ANSA: Adaptive Near-Sensor Architecture for Dynamic DNN Processing in Compact Form Factors

Reid Pinkham, *Member, IEEE*, Jack Erhardt, *Member, IEEE*, Barbara De Salvo, *Fellow, IEEE*, Andrew Berkovich, *Member, IEEE*, and Zhengya Zhang, *Senior Member, IEEE*

***Abstract*— Advanced edge sensing/computing devices, such as AR/VR devices, have a uniquely challenging adaptive baseline workload and camera sensor structure. These devices must process images in real-time from multiple sensors, placing a large burden on a typical centralized mobile SoC processor. Augmenting the sensors with a package-integrated near-sensor processor can improve the device's processing performance as well as reduce energy consumption. This near-sensor processor must adapt to the dynamic workloads, fit within a limited silicon footprint and energy envelope, and satisfy the real-time requirement. In this work, we present ANSA, a near-sensor processor architecture supporting flexible processing schemes and dataflows to maintain high efficiency for dynamic CNN workloads. ANSA is scalable to sub-mm^2 sizes to match the footprint of advanced image sensors. ANSA supports module-level power gating to adapt the compute capacity to dynamic workloads. Finally, ANSA leverages recent advancements in high-density non-volatile memory and 3D packaging to support weight storage within the area constraints of an image sensor. Overall, ANSA achieves inference energy consumption up to 30× lower than a standard SIMD baseline. Additionally, our design's scalability allows it to achieve up to 2.76× lower average inference energy at 4.5× lower silicon area compared to competing edge accelerator designs.**

***Index Terms*— Edge computing, low power computing, augmented reality, virtual reality, DNN accelerator.**

I. Introduction

VISUAL compute constitutes a major contribution to total power consumption on AR/VR devices [1]. Recent trends in modern AR/VR devices have seen the inclusion of higher resolution sensors [2], higher frequency inference, and the inclusion of more human-machine inference modalities on a single device [3]. As these trends can result in increased energy cost for CNN processing on both AR and VR devices in the near future, techniques to support low-energy real-time processing on battery-limited devices is of key interest.

Manuscript received 8 August 2022; revised 28 October 2022 and 22 November 2022; accepted 27 November 2022. Date of publication 23 December 2022; date of current version 27 February 2023. This work was supported in part by Meta. This article was recommended by Associate Editor M. Martina. *(Reid Pinkham and Jack Erhardt contributed equally to this work.) (Corresponding author: Jack Erhardt.)*

Reid Pinkham is with the Department of Electrical Engineering and Computer Science (EECS), University of Michigan, Ann Arbor, MI 48109, USA, and also with the Reality Labs—Research, Redmond, WA 98052 USA (e-mail: pinkhamr@umich.edu).

Jack Erhardt and Zhengya Zhang are with the Department of Electrical Engineering and Computer Science (EECS), University of Michigan, Ann Arbor, MI 48109, USA (e-mail: erharj@umich.edu; zhengya@umich.edu).

Barbara De Salvo and Andrew Berkovich are with the Reality Labs—Research, Redmond, WA 98052 USA (e-mail: barbarads@meta.com; andrew.berkovich@meta.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2022.3228725.

Digital Object Identifier 10.1109/TCSI.2022.3228725

Recently, advancements in both packaging and imaging technology have enabled limited processing to be directly added to an image sensor to create a smart sensor. A smart sensor typically features a 3D integration of a CMOS image sensor with a memory die and a processor die. As processing density increases, it becomes feasible to fit an increasing amount of processing on the device. By co-locating the processing with the sensor, data is processed near the source and even more energy can be saved by cutting the transmission since only the result of computation is relayed to any off-device processor and compute is orders of magnitude less expensive than even on-chip communication in modern devices [3]. This approach also enhances privacy and security since the images never leave the device. Supporting this low-energy real-time processing on a silicon-area-limited edge processor is of key interest.

However, modern CNN models represent challenging workloads for near-sensor processing for several reasons. Firstly, a near-sensor processor needs to be sufficiently flexible to adapt for dynamic workloads. Secondly, the processor must support this compute while being limited in silicon area, so that the processor can be co-packaged with a compact sensor (on the order of sub-mm^2 for an advanced image sensor), and to meet the tight area limitations typical of AR/VR devices [3]. Finally, the processor must achieve these objectives while maintaining the high energy efficiency required to meet the stringent energy demands of edge hardware applications, on the order of sub-mJ per inference demonstrated in [4], [5].

To meet the tight form factor limitations imposed by modern sensors, a near-sensor stacked processor needs to fit the footprint of the sensor, and thus its silicon area and its cache space are limited. Under a common layer-by-layer CNN processing scheme, a large cache is needed to store intermediate activations. Methods of reducing the cache capacity have been proposed by modifying the layer processing scheme [6], [7], [8], [9]. However, these methods generally achieve this reduction at the expense of activation reuse, reducing energy efficiency by incurring additional data movement between SRAMs and compute units

To meet versatile image processing needs of different tasks, a near-sensor processor needs to support different CNN models of varying computational complexities and the resulting dynamic workloads [10], [11]. The dynamic workloads are especially relevant to AR/VR devices, as they need to handle varied tasks based on input and user interactions. Most of the CNN processing architectures [12], [13], [14], [15], [16] rely on a fixed dataflow to simplify the design and improve its peak efficiency, but a fixed dataflow will not be equally efficient

across all layers of the network or different model structures. Some [17], [18] considered supporting multiple dataflows, but, as we will show later, previous flexible dataflow architectures do not scale when the silicon area and cache size are shrunk to compact sizes, a key requirement for near-sensor processing.

To meet the real-time processing requirement at high energy efficiency, a near-sensor processor needs to efficiently handle the batch size of one. Some common accelerator architectures, such as the systolic array [14] and large SIMD processors [15], [19], rely on large batch sizes for efficient computation. Increased batch sizes often hide inefficiencies in the mapping but offer an easy method to increase utilization which is not possible in edge devices. Furthermore, energy efficiency is not always equivalent to high utilization. Since the external interfaces represent a significant portion of energy consumption, reducing the external traffic should be prioritized over increasing utilization for minimizing energy.

We present Adaptive Near-Sensor Architecture (ANSA), a compute architecture for an advanced image sensor platform consisting of a 3D stack of a CMOS image sensor, a high-density NVM, and an ANSA processor. ANSA is specifically designed with three goals in mind: 1) to adapt to dynamic workloads, 2) to scale to extreme compact sizes for integration with an image sensor, and 3) to achieve competitive energy efficiency while meeting real-time processing latency.

