Since the energy cost for transmitting the intermediate activation is proportional to its size, the compression point shows the minimum amount of computation needed before energy can be saved for transmission. On systems with extremely high communication costs, split points at or below the dotted line in Fig. 2 should be considered.

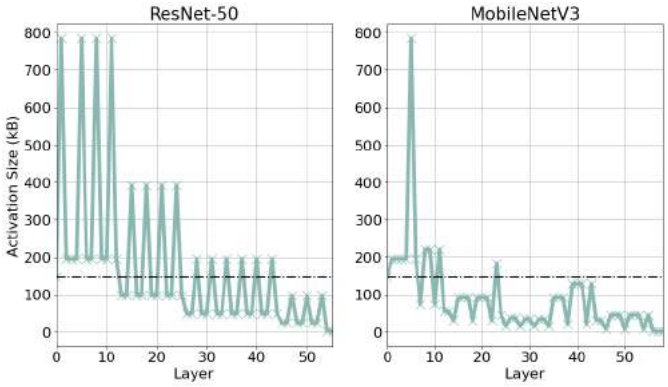



Fig. 2. Intermediate activation size at each layer of the ResNet-50 and MobileNetV3 networks. Dotted line shows the original size of the input.

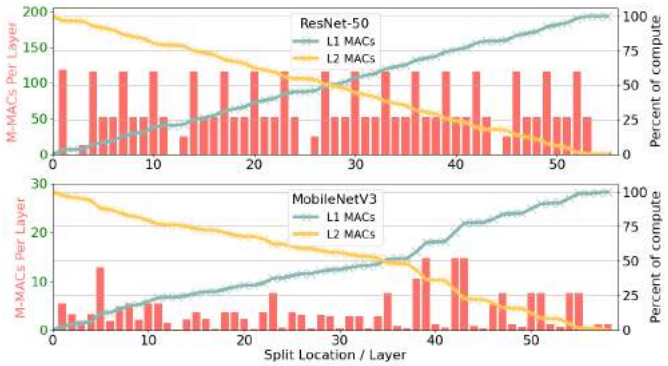



Fig. 3. Computational cost at each layer (bars, M-MACs) and proportion of compute for each processor if split at the particular layer of ResNet-50 and MobileNetV3 networks.

### C. Layer Computation Cost

A trend in modern DNN architectures is an increase in both the amount of computation and the number of layers. The amount of computation per layer is dictated by the size of the input, the size of the filter, and the number of filters. In general, the size of the activations decreases throughout the network (Fig. 2), but the number of filters at each layer increases. Both of these features lead to a comparatively consistent amount of computation at each layer.

Fig. 3 shows the amount of computation across each layer of ResNet-50 and MobileNetV3. In a system where the computation is split across two processors, L1 (on-sensor processor) and L2 (on-device edge processor), the proportion of computation each performs is dependent on the layer at which the split is made. We can see in Fig. 3 that the proportion changes close to linearly with the layer number. In other words, if the computation is shared equally between the two processors, the computation should be split at the middle layer.

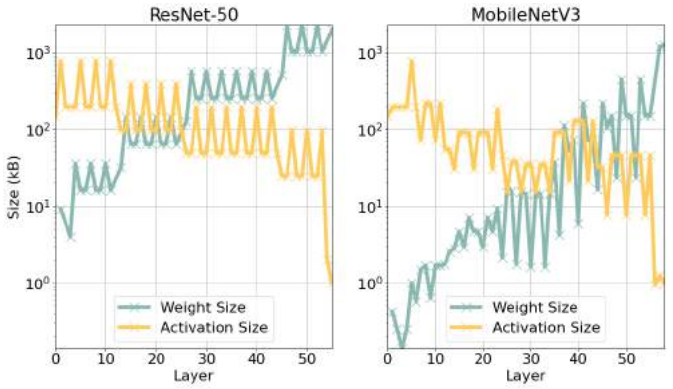



Fig. 4. Weight and activation size at each layer of the ResNet-50 and MobileNetV3 networks.

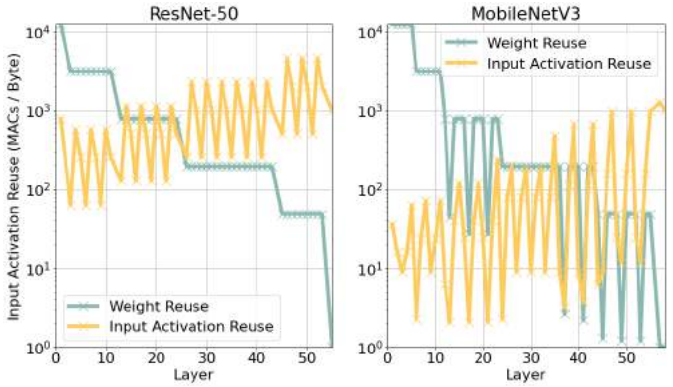



Fig. 5. Reuse of weights and activations at each layer of the ResNet-50 and MobileNetV3 networks. Units are MACs per Byte.

### D. Weight Size and Weight Reuse

In general, the size of the weights at each layer increases throughout the network. This is the opposite trend as the activations. Both the activation and the weight size at each layer are shown in Fig. 4. By the end of the network, there are an order of magnitude more weights than activations.

Since the amount of computation at each layer is nearly constant, as the number of weights increases in late layers of the network, the weights are reused less. Fig. 5 shows the reuse factor at each layer for the activations and the weights. In early layers of the network, the weights are reused more since the filters must be scanned over a large input image. In late layers of the network, the activations are reused more since more filters are applied to the activations.

From a system designer's perspective, this shift in reuse creates three different situations. If a processor is designed for the first half of the network, the highest reuse of cached data is achieved by caching weights. Similarly, a processor designed for the second half of the network has the highest cache reuse by caching activations. However, there is no clear method for a processor designed for the entire network where the average reuse is similar between weights and activations. The better strategy depends on the exact network structure. However, choosing one caching strategy will lead to inefficiencies in processing either the early or late layers of the network.

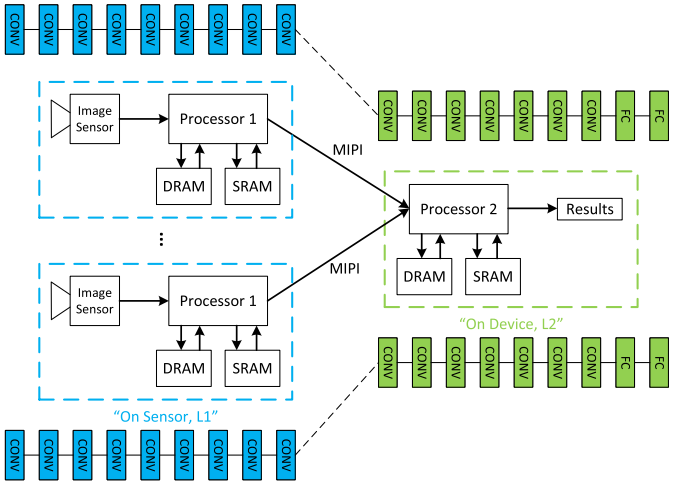



Fig. 6. The system model for design exploration. The number of L1 processors can be varied from 1 to $n$. A DNN is split between the two levels of processors.

A split processing system can increase the overall efficiency by using two different caching schemes for the early and late layers.

## III. AR/VR Compute Platform Model

We use an abstract model of a typical AR/VR compute system, similar to that in [1], to evaluate performance and energy. The model consists of the L1 and L2 levels of the processing stack (Fig. 1) and is detailed in Fig. 6.

The model consists of $n$ L1 processors representing the stacked on-sensor processing available at each image sensor. The L2 processor is a larger on-device mobile processor. Each processor in the system can have access to both an on-chip SRAM cache, as well as an off-chip DRAM system. The interface between processors matches the common MIPI interface [36] which is used by many popular camera modules.

We define five different processor types which are commonly used for DNN processing. The size of each processor is determined by the number of Functional Units (FUs) it has. Each FU is capable of performing a single Multiply-Accumulate (MAC) operation.