In summary, the contributions of ANSA are:

- *Flexible processing schemes and dataflows for dynamic workloads*: ANSA adapts the processing scheme based on a CNN layer's or a set of layers' characteristics to meet the on-chip storage limitation while minimizing the processing energy. ANSA also adapts the dataflow for the optimal mapping of each layer of multiple CNN workloads onto compute units to reduce on-chip cache bandwidth and meet the latency requirement while minimizing the processing energy.
- *Shuffle buffer for depth-wise layers*: ANSA uses a shuffle buffer to streamline the input to the VMM to reduce cache bandwidth and increase utilization in processing depth-wise layers.
- *Scalability to extreme compact sizes*: ANSA's adaptivity allows it to be scaled down to extreme compact design points on the order of sub-mm^2 while maintaining a greater degree of processing efficiency than existing adaptive NN architectures. We quantify aspects of the ANSA design that have the largest impact on energy and performance at different scales, and introduce several configurations suitable for demanding edge platforms such as an AR/VR device.
- *Agile power control*: ANSA utilizes NVM and module-level power gating of SRAM resources to adapt to a dynamic CNN workload on a per-input-frame basis to minimize energy.

Leveraging these features, ANSA can be implemented in compact form factors and deliver better energy efficiency to real-time compute systems for both single- and multi-model CNN workloads. Running a representative dynamic workload, ANSA is able to reduce the required processor area by 6.4× and 2.5× compared to NVDLA [19] and architectures featuring state-of-the-art edge-optimized processing schemes [7], respectively. Additionally, the energy is reduced by 1.66× and 1.97×, respectively.

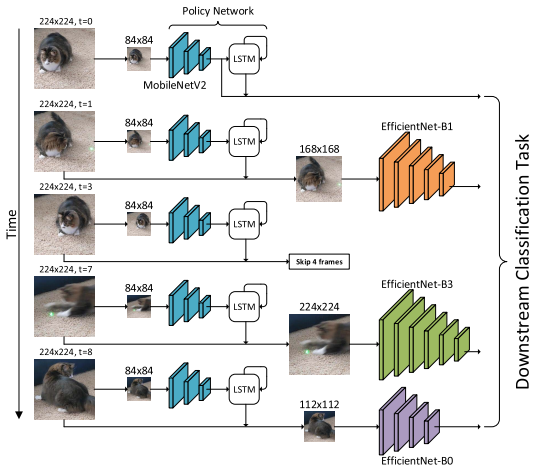


Fig. 1. AR-Net structure. The algorithm uses a lightweight policy network to select between multiple resolution/network pairs at run time which reduces the overall processing load.

The rest of the paper is structured as follows. In Section II we give an overview of the CNN models, NVM technologies, and advanced sensor platforms for near-sensor processing. In Section III we conduct a survey of existing flexible accelerators and evaluate their suitability for near-sensor processing. In Section IV we detail ANSA's architectural elements. In Section V we discuss ANSA's flexible layer processing schemes. In Section VI we discuss ANSA's flexible dataflows. Finally the results are presented in Section VII.

II. Background

We use AR/VR devices as the primary motivation of this work as they represent the frontier of edge platforms in terms of computing and sensing capabilities. The characteristics of AR/VR devices are applicable to other edge platforms.

AR/VR devices use a mixture of traditional and Neural Network (NN) based image processing techniques. Here, we focus on the NN, and specifically the CNN based techniques since they usually represent the dominant portion of the processing workload for modern sensing devices [1]. Many common tasks for an AR/VR device use CNNs, such as hand tracking [20], gesture recognition [21], [22], scene recognition and segmentation [23], mapping [24], and more. All these algorithms must be run in real time, defined as 30 frames per second (FPS), representing a baseline workload. It is often not feasible to simultaneously run all of these algorithms in parallel. However, many applications use a common backbone CNN to extract a hyper-dimensional feature map as a prepossessing stage. Sharing this backbone between workloads reduces the processing and computational diversity required. In this work, we focus on accelerating this backbone near-sensor. Common choices for this backbone are ResNet-50 [25], MobileNet [26], [27], and EfficientNet [28].

A. Dynamic Workloads

Recently, there has been an effort to reduce the complexity of the backbone network. For example, AR-Net [29] uses an adaptive resolution backbone to feed a classifier for action recognition. It employs a lightweight recurrent policy network to select one of multiple actions at each frame. The actions

TABLE I
BACKBONE SPECIFICATIONS FOR AR-NET

Backbone Network	Resolution	Weight Size	MACs
0: MobileNet-v2	14% (84^2)	3.5 MB	53.4 M
1: EfficientNet-B0	25% (112^2)	8.8 MB	167 M
2: EfficientNet-B1	56% (168^2)	11.3 MB	443 M
3: EfficientNet-B3	100% (224^2)	16 MB	1.07 G

can be one of a set of backbone networks operating on full or reduced resolution input image, listed in TABLE I, or skipping one or more frames of processing, as illustrated in Fig 1. The backbones output a latent feature vector of the same size which is used by a classifier to determine the action in a video clip.

By switching between backbones, AR-Net can reduce the computation and energy consumption of processing. However, its setup poses a challenge for a constrained near-sensor processor for two reasons. First is the large amount of storage needed for the model weights of all the backbone networks. Second, the processor must be capable of handling the largest backbone which requires two orders of magnitude more compute than the smallest backbone. A processor will necessarily be over-provisioned for small and medium-size workloads, which is costly for a near-sensor processor where silicon area and power are a premium.

B. NVM Technologies

In this work, we consider an advanced image sensor platform with stacked compute and memory integrated in one package. The most common option for on-chip memory is SRAM which has a good density and high speed, but consumes high standby power and is volatile. For the amount of weight storage necessary for this application, modern SRAMs can draw up to 90mW of static power, which accounts for a power draw up to 19× higher than the average consumed by compute resources and exceeding the per-inference energy objective of our proposed architecture. Many processors also use DRAM as an off-chip high-capacity storage backed by SRAM. On a constrained sensor platform, however, this solution costs an undesirable complexity [30], and also has high standby power and is volatile.

As image sensor sizes continue to shrink, it is challenging to fit the required CNN parameters in the stacked memory die that cannot be shrunk as much as the image sensor. One possible solution is to use high-density stacked memories such as HBM [30]. These memories have a higher density than traditional SRAM, but tend to operate at lower frequencies and require a significant amount of supporting controllers and circuits. We consider emerging non-volatitle memory (NVM) such as Resistive RAM (RRAM) [31] and Magnetoresistive RAM (MRAM) [32] which retain their contents between power cycles and are ideal for devices on standby for long periods. Both of these technologies offer similar latency and densities at or exceeding that of SRAM for a comparable technology node [33], [34], with density expected to be scalable in the same way as SRAM has been in recent years. Major foundries now support both of these memories in their existing technology nodes [31], [32].