The first two types reflect the general-purpose processors: CPU and GPU. The CPU is modeled as a series of vector multipliers with a DRAM-backed cache interface. The GPU processor type uses the streaming friendly memory hierarchy common in GPUs. A set of FUs share a local SRAM scratch-pad, and this is backed by a DRAM interface. Most operations require fetching data from the DRAM.

The final three processor types reflect three common caching techniques in DNN accelerators. Each type splits the on-chip SRAM between FUs. The weights and activations are treated separately. The first type of processor is the Cache-Weight (CW) processor which keeps weights in SRAM and relies on the external DRAM to store activations. The second type is Cache-Activation (CA) which stores activations in SRAM and weights in DRAM. The final type (CA+CW) keeps both activations and weights in SRAM. The CA+CW

TABLE I
ENERGY AND POWER OF THE PROCESSOR AND SYSTEM COMPONENTS

| Component | Cost/Operation | |
|---|---|---|
| DRAM Energy | 41.7 nJ/Byte (rd) | 39.4 nJ/Byte (wr) |
| SRAM - Dynamic Energy | 1 - 5 pJ/Byte | |
| SRAM - Static Power | 2 nW/Byte | |
| FU Energy | 47.6 fJ/MAC | |
| Sensor-L1 Interface Energy | 0.24 nJ/Byte | |
| L1-L2 Interface Energy | 0.80 nJ/Byte | |

type is the only one of the five types which does not use a DRAM interface.

The CA+CW processor type is often the only feasible type for on-sensor processing because it does not require a DRAM interface. However, with advanced chiplet-based [37]–[39] and stacked-chip [40], [41] designs putting large DRAM, HBM, or SRAM arrays on or next to the sensor, processor types other than CA+CW will be feasible for on-sensor processing in the near future. Therefore, we consider all these types.

The efficiency of each processor type is variable. It depends on many factors such as the amount of workload, the ratio of weights to computation, the available memory bandwidth, and the size of the processor. We assume that each processor in our system is optimized for high utilization, and has associated efficiencies for each type of DNN operation.

Across all five types of processors, the cost of each memory and MAC operation is standardized. See Table I for the specific values. The DRAM access is based on a common DDR4 DRAM chip from Micron [42]. The SRAM values are based on trends extracted from the a commercial 28nm SRAM compiler. The cost of the read/write operations depends on the size of the SRAM array. Additionally, leakage power is modeled proportional to the size of the SRAM array. The energy cost of a MAC operation is based on an 8-bit MAC synthesized in the same 28nm technology.

The energy consumption of the sensor and inter-processor interface is on a per-byte basis. The image sensor uses a MIPI interface which reflects what is commonly available in modern camera modules. The inter-processor interface is modeled to have the same bandwidth as the sensor interface. There is a wide range of possible interfaces a system designer could choose from. We use a conservative estimate for the per-byte energy consumption to reflect a realistic system given the bandwidth requirements. The energy per byte for each interface is shown in Table I.

## IV. SYSTEM DESIGN CONSIDERATIONS

In the previous section we discussed how different features of the network may influence system design choices. Here, we focus on how, given a network and workload distribution, design choices impact system energy and performance. We use the system outlined in Section III with one L1 processor and one L2 processor. The results were obtained using DAS which is detailed in Section VI.

### A. Minimum Requirements

Not all possible hardware system configurations can support the expected DNN model for AR/VR processing. For a given DNN model, some minimum conditions must be met.

- The system must have enough compute capacity (MACs/s) to process the model in real time.
- The system must have enough memory, commonly SRAM or DRAM, to store the weights and activations to support processing.
- The system must have enough bandwidth between processors to transfer intermediate activations to support real-time processing.

Checking these minimum requirements is a straightforward way to rule out some possible hardware system configurations. However, these requirements alone are not enough to ensure the system will run in real time. For example, some DNN layers will have a low processor utilization because of memory bandwidth limitations. Because these situations are highly dependent on run time conditions, a simulation tool like DAS is needed to ensure they can be suitable candidates.

### B. Single Variable Impact

First, we consider how a single aspect of the design influences system energy and performance while keeping all the other aspects of the design fixed. We focus on single variables first as a way of simplifying analysis before exploring the more complex interplays between various aspects of the design in the next subsection. We consider ResNet-50 and MobileNetV3 in this study. The network split location is set to layer 28 and layer 35 for ResNet-50 and MobileNetV3, respectively, which corresponds to an equal division of computation between L1 and L2. We will allow the split point to move in later studies.

*1) L1 Processor Caching Strategy:* In a two-processor system for AR/VR, the L1 processor is usually size-limited and the L2 processor is generally larger than the L1 processor. Therefore, we intentionally limit the L1 processor to a small size of 64 FUs, and set the L2 processor to 2,048 FUs based on the current state-of-the-art Qualcomm Snapdragon 865 mobile SoC [5]. For analysis purposes, we assume the L2 processor is equipped with enough SRAM to cache both weights and activations (CA+CW): 20MB and 5.2MB for ResNet-50 and MobileNetV3, respectively. The type of L1 processor (CPU, GPU or accelerator of different caching strategies) is varied in this study. The inference energy and latency of each configuration are shown in Fig. 7.

*a) Latency analysis:* Across the different processor types, the L1 processor latency is nearly constant. The performance of the L1 processor is limited by the size instead of the interface bandwidths. The energy per inference varies with the type of the L1 processor. There is an increase from the CPU to GPU energy which is reflective of the increase in DRAM accesses by the GPU.

*b) Energy analysis:* For both networks, there is a clear energy advantage in caching activations (CA) over caching weights (CW). This result runs counter to the common reasoning from Section II that caching weights in early layers is more advantageous since in early layers there is a higher reuse of weights. However, the decrease in DRAM energy indicates that there are overall fewer DRAM accesses when caching activations, despite the lower reuse. To understand

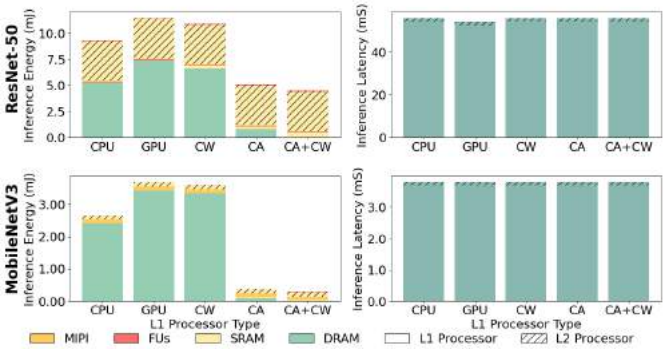



Fig. 7. Energy and latency of ResNet-50 and MobileNetV3 inference for a 2-processor system with various L1 processor types.

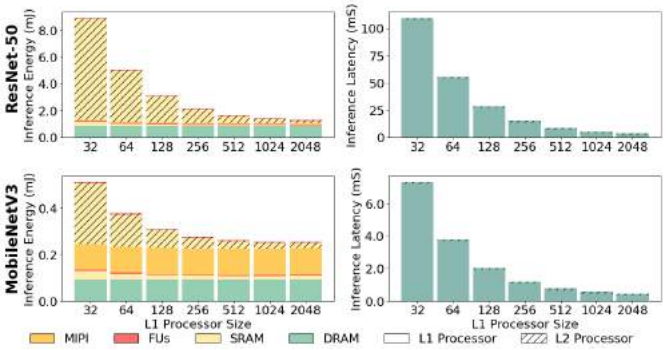



Fig. 8. Energy and latency of ResNet-50 and MobileNetV3 inference for a 2-processor system with various L1 processor sizes.

why, we refer back to Fig. 4. The activations are one to two orders of magnitude larger than the weights in the first portion of both networks. As shown in Fig. 7, moving from CW to CA results in significant DRAM bandwidth savings, and thus an energy reduction.

*c) Summary:* It is important to not only consider the reuse of what is being cached, but also the size and the associated opportunity cost with what is being cached. The total access, i.e., the product of reuse and data size, matters. In the case of ResNet-50, even for the early layers, the difference in size between activations and weights is significant and the difference in reuse is relatively less. The savings are amplified with MobileNetV3 where the early layers have over two orders of magnitude difference in weight and activation size. Therefore it is advantageous to cache activations to save energy. However, CA requires a larger SRAM and thus it involves a trade-off between the size of SRAM and the number of FUs that could be fit on a limited-size L1 processor.

*2) L1 Processor Size:* We still fix the size of the L2 processor to 2,048 FUs and use CA+CW. Based on the analysis from the previous section, we fix the L1 processor to use CA to reduce DRAM bandwidth. The inference energy and latency of each L1 processor size are shown in Fig. 8.

*a) Latency analysis:* The system latency decreases proportionally to the size of the L1 processor. At equal L1 and L2 processor sizes, the latency from each processor is roughly the same since we are using an equal division of work.

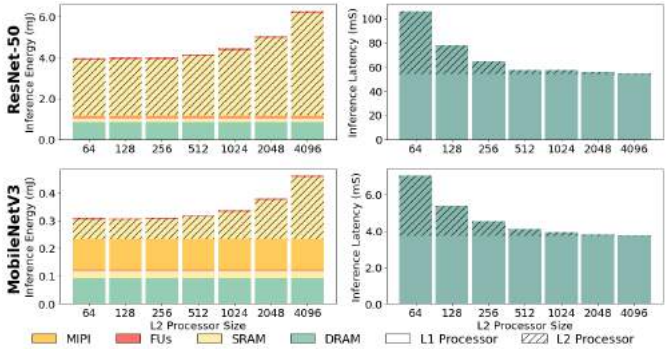



Fig. 9. Energy and latency of ResNet-50 and MobileNetV3 inference for a 2-processor system with various L2 processor sizes.

*b) Energy analysis:* Though ResNet-50 and MobileNetV3 share similar trends in inference energy, the proportion of energy devoted to communication is much higher in MobileNetV3, which is due to the much lower compute workload by MobileNetV3.

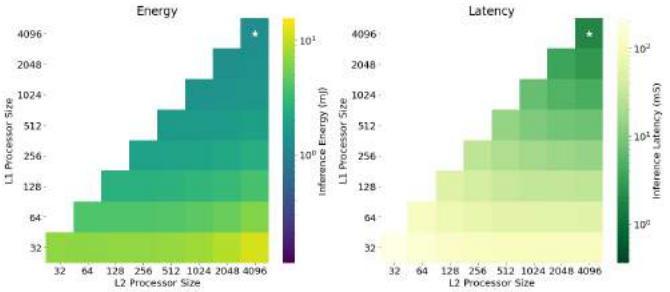
A less obvious but important result is that the total inference energy also decreases with larger L1 processor sizes. For the L1 processor, the dominant energy component is the DRAM access. Since the L1 uses CA, all access to weights is done through the DRAM interface. The total energy consumed by the L1 processor is nearly constant across L1 processor sizes due to the dominant DRAM access energy and the fixed amount of weights that needs to be accessed from DRAM. On the other hand, the L2 processor uses CA+CW, and its dominant energy component is SRAM access and leakage. A small L1 processor stretches the latency of the L2 processor, resulting in significant SRAM leakage. As the L1 processor size increases to keep its processing latency more balanced with the L2 processor, the utilization of the L2 processor improves and the leakage energy wasted by the large idle SRAMs is decreased.

*c) Summary:* It is essential to keep the latency balanced between the L1 and L2 processor to keep both the latency and energy low. Balancing the latency requires matching the L1 and L2 processor workload to their sizes to maximize utilization and reduce the inefficiency due to static SRAM energy. Since the L1 processor is often limited in size, the split location of the network needs to be adjusted accordingly.

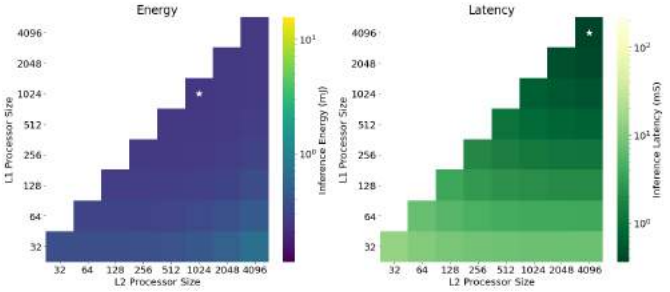
*3) L2 Processor Size:* Now consider the size of the L2 processor for a fixed L1 processor size. We fix the L1 processor to 64 FUs[2] and use CA. The L2 processor sues CA+CW. The inference energy and latency for each L2 processor size are shown in Fig. 9.

*a) Latency and energy analysis:* The latency decreases with a larger L2 processor size. However, the minimum energy is at 128 FUs for the L2 processor, beyond which the energy increases. As the size of the L2 processor grows while the L1 processor is fixed in size, the under-utilization of the L2 processor becomes a problem again, causing increasing SRAM leakage. To support a large L2 processor, the on-chip

[2]Even though the L1 processor can reasonably reach sizes above 64 FUs, we limit it here to better show the L2 performance trends.

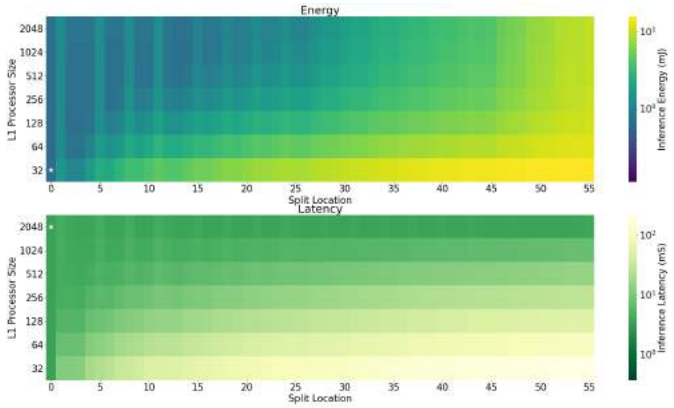



(a) ResNet-50

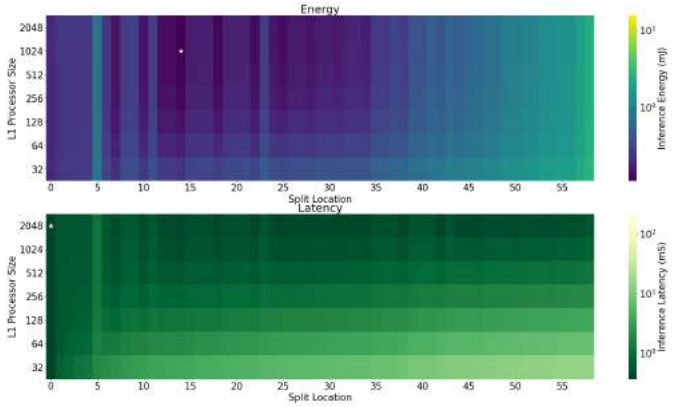



(b) MobileNetV3

Fig. 10. Energy and latency of inference for a 2-processor system with various L1/L2 processor size pairs.




(a) ResNet-50




(b) MobileNetV3

Fig. 11. Energy and latency for a 2-processor system with various L1 size and split location pairs.

SRAM needs to be fragmented to meet the data bandwidth required by more FUs. In all cases shown in Fig. 9, the total size of the on-chip SRAM remains constant across the configurations, but the number of banks increases to support more FUs, which is more costly in terms of both leakage and access.

*b) Summary:* This insight ties back into the importance of balancing the latency between the L1 and L2 processor.

### C. Two Variable Impact

After considering the impact of only changing single variables, we will examine the interplay of two variables and the impact on the system energy and performance.

*1) Balancing of L1 and L2 Processor Latency:* We revisit the example of L1 and L2 processor size. We use the same splitting location for both networks as before which balances L1/L2 computation, and assume that the L1 processor uses CA and the L2 processors uses CA+CW. Fig. 10 shows the inference energy and latency for each of the processor size pairs. We cap the L1 processor at the size of the L2 processor. The star on the plots indicates the global minimum.