C. Compact Sensor Platforms

In order to integrate processor, memory, and image sensor, three dies must be integrated into the same package [5], [35]. Common integration technologies include Through Silicon Vias (TSV), copper pillars, micro-bumps with an interposer, and wirebonding. By combining these technologies, it is possible to integrate more than two dies in the same package. TSVs and copper pillars allow for multiple dies to be stacked on top of each other to create a dense 3D multi-die configuration but are costly to manufacture. A 2.5D integration on a passive interposer is more cost effective and can increase yield, but it increases the total package size.

For the advanced packaging options, the area of the sensor, memory, and compute dies must be closely matched, particularly for using TSVs. Given a per-pixel area of the sensor, we can use this requirement to place a lower bound on the NVM and compute area. Typical state-of-the-art rolling shutter image sensors can have a pixel size of 1 μm^2, with a global shutter having a pixel size as low as 4.8 μm^2 [36], [37]. For the standard 224^2 input image to the largest backbone, this would correspond to a sensor area of 0.051 mm^2 and 0.243 mm^2 for rolling and global shutter sensors respectively. The sub-mm^2 image sensors place a stringent limit on the silicon area allocation for processing.

It is important for a near-sensor processor to store all CNN weights in near-sensor memories to minimize access latency and energy. For an algorithm such as AR-Net using multiple backbones, a weight memory of nearly 40 MB is needed. If we use a 22nm MRAM [32] to implement weight storage, the MRAM area exceeds 10 mm^2. This disparity in area between the sensor and memory is a key challenge which must be overcome. Solutions to bridge this gap include stacking multiple memory dies, weight compression through sparsity or tensor decomposition, sharing weights per layer between networks, or using a more advanced process node [33]. For this work, we assume a combination of these techniques will bridge the gap between sensor and memory area in the near future.

III. RELATED WORK

This work is related to adaptive NN architecture and its application to near-sensor processing. Near-sensor processing needs to be compact and energy-efficient, but previous adaptive NN architectures are not scalable to extreme compact design points and they do not offer a low energy advantage in particular. In the following we briefly review related work and the challenges that need to addressed.

A. Flexible NN Architectures

Several recent works have proposed flexible processing schemes to efficiently support inference on a range of CNN backbones. MAERI [18] supports this compute scheme with a configurable heterogeneous compute fabric, and a dataflow graph partitioning algorithm acting as a compiler from ML algorithms onto this fabric. FlexFlow [17] supports multiple convolutional loop unroling patterns in a more conventional systolic array to achieve high data reuse across common CNN layers. Eyeriss v2 [4] uses an adaptive NoC to achieve adaptive bandwidth across compute workloads, and exploits compute sparsity at the PE level. These flexible architectures all follow a common layer-by-layer processing scheme.

These works achieve high performance and energy efficiency on a variety of workloads, and can be scaled up to support large workloads, but they suffer significant performance and efficiency losses when scaled down in compute resources due to their reliance on SRAM of a sufficiently large size to store weights of large neural network models. For a fixed area budget, layer-by-layer architectures must sacrifice compute resources to make space for cache, resulting in up to 24× higher energy consumption at small processor sizes than the ANSA design we will demonstrate in this paper.

B. Multi-Layer NN Architectures

Related works have proposed to reduce the reliance on large on-chip caches. [6] proposes splitting activations into tiles and processing this data through multiple layers before moving to the next tile. [7] further proposes the use of buffering rows of intermediate activations to support depth-first execution. This concept is extended in [9] to include redistribution of receptive field in CNN architectures from earlier to later layers, reducing redundant computation cost associated with computing on patches of activations. However, these compute schemes do not scale in bandwidth, and thus can result in poor resource utilization and lower energy efficiency for highly parallel compute resources such as VMMs. Additionally, these works do not support the flexible dataflows necessary for adaptive algorithms and multiple backbone networks. Finally, these compute methods incur substantial data movement, as weights must be reloaded frequently when switching layers, and activations are frequently moved in and out of compute resources. This lack of flexibility results in up to 29× higher energy consumption at small processor sizes than the ANSA design we will demonstrate in this paper.

C. On-Sensor Co-Processors

The recent work from Sony [5], [38] has integrated CNN compute on sensor. However, this design features a megapixel scale image sensor, while the processing is limited to a single lightweight CNN model on a downsampled sensor image. The mismatch between the high sensor resolution and the low compute capacity demonstrates the difficulty in designing adaptive and compact near-sensor processing to achieve high efficiency and real-time performance. This serves as the primary motivation of this work.

IV. ANSA Architecture Elements

ANSA adopts a simple, parameterizable design with the flexibility to adapt to layers of different shapes and changing workloads, and to scale to fit a range of area budgets. A key aspect of our architecture is its flexibility in both *processing schemes*, which control how CNN layers are assigned to it, and *dataflows*, which control how computation is performed for each layer. At least one of these two aspects are fixed in common CNN accelerators to reduce design complexity. However, here we demonstrate how maintaining this flexibility allows us to scale to efficiently compute a diverse range of workloads.

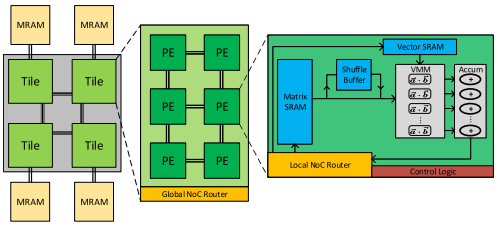


Fig. 2. ANSA's high-level architecture. The example design has 4 tiles with 6 PEs per tile. Each PE contains a VMM, local memories, an accumulation buffer, and control logic. Data is transported on the chip via a two-level NoC driven by global routers in each tile, and local routers within each PE.

A. Top Architecture

ANSA is an inference architecture. The CNN weights are stored in NVMs that are placed in a read-only mode in runtime. An overview of ANSA is shown in Fig 2. At the top level, the architecture shares similarities with previous work [4], [16], [39], but differences are in how we map workloads to the architecture. ANSA is comprised at the top level of a number of tiles connected via a global Network on Chip (NoC). Each tile has a dedicated interface to an external NVM, in this case MRAM. Within each tile there are a number of PEs. Each PE is connected via a local NoC. Together, the global and local NoCs form a hierarchical-mesh NoC (HM-NoC), similar to that proposed in [4], [16]. The global NoC connects the tiles and external NVMs while the local NoC connects PEs within one tile.

ANSA allows for module-by-module power gating of PEs, tiles, and MRAM. Combined with support in the CNN mapping, powering down unused portions of the processor eliminates their static energy consumption. ANSA can efficiently adapt its compute capacity to meet the requirements for each step of workload execution.

The number of tiles and MRAM interfaces, the number of PEs per tile, the shape of the Vector-Matrix-Multiplier (VMM) inside a PE, and the capacity of the SRAM buffers are the parameters which specify a particular design. In Section VII, we explore how each of these parameters affects the performance of the accelerator. In the following, we give a brief overview of key aspects of ANSA.

B. PE

Each PE contains the control logic to perform convolution, non-linearity, pooling, weighting, and fully-connected operations. The central element of the PE is the VMM, as shown in Fig 2. The VMM computes $y = Ax$, where x is a vector of length L_{vec} and A is an $N_{vec} \times L_{vec}$ matrix. Each PE is backed with some dedicated vector SRAM and matrix SRAM for both inputs. All values are 8-bits. In CNN processing, a vector typically represents a segment along the channel dimension of a single spatial location. The VMM computes N_{vec} sets of L_{vec}-length inner products in every cycle, i.e., one vector-matrix product each clock cycle. The outputs are summed in accumulation registers, which are then passed to the local NoC router to other PEs in the tile.

It has been previously shown that compared to Matrix-Matrix Multiplication (MMM) and Vector-Vector Multiplication (VVM). VMM represents a good balance between flexibility and complexity for CNN workloads [40],

as they exhibit high data reuse without reliance on batching like MMM or systolic array structures, which rely on batch-level parallelism to fully utilize the MAC units for achieving high throughput and high efficiency [14]. Supporting individual VVM units which do not share a common input increases the memory bandwidth usage, and requires complex distribution and accumulation paths. On the other hand, supporting MMM units decreases the memory bandwidth, but also reduces flexibility and the hardware utilization.

C. Flexible Processing Schemes

To meet the size constraint for near-sensor processing and optimize for energy efficiency, ANSA supports multiple processing schemes on a per-layer basis. It is common in CNN accelerators to process a model layer-by-layer. In this scheme, each layer is processed in full before the next layer's processing begins. The alternative is multi-layer or fused-layer processing, where partial results from one layer are immediately fed to the next layer processing. Comparing layer-by-layer and multi-layer processing, layer-by-layer simplifies the data management during computation since only one set of input activations, weights, and output activations are required at any given time, but it takes more activation memory, especially when an input resolution is high. Multi-layer processing reduces activation memory, but it requires larger weight memory and/or consumes more NVM bandwidth. A detailed discussion is given in Section V.

D. Flexible Dataflows

To improve energy efficiency and performance, ANSA provides multiple dataflow options. The weights and input activations can be loaded to either the vector SRAM or the matrix SRAM inside a PE. Mapping can change from layer to layer to achieve the most energy-efficient computation. However, the flexibility adds complexity in the accumulation phase because the order of the output data must be matched to the order required at the input of the next phase. This reordering of data results in an overhead which must be accounted for.

Depending on the dataflow, accumulation of the results may take place locally, within a tile, or between multiple tiles. Localized accumulation requires a minimal amount of NoC communication, but limits the flexibility of the possible layer mapping. For example, if all channels of the input are mapped across multiple PEs, intermediate partial sums must be transmitted across the NoC to a single accumulation unit for summation, which can saturate the NoC. Alternatively, the computation for each filter can be localized to a PE, where the complete computation across channels will occur in rounds, reducing NoC traffic at the expense of more scratchpad reads and writes. The dataflows options are discussed in Section VI.

E. Shuffle Buffer

The Depthwise Convolution (DWConv) operation is common in lightweight networks to reduce the number of MACs for a 2D convolution. Unlike conventional convolution layers, the DWConv layer does not sum across the channel dimension, instead only summing across the width and height of the filter. Common accelerators rely on summing across the channel dimension in the vector units and cannot distribute inputs spatially, resulting in a low hardware utilization for DWConv layers. For real-time operations, the batch dimension cannot be utilized to improve utilization.

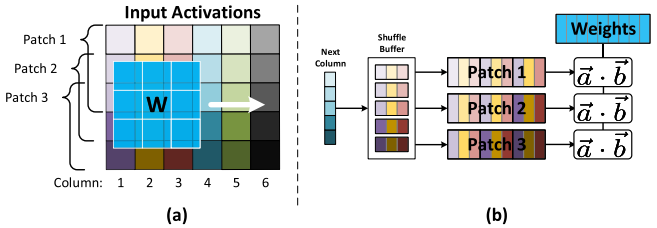


Fig. 3. Functional diagram of an example shuffle buffer sized for a 3×3 DWConv. (a) A 3×3 weight matrix is convolved over the input activations. (b) The input activations are ordered by column and split into three patches. A column of 5 inputs are fetched; the column is split to three patches, each corresponding to a 3×3 region of the input activation; and patches are flattened to form a 1D vector to each input of the VMM.

To address this challenge, each ANSA PE includes a shuffle buffer between the matrix SRAM and the VMM, which can distribute inputs to the VMM spatially instead of along the channel dimension. Fig 3 shows the operation of the shuffle buffer. In each cycle, the shuffle buffer receives a column of input activations from the matrix SRAM, and the shuffle buffer redistributes the saved inputs to form patches of input activations which are sent to the VMM. In this manner, the shuffle buffer exploits the reuse of the input activations between patches to reduce the required bandwidth and supports a higher VMM utilization for DWConv layers.

F. System Level Integration

The architecture described thus far fills the role of an on-chip CNN processor. Additional system-level components exist in the form of the Image Signal Processor (ISP) and off-chip interfaces. In this work, we assume similar system level design to that used previously in [5].. We assume layer-by-layer mappings are assigned offline and stored in a small instruction SRAM, which are loaded into a controller at runtime to control the ANSA compute fabric. We also assume post-ISP images are being used, and a MIPI interface to communicate backbone results off-chip. The system-level components, ISP and MIPI interface, contribute a fixed overhead to support 30 FPS real-time processing of fixed image sizes, and they do not impact the CNN accelerator design space which is the focus of this work.