*a) Latency analysis:* Increasing the size of either processor will lead to a lower latency with the lowest latency achieved when both processors have their largest size. In order for a network to run in real-time, we require the system to process at least 30 frames per second (FPS). From Fig. 10a we can see that a very large L1 processor is needed to meet the threshold of 33 ms per frame. However, Fig. 10b shows that even the smallest L1/L2 processor combination can finish the MobileNetV3 computation in real-time, albeit inefficiently.

*b) Energy analysis:* For each size of the L2 processor, the energy decreases as the size of the L1 processor increases to balance the L1 and L2 processing latency. For each size of L1 processor, a much larger L2 processor generally leads to a higher energy. For both networks, the minimum energy configuration is when both processors are of equal size, which matches the split near the middle of the network.

*c) Summary:* The above trends are in line with the earlier conclusions and highlight the need to choose the L1 and L2 processor size to match their workload to balance their latency.

*2) L1 Processor Size and Network Split Location:* One key message from the previous analyses is to choose a processor size suitable for its workload. The workload for each processor is adjusted by choosing the network split location. For this study, we fix the L2 processor at 2,048 FUs, and assume that the L1 processor uses CA and the L2 processor uses CA+CW. The inference energy and latency at each splitting location for ResNet-50 and MobileNetV3 are shown in Fig. 11a and Fig. 11b, respectively. The star on each plot indicates the global minimum.

*a) Latency analysis:* The latency plots for both ResNet-50 and MobileNetV3 indicate the global optimal network splitting location is at layer 0. For small L1 processors, it is generally not worthwhile to assign any L1 workload and expect the processing latency to be balanced with the L2 processor to achieve full utilization. As the L1 processor size increases, there are more possible split locations at or beyond the compression point.

*b) Energy analysis:* The energy plot differs between the ResNet-50 and MobileNetV3. For ResNet-50, the global minimum-energy configuration is to skip L1 processing entirely. As the split location is moved later in the network, the L1 processor needs to fetch an increasing amount of weights from DRAM, which makes splitting at later layers unappealing. However, with a large L1 processor, multiple nonzero split locations early in the network are very close to the global minimum energy and latency.

Different from ResNet-50, in MobileNetV3 there are many suitable split locations throughout the early network layers. The difference is attributed to the network size and structure. MobileNetV3 is a smaller network, requiring 74% less storage and 94% less computation than ResNet-50. The reduced storage and computation for MobileNetV3 magnifies the importance of the communication energy between the L1 and L2 processor. To reduce the communication energy, it is beneficial to perform more L1 processing to decrease the activation size before moving to the L2 processor. For MobileNetV3, the minimum-energy split location varies with the L1 processor size. For a very small L1 processor, the optimal split location is in the first few layers, i.e., at or near the compression point. With a larger L1 processor, the minimum-energy split location moves to a later layer. A larger L1 processor can handle more workload while still balancing the L1 and L2 processing latency.

*c) Summary:* The above results highlight a key difference between larger DNNs like ResNet-50 and smaller mobile-oriented DNNs like MobileNetV3. Smaller networks can best leverage the two-processor system to reduce their overall energy consumption. Large networks may not see an advantage of splitting computation. For networks even smaller than MobileNetV3, it is possible that the lowest energy configuration is to shift computation entirely to the L1 processor. In general, smaller networks shifts the minimum energy split location to later layers.

### D. General Design Principles

Each of the discussed trade-offs provides a trend exhibited by a typical DNN workload on a multi-processor edge computing system. From these trends, three principles can be summed up for designing a close to optimal split-processing system in terms of both inference energy and latency without an exhaustive search.

*Principle 1: A minimum on-sensor (L1) processor size is required for efficient split processing.* The L1 processor must have the compute capacity to process the layers up to the crossover point where the activation size falls below the input image size. This represents the first point in the network where the communication costs from L1 to L2 are reduced. That is, the following equation must be satisfied where $p_{comp}$ is the proportion of compute before the compression point, $R_{Lx}$ is the average rate of processing for the corresponding processor in OPS/s, and $MACs$ is the total number of MAC operations for the network. $R_{Lx}$ can be estimated from processor profiling and accounts for any utilization inefficiencies.

$$MACs \cdot \left( \frac{p_{comp}}{R_{L1}} + \frac{1 - p_{comp}}{R_{L2}} \right) < 33ms \tag{1}$$

*Principle 2: Allocate on-sensor processor cache to maximize size-reuse product.* Often in general-purpose processors the cached data has the highest reuse which reduces the number of times the individual data is fetched from off-chip storage. However, this does not minimize total off-chip bandwidth. The energy consumed by the external interfaces is significant for on-sensor processors. Caching items with the highest size-reuse product minimizes the total amount of data fetched from external storage. In our case, this is done by choosing an architecture with an appropriate caching scheme (CA, CW, etc.).

*Principle 3: Match workload split to on-sensor (L1) and edge (L2) processor size.* To achieve efficient inference, it is important to match the workload partitioning to the L1 and L2 compute capacity. In general, this translates to balancing latency between both processors by choosing an appropriately located split location. This leads to high processor utilization with more efficient use of system resources and consequently more efficient processing. The following equation will generally hold true for efficient distributions of work.

$$\frac{MACs_{L1}}{R_{L1}} \approx \frac{MACs_{L2}}{R_{L2}} \tag{2}$$

In addition to these three principles, we can determine if the addition of on-sensor processing will reduce inference energy for a particular algorithm analytically. Without loss of generality, we assume both processors are using the same manufacturing node and therefore the energy of each unit of computation and storage is equal at both the L1 and L2 processors. In this case, the predominant differences in inference energy will be from the change in communication energy and the change in SRAM static energy consumption. Given that the split location is after the compression point, we expect the energy from the L1-L2 communication to decrease. However, adding the L1 processor will increase the total SRAM capacity of the system and possibly the length of computation which will increase the total energy lost due to SRAM leakage.

We can estimate the change in communication cost as $\Delta E_{com}$ in the following equation. Here, the energy per byte of the communication is $E_{com/B}$, the compression factor at the split point is $CF$, and the size of the input is $B_{input}$. If we are compressing ($CF < 1$), this is expected to be negative.

$$\Delta E_{com} = E_{com/B} \cdot B_{input} \cdot (CF - 1) \tag{3}$$

The change in SRAM energy, $\Delta E_{sram}$, is dependent on both the compute capacity of the L1 and L2 processors in Ops/s, as well as the previous and new sizes of the SRAM. If we assume that both processors will use the most efficient CA+CW approach, the sizes of the SRAMs will need to be large enough to store the weights and largest activations for the assigned layers from the network. With this, the following equation estimates the change in inference energy when adding

TABLE II
COMPARISON OF BASELINE, HAND-TUNED DESIGNS AND THE GLOBAL MINIMUM-ENERGY DESIGN FOR RESNET-50 AND MOBILENETV3

| Network | Style | Configuration | | | | Split Location | Latency (ms) | | | Energy (mJ) |
|---|---|---|---|---|---|---|---|---|---|---|
| | | L1 | | L2 | | | L1 | L2 | Total | |
| | | Size | Type | Size | Type | | | | | |
| **ResNet-50** | Principle 1 | 256 | CW | 4096 | CA+CW | 13 | 6.156 | 1.281 | 7.437 | 5.191 |
| | Principle 1+2 | 256 | CA | 4096 | CA+CW | 13 | 6.097 | 1.281 | 7.378 | 1.060 |
| | Principle 1+2+3 | 256 | CA+CW | 4096 | CA+CW | 3 | 1.276 | 1.589 | 2.865 | 0.586 |
| | Global Min-Energy | - | - | 4096 | CA+CW | 0 | 0.151 | 1.652 | **1.802** | **0.545** |
| **MobileNetV3** | Principle 1 | 256 | CW | 4096 | CA+CW | 7 | 0.466 | 0.090 | 0.556 | 1.749 |
| | Principle 1+2 | 256 | CA | 4096 | CA+CW | 7 | 0.449 | 0.090 | **0.539** | 0.157 |
| | Principle 1+2+3 | 256 | CA | 1024 | CA+CW | 7 | 0.449 | 0.357 | 0.806 | 0.143 |
| | Global Min-Energy | 256 | CA | 512 | CA+CW | 14 | 0.599 | 0.634 | 1.233 | **0.117** |