V. Flexible Processing Schemes

A major drawback of the layer-by-layer technique is that the entire input and output activation must fit in on-chip cache for the most efficient processing. Typical CNNs for image processing gradually compress the intermediate representation throughout the network. For example, Fig 4 shows the activation size throughout the EfficientNet-B3 network. With this network, there is a large peak in activation size in the early layers of over 1.7 MB, assuming 1 B per weight value. The large on-chip memory can be prohibitive for near-sensor

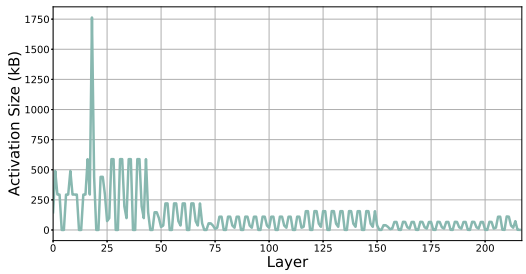


Fig. 4. Activation sizes for all layers of the EfficientNet-B3 network at 224×224 resolution.

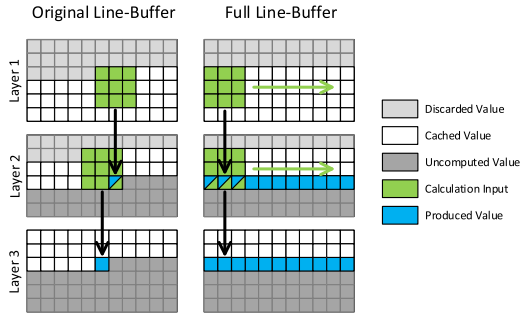


Fig. 5. Intermediate activations between layers are stored in line buffers and processed immediately after they are produced, reducing on-sensor memory requirements.

processors. However, throughout the rest of the network, significantly less memory is required. While processing layer-by-layer is desirable for simplicity and efficiency, it can increase the size of the processor.

Alternative processing schemes have been proposed to alleviate the memory burden. These include tiling the input before processing, and fused-layer computation [6]. Recently, a Line-Buffer (LB) approach to depth-first execution was proposed [7], as illustrated in Fig 5. With this method, only a few rows of the input are buffered for each layer, dramatically reducing the on-processor storage requirement. Inputs can be further tiled or split along their width to reduce the required buffering capacity. In the example of EfficientNet-B3, the LB approach allows the three layers around the largest activations to be fused to reduce the activation caching by 70% to 600 kB.

However, the LB approach cannot reduce the weight storage requirements. While the previous work [7] assumes that weight sizes are significantly smaller than activation sizes, as is often true for high-resolutions models. With lower-resolution backbones the weight size can be as large as or larger than the activations. In ANSA, weights are located in external NVM and cached on-chip when used. In the worst case when the weights are not fully cached on-chip, the weights need to be streamed into the processor on demand, costing higher energy and latency to access the NVMs.

A. Processing Scheme Options

Both layer-by-layer and multi-layer LB approaches can be beneficial depending on the layer and input size. Therefore, ANSA supports the complete spectrum of layer processing schemes listed below on a per-layer or set-of-layers basis.

- **Full Layer**: This is a layer-by-layer scheme that requires the input activations, output activations, and layer weights to fit in on-chip caches. This scheme is the most energy-efficient and performance-optimal since activation and weight reuse is maximized and the NVM read is minimized.
- **Stream Weights**: This is a layer-by-layer scheme for designs with a smaller on-chip caches. The full weights of a layer may not fit in the on-chip caches and must be streamed in as they are used. Only a portion of the layer's weights are cached at any given time, requiring a higher NVM bandwidth.
- **Full Line-Buffer (Full LB)** : This scheme is most similar to that proposed in [7]. Here, several consecutive layers are fused for computation. During inference, rows of the intermediate activations are buffered, and can be further split into smaller widths according to some splitting parameter. Unlike in [7], this scheme will produce a full row of the output at each step instead of location-by-location to reduce the cache bandwidth required to successively read the weights. This scheme requires the first layer input activations, the buffered lines, final layer output activations, and weights for all fused layers to fit in on-chip caches.
- **Partial Line-Buffer (Partial LB)**: Derived from the Full LB scheme, this scheme does not require all the fused layers' weights to fit in on-chip caches. Similar to the stream weights scheme, weights are streamed from the NVM when required. Compared to the Full LB scheme, this scheme uses more NVM read.
- **Stream Line-Buffer (Stream LB)**: This scheme is a further reduction of the partial LB scheme, and is equivalent to the processing scheme proposed in [7] with weight streaming. Here, a single spatial location of the fused-layer output (instead of one row of output) is computed at each step. This scheme reduces the caching requirements of the activations at the cost of increasing the NVM read bandwidth to fetch the weights for each input location.

B. Processing Scheme Assignment

Before inference, each CNN layer must be assigned one of these processing schemes for mapping to an ANSA processor. This assignment will depend on both the layer's characteristics and the processor's memory capacity. When the memory capacity on the processor is sufficient to support the layer-by-layer approach, the Full Layer or Stream Weights scheme is assigned. When the memory capacity is not sufficient to support the layer-by-layer approach, one of the LB schemes will be assigned. The choice and number of layers to be fused must also be considered. For deep CNN models, this creates a large design space with no straightforward method to determine which layers to fuse for the most efficient computation.

We use a greedy assignment beginning from the first layer of the network, with some provisions for non-greedy assignment in specific scenarios we demonstrate later in this section. If the processor has the activation cache capacity to support full layer inference, that processing scheme is used, either Full Layer or Stream Weights based on the processor's available memory cache, and that layer is marked as complete. If the processor does not have sufficient activation cache, however,

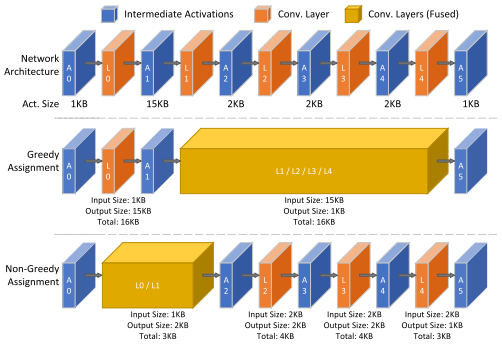


Fig. 6. Scenario in which greedy assignment scheme can produce sub-optimal mappings. Each layer must fit its inputs and outputs into a 16KB Cache.

an appropriate Line Buffer method is assigned to this layer, and subsequent layers are fused with this same method until the output activation of the fused layers is small enough to fit into the available cache. Note that processing scheme assignment is based on both activation and weight cache availability, and that these two conditions are orthogonal. Beginning with the Full LB without splitting, successively more layers are fused as long as the input activations, layer weights, and output activations can fit in on-chip caches.

If there is no length of fused layers which will fit, the input splitting is applied and the search is repeated. If the splitting creates chunks that are too small, the splitting is reset and the next LB scheme is tried. If after this search process no mapping is found, the mapping is rolled back to the previous non-fused layer and the LB scheme is repeated from there. If after rolling back no mapping will fit onto the processor, the processor is considered insufficient to run the CNN.

We find that the greedy assignment generally produces satisfactory mappings competitive with hand-optimized mappings of the layers on a range of EfficientNet [28], MobileNet [26], and ResNet [25] backbones, which are representative of the general compute patterns found in feature extractor networks targeted in this work. However, there are a few scenarios that can result in sub-optimal configurations. For example, consider the workload described in Figure 6 , accelerated on an architecture with 16KB activation SRAM. A greedy assignment will process the first layer in full, and then must fuse many subsequent layers together while the 15KB activation cannot yet be discarded. Instead, it can be more efficient to use a LB scheme starting with the previous layer, reducing the overall number of fused layers. This scenario can occur within MobileNet blocks which feature local upsampling and downsampling, specifically in cases when system activation cache is slightly greater than the maximum activation size. Our assignment algorithm checks for such cases by comparing the assignment to one with slightly more and slightly less capacity, and flags the assignment for review if the situation is detected.

Fig 7 shows an example assignment of processing schemes for each layer of EfficientNet-B3. The early layers use the LB schemes until intermediate activation sizes decrease sufficiently for layer-by-layer methods to become feasible. There are a few layers which use the least efficient method, Stream LB, because the sum of the weight and activation size outgrow the 400 kB capacity of the processor.

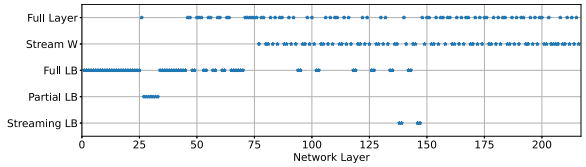


Fig. 7. Processing schemes for all layers of EfficientNet-B3 for an ANSA processor with 350 kB on-chip matrix cache and 50 kB vector cache.

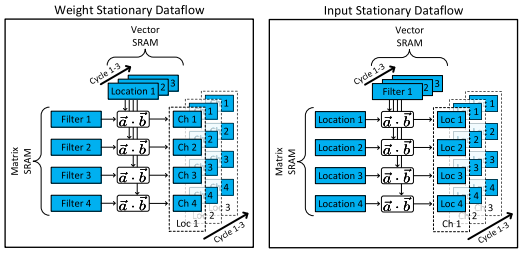


Fig. 8. Weight Stationary and Input Stationary mapping on a PE.

C. Hardware Support

ANSA treats each of these processing schemes as a special case of a full layer operation. During processing, the PEs execute an operation over a specified chunk of data. To switch between the layer-by-layer and the LB schemes, the shape of the input data is reduced from the full frame to the subset of activations corresponding to the LB scheme. In this way, reconfiguration between different processing schemes is limited to reconfiguration in address generation and a minimal number of control MUXes; therefore per-layer reconfiguration costs are minimized in our design flow. The majority of the work to coordinate the various schemes is accomplished at the compilation step when the configurations for each PE are generated, and baked statically into the instructions used to accelerate a given backbone. As this reconfiguration is restricted to a minimal set of MUXes being switched by a controller, this bandwidth is considered negligible and not estimated in our evaluations. In this way, ANSA is able to seamlessly switch processing schemes between layers with minimal dynamic processing required to resolve data movement. The reconfiguration cost of transitioning between NN backbones is also minimal, and amortized over many long frame processing windows.

VI. Flexible Dataflows

ANSA supports different mappings of the computation onto the VMMs, which we refer to as computation dataflows. These dataflows are defined by two aspects: the ordering of the weight and input activation loops during computation, and how summing across the channel dimension takes place between multiple PEs.

A. Dataflow Options

1) Weight Stationary or Input Stationary (WS/IS): The ordering of the weight and input activation loops correspond to how data is reused between cycles, as shown in Fig 8. In the weight stationary (WS) dataflow, weights are held in the PE's matrix SRAM between cycles while the input activations are streamed into the vector SRAM. PE outputs in the WS dataflow are generated in a channel major order, and can

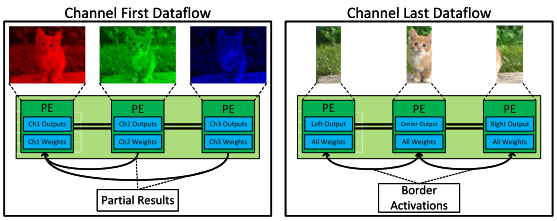


Fig. 9. Channel first and channel last mapping across PEs.

be used in subsequent layers without reshaping. Conversely, in the input stationary (IS) dataflow, input activations are held in the matrix SRAM. Outputs in the IS dataflow must be reshaped between layers; ANSA provisions buffering in the accumulation phase of computation to accommodate this.

2) Channel First or Channel Last (CF/CL): The accumulation of partial sums along the input channel dimension can take place in two steps. Within the VMM, the inputs are vectors of L_{vec} channels at a single spatial location. The VMM computes the L_{vec} partial sums and accumulates them as part of the dot product operation. However, in all but the first layers of popular CNN backbones, the number of channels $N_{channels}$ is large and we can expect that $N_{channels} > L_{vec}$ for practical choices of L_{vec}. As such, the accumulation along the channel dimension is done in multiple rounds. With multiple PEs this gives two options for where accumulation can take place, as depicted in Fig 9.

In the first case, each PE processes L_{vec} channels of the entire input, and intermediate results are summed across PEs. When $N_{PE} \cdot L_{vec} < N_{channels}$, the computation is spread between multiple rounds. We refer to this dataflow as Channel First (CF) as the channel accumulation is completed in one round if possible. Alternatively, the input activation can be divided spatially amongst PEs, with each PE responsible for all channels of its tile. Intermediate results are cached locally and accumulated over multiple rounds. We refer to this dataflow as Channel Last (CL) as the channel accumulation is spread over multiple rounds.

The CF and CL dataflows have different accumulation patterns. With the CF dataflow, partial results are transmitted over the NoC and summed at the destination PE. For the CL dataflow, only overlapping partial results are transmitted between neighboring PEs, reducing the amount and energy of the NoC communication. However, the CL dataflow replicates the weights on each PE, increasing the storage capacity.

B. Dataflow Selections

Combining the WS/IS and CF/CL options yields four dataflows. How well each layer performs under each dataflow depends on the layer's size, computational complexity, the dimensions and number of PEs, and the processing scheme.