L1 compute. Here, $t_{inf}$ is the inference time and $B_{sram}$ is the total size of the system's SRAM with the $\Delta$ indicating the change after adding the L1 compute. It is expected that $\Delta B_{sram}$ and $\Delta t_{inf}$ are both positive.

$$\Delta E_{sram} = t_{inf} \cdot \Delta B_{sram} + \Delta t_{inf} \cdot (B_{sram} - \Delta B_{sram}) \quad (4)$$

Given these two quantities, the energy saved in communication must outweigh the energy gained from the additional SRAM and compute time if we expect to see an energy saving from splitting the computation:

$$\Delta E_{com} + \Delta E_{sram} < 0 \quad (5)$$

### *E. Case Study*

To highlight the complexities of the split-processing system design and how the three principles can be applied, consider the task of designing a system with the following constraints that are reflective of a practical AR/VR platform: 1) the L1 processor is limited in size to 256 FUs; the L2 processor is limited to 4,096 FUs; and the L1 processor size cannot exceed the L2 processor size; 2) the DNN model must run at least in real time at 30 FPS, and a frame is sized $224 \times 224 \times 3$. The goal is to find the optimal network split location, and the appropriate size and type of the L1 and L2 processor. We separately consider a system to run ResNet-50 and another to run MobileNetV3.

We experimented with three designs for each network. We begin with a baseline design which uses the conventional approach choosing the largest processor size affordable and caching schemes to maximize reuse. The initial split location is at the compression point. Successively more principles are employed to find the optimal design. We include the global minimum-energy configuration that meets the real-time requirement found by DAS. Results of this case study are summarized in Table II.

For ResNet-50, none of the designs which support splitting are able to reduce the energy consumption of the system. Using Equations 3, 4, 5, adding an L1 processor with 256 PEs and splitting at the compression point is expected to increase inference energy by $312\mu J$. The third hand design using all three principles is able to improve on this by balancing latency in the two processors. However, the overall energy remains higher than performing the compute exclusively on the L2 processor. In this situation, a larger L1 processor would be required to see an energy savings with the L1 processor.

In the case of MobileNetV3, each successive principle reduces the inference energy of the system. In this case, the size of the L2 processor is decreased with principle 3 to balance the compute latencies. For both networks, considering even the first two design principles reduces the inference energy substantially. When applying all three principles, the design is nearly as efficient as the global minimum-energy point, only 7.5% and 22% more energy for ResNet-50 and MobileNetV3, respectively.

TABLE III
QUEST 2 PER-INFERENCE ENERGY IMPROVEMENT WITH L1 PROCESSING

| Network | # Sensors | Original | With L1 | Savings |
|---|---|---|---|---|
| ResNet-50 | 4 | 74.0 mJ | 64.9 mJ | 12.3% |
| MobileNetV3 | 4 | 26.9 mJ | 9.53 mJ | 64.6% |

### *F. Oculus Quest 2 Optimization*

We demonstrate how L1 processing and splitting of DNN computation can be used to improve inference energy for an existing VR computing system. The Quest 2 [43] uses a Snapdragon XR2 platform with an Adreno 650 GPU which can perform 4,096 FP16 calculations per cycle. We model this performance with DAS using an L2 processor with 4,096 FUs and a GPU memory hierarchy. We consider the case of augmenting the Quest 2 with an L1 processor at each camera sensor. To maintain a reasonable comparison, we assume the L1 processors have 1MB of SRAM and 32 FUs, matching realistic sensor sizes discussed in Section V. Table III shows that with the addition of L1 processing on the Quest 2, the per-inference energy savings are 12.3% for ResNet-50 and 64.6% for MobileNetV3.

## V. AR/VR SYSTEM CONSTRAINTS AND OPTIMIZATIONS

Unlike in the previous section which uses the number of FUs as a proxy for the size of the L1 processor, in this section we will impose realistic platform and silicon area constraints. Here we present the results from DAS to capture and understand trends across the entire design space. We found that hand designs following the design principles from the previous section still result in near-optimal designs under these constraints.

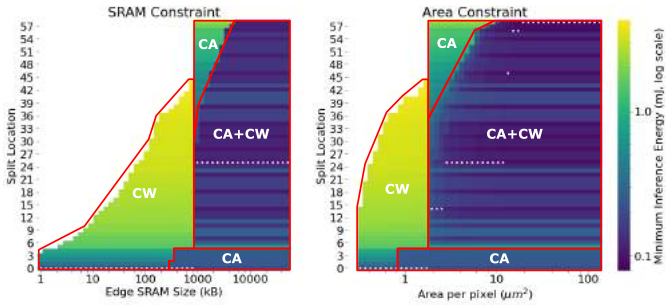



Fig. 12. Baseline system energy design space running MobileNetV3 with SRAM and area constraints. Along the y-axis is each possible split location. Each location on the graph represents the lowest energy achievable for each configuration (Split location + SRAM/Area pair). Lowest-energy caching scheme for each region is annotated on plot. White markers indicate the lowest-energy split location given the SRAM capacity/pixel area upper limit. Blank areas represent configurations which either do not run in real time (30 FPS), or are not possible given the L1 SRAM/area constraint.

The first constraint applies to the sensor platform where the SRAM for the L1 processor can be separated from the compute to create a three-level (sensor-memory-compute) stacked platform [40]. In this platform, we explore how the size of the SRAM die limits the design of the L1 processor. The second constraint is the silicon area available for the L1 processor. For a three-level stacked platform, the area of the L1 processor cannot exceed the size of the image sensor. Constraining the area also implicitly limits the power consumption of the L1 processor. Both of these constraints reflect a design which could be realistically accommodated in current AR/VR platforms. In such platforms, the area of the image sensor is limited to reduce the overall size of the corresponding optics on the device. Baseline results for both the SRAM and area design spaces for running MobileNetV3 are shown in Fig. 12.

### A. L1 SRAM Size Constraint

We vary the available amount of L1 SRAM from 1 KB to 64 MB in logarithmic steps. The general trend of the plot and the optimal network split locations are similar. Each L1 processor must have at least 500 KB of SRAM for a meaningful on-sensor processing. The optimal configurations with 1 MB of SRAM use a 256-FU L1 processor and a 2,048-FU L2 processor. These are well matched to existing on-sensor processors [4], [44] and mobile SoCs [5]. Below 500 KB of L1 SRAM, it is significantly worse to assign any workload to the L1 processor since only the CW caching scheme is possible due to capacity limitations. Configurations with more than 1 MB of SRAM do not see additional energy savings.

The 500 KB L1 SRAM size corresponds to the point where the activations for the first few layers can fit in the SRAM which supports the more efficient CA method. The transition at 1 MB corresponds to being able to cache the largest layer activations in the network, and therefore no longer limits possible split locations. Note that the size of the weights of the early layers in MobileNetV3 is insignificant compared to the size of the activations (Fig. 4). If the SRAM is large enough to cache activations, the incremental capacity to support CA+CW is relatively minor.

Above 1 MB for the L1 SRAM, there are many near-optimal configurations, which opens up the possibility to dynamically shift the configuration based on the L2 compute availability. With a 6 MB L1 SRAM, a 512-FU L1 processor alone can perform near-optimal processing up to the full network. For a system designer, this is a good argument for a slightly larger than optimal L1 processor to provide run-time flexibility.