Some properties of the network layer can give clear indications of the most suitable dataflow. For layers with small weights relative to the size of the input activations, common early in networks, WS is the most efficient as it maximizes weight reuse. Hardware structure also effects dataflow efficiency. For example, for VMM's with $N_{vec} = 8$, a 3×3.2D convolution does not fit evenly onto the VMM, necessitating either extra rounds or additional complexity in the accumulation step. Similarly, choosing between CF and CL depends both on layer and hardware properties. For example, to map a spatially small input onto a large number of PEs, it is infeasible to evenly tile the input without causing a significant amount of NoC traffic or redundant computation.

Switching dataflows between layers can cause complexity during the accumulation phase, as the destination of the outputs changes with dataflow. When switching between WS and IS dataflows, intermediate activations must be moved between matrix and vector SRAMs within PEs, respectively. Similarly, when switching from CL to CF dataflows, activations may need to be distributed across multiple PEs, incurring additional NoC traffic.

ANSA maintains flexibility by supporting all four dataflows. Each dataflow can be paired with each of the 5 processing schemes presented in Section V to create 20 possible combinations. At compile time the energy requirements of each of the combinations is estimated for each layer, including the latency and bandwidth required to switch methods. Dataflows are chosen to minimize the total energy consumption while satisfying the latency requirement.

VII. Results

We first discuss our methodology for evaluating ANSA. Next we compare the layer processing schemes with flexible dataflow options, and discuss the impact of key architectural features on performance and energy. We evaluate ANSA at a range of design points to compare ANSA to state-of-the-art accelerators across a range of processor sizes. Finally, we evaluate ANSA on the dynamic AR-Net workload. Given a set of hard constraints on the area of the processor and the distribution of backbone workloads, we discuss the resulting best configurations.

A. Evaluation Methodology

We estimate ANSA's area and energy based on a TSMC 28nm CMOS technology. The VMM and shuffle buffer were designed in RTL, then synthesized and placed and routed using the Cadence Innovus tool with a 500MHz clock. These designs were synthesized to support 3×3 convolutions, and designed with VMM vector dimensions of sizes 4, 8, 12, and 16; and matrix dimensions of sizes 4, 8, 16, 32, and 64. Area was extracted from these post-APR design layouts. A sample APR layout photo is shown in Figure 10. Power measurements for these designs were measured at several fixed input switching rates through back-annotated gate-level simulation, with and without the shuffle buffer. During simulation, the actual input switching rate for each VMM is computed, and power estimates are interpolated between those measured from these fixed-rate gate-level simulations. An SRAM compiler in the TSMC28nm HPC+ PDK was used to estimate the power and area of the matrix and vector SRAMs within the PE. These compiled SRAMs feature a sleep mode which is used to implement power gating for these devices. MRAM area and power estimates are extracted from the 22nm MRAM macro described in [32]. The NVM interfaces are 16 B wide and operate at 100 MHz, and follow a simple, SRAM like interface [32]. Weights are fetched to on-chip cache for processing. The NoC is modeled on a per-hop basis with a fixed per-hop energy and latency cost extrapolated

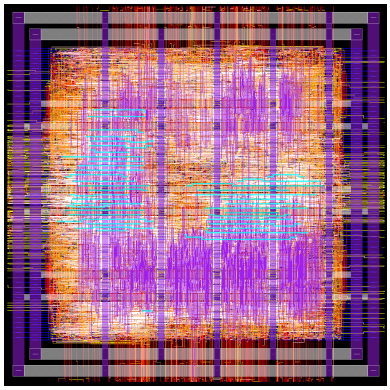


Fig. 10. Layout photo of a VMM supporting 8×8 matrices and 8×1 vectors with a Shuffle Buffer designed to accelerate 3×3 depthwise convolutions.

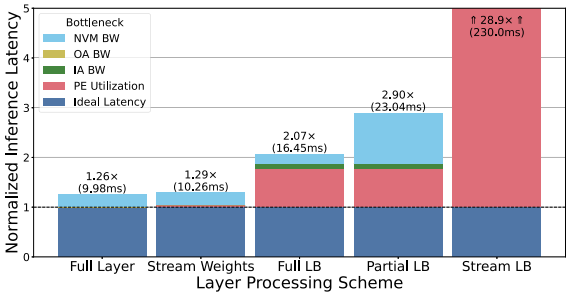


Fig. 11. Normalized latency for each processing scheme when running EfficientNet-B3. The latency of each method is broken down into portions attributed to each bottleneck.

from [41]. The transfer latency is estimated based on hop count accounting for congestion.

We wrote a mapping program to determine how each CNN layer is processed. This program takes the architecture definition and a CNN definition to assign the optimal layer processing scheme for each layer following the method described in Section V-B. Dataflows are then chosen to minimize total energy consumption while keeping total latency below the real-time threshold as described in Section VI-B.

After choosing processing scheme and dataflow for each layer, we can determine the total bandwidth for all interfaces for each layer. This allows the estimate of the computation time and energy consumption on a per-layer basis. This estimate includes the data shuffling between layers in the accumulation phase.

To estimate the effect of power gating on the energy consumption and latency, we first estimate the power consumption of the entire hardware where only a subset of the tiles or a subset of VMMs within a single tile is utilized. This first estimate includes the static power consumed by the unused portions of the hardware. We then simulate power gating these unused portions to obtain the final energy estimate.

It is important to note that we measure energy consumed per inference while running at a fixed frame rate of 30 FPS. This means that leakage from the device is considered for the full 33 ms window. Measuring energy in this manner gives a more accurate estimate of runtime energy consumption.

B. Comparing Processing Schemes

First we analyze the latency bottlenecks of each processing scheme. Fig 11 shows the computation latency when one processing scheme is exclusively used as in a common CNN accelerator. This comparison uses the smallest-size ANSA processor which supports full layer computation across the entire EfficientNet-B3. The processor consists of 4 tiles, each with 2 PEs of size $N_{vec} = 4$ and $L_{vec} = 8$.

1) Latency Analysis: The latency for each processing scheme in Fig 11 is broken down by parts attributed to potential system bottlenecks including external NVM bandwidth, on-chip Output Activation (OA) or Input Activation (IA) bandwidth, and PE utilization. These bottlenecks increase the processing latency above the ideal latency defined as the latency when all PEs are fully utilized every cycle. Imperfect mapping across PEs is reflected in the PE utilization bottleneck.

The inference latency when using only layer-by-layer processing schemes (Full Layer and Stream Weights) is close to ideal. For both the Full Layer scheme and the Stream Weights (SW) scheme, the only noticeable bottleneck is the external NVM bandwidth. The two layer-by-layer schemes are similar because the ANSA processor considered in Fig 11 has a large enough cache to store a full layer of weights. In a smaller ANSA design point where the cache size is limited, the Stream Weights scheme will incur higher latency due to accessing off-chip weights.