### B. Pixel Area Constraint

The image sensor area scales proportionally to the image resolution and so does the L1 processing. Therefore, we can consider the area constraint in terms of per-pixel area of the image sensor. We vary the per-pixel area from 0.25 $\mu m^2$ to 150 $\mu m^2$. Modern low-power global-shutter image sensors [45] have a per-pixel area in the range of 4.8-9 $\mu m^2$. For the L1 processor, we assume a 28nm technology and use a FU design based on Eyeriss v2 [12].

Similar to the SRAM constraint, there is a minimum per-pixel area required for a meaningful on-sensor processing at 4 $\mu m^2$ which is close to today's state-of-the-art global shutter sensors [46], [47]. As the pixel size continues to scale down, a more advanced processing technology can be used. Alternatively, a higher-resolution image could be down-sampled for processing. Above 4 $\mu m^2$ pixel area, the optimal network split location for a single sensor system shifts from layer 14 to layer 25, 46, 56, and finally the full network. These configurations correspond with progressively more FUs at the L1 processor.

### C. Impact of Improved Memory Technology

Memory systems consistently consume the largest portion of energy on the L1 processor. Both the external DRAM access and SRAM leakage can cost orders of magnitude more energy than the compute itself. Improving the energy efficiency of DRAM access is an industry focus, e.g., the recent LPDDR5 reduces the per-byte data access energy by 6× over the previous LPDDR4 generation [48]. Combined with techniques bringing DRAM onto the package [49], [50], we can expect an order of magnitude energy reduction in DRAM technology in the near future. Similarly, ultra-low-leakage SRAM has also been demonstrated [51], [52] to achieve at least an order of magnitude lower leakage power in the near future. Hence, without loss of generality, we consider the impact on the design space given an 8× reduction in DRAM access energy and SRAM leakage power.

In the baseline shown in Fig. 13(a), if an L1 processor is only equipped with small amounts of SRAM, CW must be used and activations need to be fetched from DRAM presenting an energy bottleneck. With a reduction in DRAM access energy by 8× as shown in Fig. 13(b), CW becomes energy-efficient for a small L1 processor. With the reduced DRAM access cost, only 4 kB of L1 SRAM is needed for meaningful on-sensor processing. This is a large improvement from the 500 kB required in the baseline. With only 4 kB,

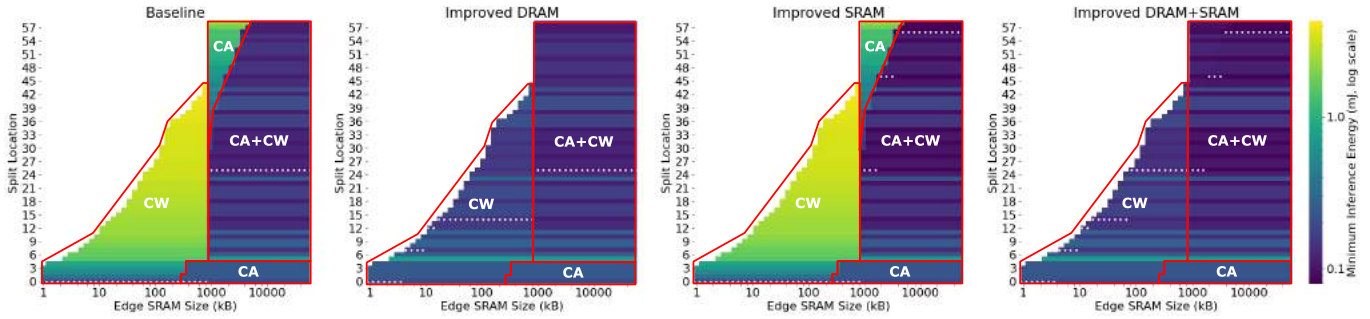



Fig. 13. System energy design space running MobileNetV3 with DRAM energy and SRAM leakage power improvements. (a) represents the baseline system using energies shown in Table I. (b-d) consider an 8× reduction in DRAM communication energy, an 8× reduction in SRAM leakage power, and a combined 8× reduction in both DRAM communication and SRAM leakage, respectively. See Fig. 12 caption for description of plot.

even the smallest sensors are able to support meaningful L1 processing.

A reduction in SRAM leakage power mostly affects the configurations with a large L1 processor. As shown in Fig. 13(c), after an 8× SRAM leakage reduction, a late split point becomes energy-efficient as the workload can be increasingly shifted to a large L1 processor with large low-leakage SRAMs to support CA+CW. Now, above 2 MB of L1 SRAM it is beneficial to process the majority of the network on-sensor.

Considering both the DRAM access energy and SRAM leakage power improvements, energy consumption is improved for both small and large L1 processors as shown in Fig. 13(d). The individual improvements apply to the combined system. Additionally, with both improved DRAM access energy and SRAM leakage power, an L1/L2 equal split can be achieved with 100 kB of L1 SRAM, an order of magnitude reduction over the baseline.

As discussed in the previous sections, with the current memory technology the minimum storage requirements for realistic on-sensor compute are large. However, here we show that an improvement in DRAM access energy can enable more resource-constrained on-sensor processing. Additionally, improved low-leakage SRAM will enable the majority of the DNN to be processed on-sensor, shifting the load away from the L2 processor. Future emerging memory technologies such as RRAM [53], MRAM [54], and 3d-XPoint [55], could further reduce energy consumption, making on-sensor processing feasible for the most energy- and area-constrained systems.

## VI. Simulation Methodology

Current tools allow designers to benchmark individual processors and interfaces [21], [56]. However, optimizing per-processor performance does not necessarily optimize the whole system performance. In order to best map DNN algorithms to a multi-processor system, the designer must understand how the various components interact with each other and the trade-offs. To model these trade-offs, we created DAS. Unlike previous tools, DAS is focused on partitioned algorithms which will be distributed across multiple levels of processors. To speed up the exploration over a vast design space, DAS uses a series of counters and cycle-wise behavioral models for each of the system components. Details on the processor model are in the next section, other component descriptions were omitted here for brevity.

The system is defined with individual component specifications and a connection map. DAS takes as input this system definition and a workload mapping. During simulation, it records all data movement and computation to create a simulation trace. After the simulation, this recorded trace is combined with energy for each operation (see Table I) to estimate the energy consumed during the computation.

### A. Processor Model

We detail the simple processor model the simulator uses to exemplify the basic design principles behind DAS. For AR/VR systems, the processor size, type, and performance can vary widely and thus the model must be flexible. In its most basic form, the processor takes as input instructions and data, performs computation, and outputs results. The details which differentiate the types of processor involve the efficiency of processing, how on-chip data is handled, and its external interfaces. For DAS, a basic processor type has four interfaces: an input and output interface for sharing data to other parts of the system, an on-chip cache interface, and an external memory interface. In addition, the processor's size is defined by the number of MACs it can perform per cycle.

The processor operates on a basic compiled instruction. The instruction dictates the number of reads/writes to each of the interfaces, as well as the computation load in number of MACs. During simulation, the processor uses this instruction to approximate how a real processor would perform the computation. Each portion of the instruction has a corresponding counter which tracks the progression of the simulated computation. The processor uses rules as to how the processing can proceed to enforce run-time dependencies.

As computation progresses, the counters are incremented until they reach the requirement for the instruction. The processor is finished with the instruction when all counters are finished. Then, the next instruction is loaded, counters are reset, and the process restarts. This method of simulating the computation allows DAS to replicate any processor starving issues or bandwidth limitations during computation, as well as any non-trivial interactions between processors.

TABLE IV
COMPARISON OF MEASURED AND SIMULATED PERFORMANCE METRICS

| Platform | Workload | Inference Energy | | Inference Latency | |
|---|---|---|---|---|---|
| | | Published | DAS | Published | DAS |
| Edge-TPU [58] | MobileNetV2 | 11 mJ | 10.1 mJ | 2.6 mS | 1.2 mS |
| Thinker [59] | Alexnet | 0.57 mJ | 0.51 mJ | 2.0 mS | 1.9 mS |
| STM-SOC [60] | Alexnet | 0.44 mJ | 0.43 mJ | 10.8 mS | 9.9 mS |

### B. Performance Validation

To ensure the outputs from DAS are reasonable approximations of the actual hardware performance, we validated the results against three existing edge platforms: Google's Edge-TPU [57], [58], Thinker [59], and STM-SOC [60]. To perform the validation, we modeled the accelerator systems in DAS based on publicly available specifications, then compared the reported performance to the DAS results. For the Edge-TPU, DAS was not able to model the API latency leading to a significant difference. The results of the simulation are shown in TableIV. Overall, DAS tends to slightly underestimate both energy and latency from the published values.

## VII. RELATED WORK

Many domain-specific accelerators have been developed in recent years targeting DNNs [6]–[17]. Many of these accelerators focus on performance of a single-processor system and are optimized for a combination of low power, high utilization, high throughput, and low latency. Previous accelerators for AR/VR [17] focus on challenges of a diverse workload for AR/VR, but do not consider edge processing and thus split processing of DNNs. In our multi-processor system for AR/VR, we care most about the overall energy efficiency. We abstract the general types of accelerators into how they manage their data during processing. Some accelerators [15], [16] use a mixture of caching schemes to maintain high utilization in all layers of the network. We plan to include this hybrid caching scheme type in our future work.

Previous work attempts to better understand how splitting a network may affect computation. Neurosurgeon [18] addresses splitting between a mobile device and the cloud to reduce energy without compromising latency. Neurosurgeon uses a round-trip processing latency and also considers non-DNN workloads. Similar to our ResNet-50 case study, the work shows that for some networks it is best to offload all processing to the larger processor which in their case is the cloud.

In [19] and [20], the authors discuss performing image processing between edge cameras and the cloud and how to split the DNN computation between the camera and the cloud. In both pieces of work, it is demonstrated that compression is needed to reduce the intermediate activation sizes before computation on the edge is worthwhile. The compression has the effect of changing the compression point of the network to overcome the increased data transmission cost.

Previous work has also considered algorithmic modifications to enable on-sensor and edge compute. Particularly, [61] introduces multi-exit CNNs for layer fusion applications. These networks are intended to be split across multiple hierarchies of compute, similar to the situation presented here. Here, the processing algorithm is modified to reduce the common-case communication cost in the system which consequently reduces overall energy requirements and enables edge processing. The principles we present here are complementary to the algorithmic approach and could be used to augment the design of future multi-exit CNNs to be well suited for on-sensor split processing.

In [21], the authors present a simulation infrastructure to analyze the performance of various DNN accelerator architectures. The authors take a fine-grained approach to modeling each architectural feature. It is shown how small design changes can lead to large performance changes. In this work, DAS uses a more abstract model of accelerators to focus on system-level performance trends. As a future direction, a fine-grained simulator like Timeloop [21] could be incorporated into DAS to explore specific accelerator architectures and their impact on a multi-processor system.

## VIII. CONCLUSION

We consider the optimization of DNNs running on a mobile system with on-sensor L1 processors and an edge L2 processor. To support the study, we developed DAS to model the system. We demonstrate the general impact the processor size, type, and network split location have on system energy and performance. Following these trends, we generalize three hand-design principles. The first principle defines a minimum size for the L1 processor based on the network characteristics. The second principle is to allocate the L1 cache space based on the size-reuse product to reduce DRAM access. The final principle is to match the workload split to the L1/L2 size to balance latency and increase processor utilization. Following these principles, we show that a near minimum-energy design can be quickly found. We show the existing Quest 2 VR platform can significantly reduce DNN inference energy with the addition of on-sensor processing. Finally, we explore the design space of an AR/VR system using two sets of real-world design constraints: stacked SRAM size and pixel area of image sensors. We show how an improvement in memory technology can expand feasibility of on-sensor processing to very resource-constrained image sensors.

## ACKNOWLEDGMENT

The authors kindly acknowledge the helpful feedback from Hans Reyserhove, Syed Shakib Sarwar, Ziyun Li, Asif Khan, Chiao Liu, and Barbara De Salvo at Facebook Reality Lab.

## REFERENCES

[1] C. Liu, A. Berkovich, S. Chen, H. Reyserhove, S. S. Sarwar, and T.-H. Tsai, "Intelligent vision systems – bringing human-machine interface to AR/VR," in *IEDM Tech. Dig.*, Dec. 2019, p. 10.

[2] R. LiKamWa, Z. Wang, A. Carroll, F. X. Lin, and L. Zhong, "Draining our glass: An energy and heat characterization of Google glass," in *Proc. 5th Asia–Pacific Workshop Syst. (APSys)*. New York, NY, USA: ACM, 2014, pp. 1–7, doi: 10.1145/2637166.2637230.

[3] R. Pinkham, T. Schmidt, and A. Berkovich, "Algorithm-aware neural network based image compression for high-speed imaging," in *Proc. IEEE Int. Conf. Artif. Intell. Virtual Reality (AIVR)*, Dec. 2020, pp. 196–199.

[4] Sony. *Sony to Release World's First Intelligent Vision Sensors With AI Processing Functionality*. Accessed: Mar. 14, 2020. [Online]. Available: https://www.sony.net/SonyInfo/News/Press/202005/20-037E/

[5] Qualcomm. (2019). *Qualcomm Snapdragon 865 5G Mobile Platform Product Brief*. Brochure. [Online]. Available: https://www.qualcomm.com/system/files/document/files/prod_brief_qcom_sd865_5g.pdf

[6] T. Chen *et al.*, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proc. ASPLOS*. New York, NY, USA: ACM, 2014, pp. 269–284, doi: 10.1145/2541940.2541967.

[7] Y. Chen *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2014, pp. 609–622.

[8] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 367–379.

[9] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ISCA*, 2016, pp. 243–254, doi: 10.1109/ISCA.2016.30.

[10] S. Zhang *et al.*, "Cambricon-X: An accelerator for sparse neural networks," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.

[11] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 1–13.

[12] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, 2019, doi: 10.1109/JETCAS.2019.2910232.

[13] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3D memory," in *Proc. ASPLOS*. New York, NY, USA: ACM, 2017, pp. 751–764, doi: 10.1145/3037697.3037702.

[14] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.

[15] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 553–564.

[16] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," in *Proc. ASPLOS*. New York, NY, USA: ACM, 2018, pp. 461–475, doi: 10.1145/3173162.3173176.

[17] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous dataflow accelerators for multi-DNN workloads," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, 2021, pp. 71–83.

[18] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, 2017, doi: 10.1145/3093337.3037698.

[19] J. Emmons, S. Fouladi, G. Ananthanarayanan, S. Venkataraman, S. Savarese, and K. Winstein, "Cracking open the DNN black-box: Video analytics with DNNs across the camera-cloud boundary," in *Proc. Workshop Hot Topics Video Analytics Intell. Edges (HotEdgeVideo)*. New York, NY, USA: ACM, 2019, pp. 27–32, doi: 10.1145/3349614.3356023.

[20] J. H. Ko, T. Na, M. F. Amir, and S. Mukhopadhyay, "Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained Internet-of-Things platforms," in *Proc. 15th IEEE Int. Conf. Adv. Video Signal Based Surveill. (AVSS)*, Nov. 2018, pp. 1–6.

[21] A. Parashar *et al.*, "Timeloop: A systematic approach to DNN accelerator evaluation," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Mar. 2019, pp. 304–315.

[22] O. VR. (2019). *Introducing Oculus Quest, Our First 6DOF All-in-One VR System*. Launching Spring. [Online]. Available: https://www.oculus.com/blog/introducing-oculus-quest-our-first-6dof-all-in-one-vr-system-launching-spring-2019/

[23] J. Kothari. *Glass Enterprise Edition 2: Faster and More Helpful*. Accessed: May 20, 2020. [Online]. Available: https://blog.google/products/hardware/glass-enterprise-edition-2/

[24] D. Ungureanu *et al.*, "Hololens 2 research mode as a tool for computer vision research," Tech. Rep., 2020.

[25] M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 10–14.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90.

[27] A. Howard *et al.*, "Searching for MobileNetV3," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1314–1324.

[28] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5 MB model size," 2016, *arXiv:1602.07360*. [Online]. Available: https://arxiv.org/abs/1602.07360

[29] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick. (2019). *Detectron2*. [Online]. Available: https://github.com/facebookresearch/detectron2

[30] L. Bose, P. Dudek, J. Chen, S. J. Carey, and W. W. Mayol-Cuevas, "Fully embedding fast convolutional networks on pixel processor arrays," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2020, pp. 488–503.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[32] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: https://arxiv.org/abs/1409.1556

[33] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.

[34] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: https://arxiv.org/abs/1704.04861

[35] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*. [Online]. Available: https://arxiv.org/abs/1510.00149

[36] M. Alliance, "Introduction to the MIPI I3C standardized sensor interface," MIPI Alliance, Piscataway, NJ, USA, Tech. Rep., Aug. 2016.

[37] B. Zimmer *et al.*, "A 0.32–128 TOPS, scalable multi-chip-module-based deep neural network inference accelerator with ground-referenced signaling in 16 nm," *IEEE J. Solid-State Circuits*, vol. 55, no. 4, pp. 920–932, Apr. 2020.

[38] E. Nurvitadhi *et al.*, "In-package domain-specific ASICs for Intel Stratix 10 FPGAs: A case study of accelerating deep learning using TensorTile ASIC," in *Proc. 28th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2018, pp. 106–1064.

[39] E. Nurvitadhi *et al.*, "Why compete when you can work together: FPGA-ASIC integration for persistent RNNs," in *Proc. IEEE 27th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, Apr. 2019, pp. 199–207.

[40] H. Tsugawa *et al.*, "Pixel/DRAM/logic 3-layer stacked CMOS image sensor technology," in *IEDM Tech. Dig.*, Dec. 2017, pp. 3.2.1–3.2.4.

[41] T. Miura *et al.*, "A 6.9 $\mu$m pixel-pitch 3D stacked global shutter CMOS image sensor with 3M Cu-Cu connections," in *Proc. Int. 3D Syst. Integr. Conf. (DIC)*, Oct. 2019, pp. 1–2.

[42] *4Gb: X4, X8, X16 DDR4 SDRAM*, Rev. K, Micron, Boise, ID, USA, 2014.

[43] O. VR. *Introducing Oculus Quest 2, the Next Generation of All-in-One VR*. Accessed: Sep. 16, 2020. [Online]. Available: https://www.oculus.com/blog/introducing-oculus-quest-2-the-next-generation-of-all-in-one-vr-gaming/

[44] R. Eki *et al.*, "9.6 A 1/2.3inch 12.3 Mpixel with on-chip 4.97TOPS/W CNN processor back-illuminated stacked CMOS image sensor," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2021, pp. 154–156.

[45] Omnivision. *1 Megapixel and Below*. Accessed: Sep. 13, 2020. [Online]. Available: https://www.ovt.com/image-sensors/1-megapixel-and-below

[46] J.-K. Lee *et al.*, "5.5 A 2.1e$^-$ temporal noise and $-105$ dB parasitic light sensitivity backside-illuminated 2.3 $\mu$m-pixel voltage-domain global shutter CMOS image sensor using high-capacity DRAM capacitor technology," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2020, pp. 102–104.

[47] G. Park *et al.*, "A 2.2 $\mu$m stacked back side illuminated voltage domain global shutter CMOS image sensor," in *IEDM Tech. Dig.*, Dec. 2019, pp. 16.4.1–16.4.4.

[48] Y.-H. Kim *et al.*, "25.2 A 16 Gb sub-1 V 7.14 Gb/s/pin LPDDR5 SDRAM applying a mosaic architecture with a short-feedback 1-tap DFE, an FSS bus with low-level swing and an adaptively controlled body biasing in a 3rd-generation 10 nm DRAM," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2021, pp. 346–348.

[49] D. U. Lee *et al.*, "25.2 A 1.2 V 8Gb 8-channel 128 GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29 nm process and TSV," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 432–433.

[50] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *Proc. IEEE Hot Chips Symp. (HCS)*, Aug. 2011, pp. 1–24.
[51] H. Pilo *et al.*, "A 64 Mb SRAM in 22 nm SOI technology featuring fine-granularity power gating and low-energy power-supply-partition techniques for 37% leakage reduction," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2013, pp. 322–323.
[52] J. Chang *et al.*, "A 20 nm 112 MB SRAM in high-$\kappa$ metal-gate with assist circuitry for low-leakage and low-$V_{MIN}$ applications," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2013, pp. 316–317.
[53] C.-C. Chou *et al.*, "A 22 nm 96KX144 RRAM macro with a self-tracking reference and a low ripple charge pump to achieve a configurable read window and a wide operating voltage range," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2020, pp. 1–2.
[54] Y.-D. Chih *et al.*, "13.3 A 22 nm 32Mb embedded STT-MRAM with 10ns read speed, 1M cycle write endurance, 10 years retention at 150°C and high immunity to magnetic field interference," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2020, pp. 222–224.
[55] F. T. Hady, A. Foong, B. Veal, and D. Williams, "Platform storage performance with 3D XPoint technology," *Proc. IEEE*, vol. 105, no. 9, pp. 1822–1833, Sep. 2017.
[56] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011, doi: 10.1145/2024716.2024718.
[57] B. Rutledge and V. Tank. *Introducing Coral: Our Platform for Development With Local AI*. Accessed: Feb. 11, 2021. [Online]. Available: https://developers.googleblog.com/2019/03/introducing-coral-our-platform-for.html
[58] R. Hadidi, J. Cao, Y. Xie, B. Asgari, T. Krishna, and H. Kim, "Characterizing the deployment of deep neural networks on commercial edge devices," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Nov. 2019, pp. 35–48.
[59] S. Yin *et al.*, "A high energy efficient reconfigurable hybrid neural network processor for deep learning applications," *IEEE J. Solid-State Circuits*, vol. 53, no. 4, pp. 968–982, Apr. 2018.
[60] G. Desoli *et al.*, "14.1 A 2.9TOPS/W deep convolutional neural network SoC in FD-SOI 28 nm for intelligent embedded systems," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 238–239.
[61] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 328–339.

**Reid Pinkham** (Member, IEEE) received the B.S. degree in physics, and the B.S.E., M.S.E., and Ph.D. degrees in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2017, 2017, 2019, and 2021, respectively. After receiving his Ph.D., he joined Facebook Reality Labs, Redmond, WA, USA, in late 2021, as a Research Scientist working on intelligent vision systems. His research interests include high performance edge compute systems, computer architecture, real-time machine learning, and emerging processing techniques.

**Andrew Berkovich** (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in electrical and computer engineering from the University of Maryland, College Park, MD, USA, in 2011, 2015, and 2017, respectively. During his Ph.D. studies, he developed biologically-inspired low-light image sensors. Since August 2017, he has been with Facebook Reality Labs, Redmond, WA, USA, where he is currently a Research Scientist developing intelligent image sensors and systems.

**Zhengya Zhang** (Senior Member, IEEE) received the B.A.Sc. degree in computer engineering from the University of Waterloo in 2003 and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Berkeley (UC Berkeley), in 2005 and 2009, respectively.

He has been a Faculty Member of the University of Michigan, Ann Arbor, since 2009, where he is currently a Professor with the Department of Electrical Engineering and Computer Science. His research interests include low-power and high-performance VLSI circuits and systems for computing, communications, and signal processing. He serves on the Technical Program Committees of IEEE VLSI Symposium on Technology and Circuits and IEEE Custom Integrated Circuits Conference (CICC) since 2018. He was a recipient of the University of Michigan College of Engineering Neil Van Eenam Memorial Award in 2019, the Intel Early Career Faculty Award in 2013, the National Science Foundation CAREER Award in 2011, and the David J. Sakrison Memorial Prize from UC Berkeley in 2009. He was an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART I: REGULAR PAPERS for the period 2013–2015 and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART II: EXPRESS BRIEFS for the period 2014–2015. He has been an Associate Editor of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS since 2015.