All three multi-layer LB processing schemes incur significantly more overhead than the layer-by-layer schemes. For the Full LB scheme, most of the latency bottleneck is caused by lower PE utilization. This is because a LB scheme partitions workloads into smaller chunks, making it more difficult to fully utilize the hardware. The Partial LB scheme has additional latency attributed to the increased NVM accesses. With Partial LB, the weights for a single layer are re-read from the NVM for each line of the input, saturating the comparably low-bandwidth NVM interface. Finally, the Stream LB scheme incurs an order of magnitude higher overhead compared to the other schemes. Due to the small number of activations stored on-chip and the higher NVM access latency, the Stream LB scheme suffers from extremely low PE utilization. Additionally, weights being reloaded for computing each output location places further strain on the NVM interface. This emphasizes that the Stream LB scheme is a last resort and used only when absolutely necessary.

2) Energy Analysis: A breakdown of the per-inference energy consumption for each processing scheme is shown in Fig 12. The same hardware configuration is used as in Figure 11, and energy is measured at 30 FPS for all schemes except Stream LB which cannot meet real-time and is measured at a slower rate.

Both of the layer-by-layer schemes, Full Layer and Stream Weights, perform nearly identically for this ANSA configuration. The majority of the energy is attributed to the SRAM bandwidth, SRAM static energy, and NVM bandwidth, with only 9.9% of the energy used for compute. Comparatively, The multi-layer LB schemes use significantly more energy. For the Full LB scheme, SRAM access energy is much higher due to lower data reuse. The Partial LB scheme uses even more energy for NVM accesses due to repeatedly reloading weights. Finally, the Stream LB scheme uses much more energy than

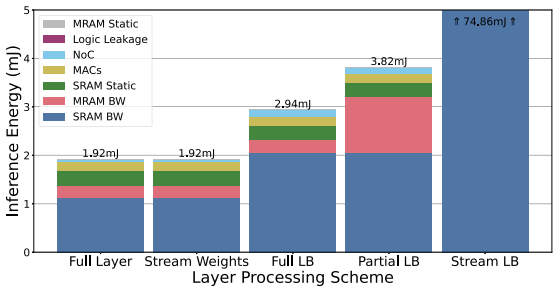


Fig. 12. Energy breakdown of EfficientNet-B3 inference when using each processing scheme. The static MRAM energy consumption is too small to be visible on the charts.

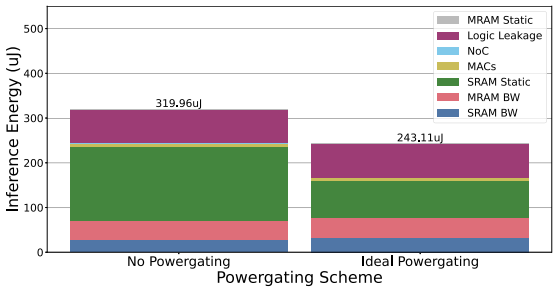


Fig. 13. Energy breakdown of MobileNetV2 inference on a processor optimized for EfficientNet-B3 inference, with no power gating and with ideal power gating enabled. In ideal power gating, static power for unused compute and SRAM components is assumed to be 0.

the other schemes, the majority of which is attributed to SRAM bandwidth.

An important assumption of the greedy algorithm for processing scheme assignment presented in Section V-B is that there is a fixed preferred order of processing schemes. With this analysis, we can see empirically that this assumption holds. Both Fig 11 and Fig 12 show a clear trend that each successive scheme from the Full Layer scheme to the Stream LB scheme has higher energy consumption and higher latency than the previous.

C. Power Gating

We also demonstrate the capability of our design to power-gate unused computational resources and how this impacts performance on dynamic workloads in Figure 13. In this analysis, we first optimize an architecture configuration for minimum inference energy on EfficientNet-B3. We find an architecture with 1 Tile and 4 PEs/Tile of size $N_{vec} = 32$, $L_{vec} = 16$, and 545.4kB of on-chip SRAM to achieve this. We then evaluate the inference energy of MobileNetV2 on this architecture, with and without power gating enabled during compilation.

The takeaway from this analysis is two-fold. We first observe that enabling power gating can improve per-inference energy by up to 1.316× in this situation. We further note that almost all of the energy savings come from reducing the total SRAM static energy draw during the inference period. As dynamic workloads are designed to have a large disparity in size between their largest and smallest models, devices with compute capacity sufficient for these large networks may have idle resources and particularly idle SRAM when running smaller models, as seen here. As these static power draws may be significant depending on the workload distribution, we find that power gating is a key technique to enable energy efficient support for dynamic algorithms.

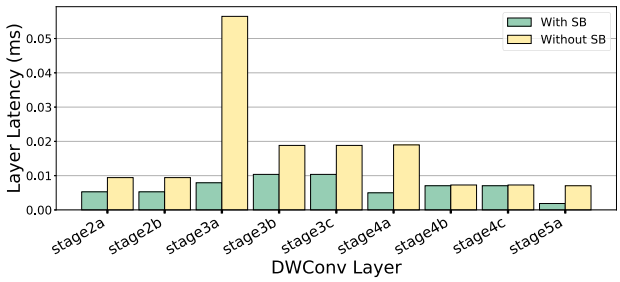


Fig. 14. Latency difference on select layers of EfficientNet-B3 with and without the shuffle buffer (SB) enabled.

D. Shuffle Buffer

Here we analyze how the shuffle buffer reduces overall inference latency by increasing PE utilization on depth-wise layers. We use the same ANSA configuration as in Figure 11, but with VMMs of size $N_{vec} = 16$, $L_{vec} = 32$. Fig 14 shows the latency of the first 9 depth-wise layers of EfficientNet-B3 with and without the shuffle buffer. The latency savings of the shuffle buffer depend on the layer characteristics, with stage3a saving over 5× latency, while minimal effect on stage4b and stage4c. Using the shuffle buffer, the total CNN inference latency is reduced from 17.7 ms to 14.7 ms (17%) for EfficientNet-B3.

The impact of the shuffle buffer on per-inference energy consumption is minimal when measured at a fixed frame rate for EfficientNet-B3. This result is due to two factors. Firstly, as the depth-wise layers constitute a relatively small portion of the total compute, the absolute bandwidth reduction is relatively small. Secondly, because the energy is measured at a fixed frame rate, the saving in latency does not reduce the leakage energy consumption. The area impact of the shuffle buffer is also relatively low, constituting below 6.7% of total PE area depending on VMM dimensions. However, for large models, the latency savings from the shuffle buffer can be significant enough to enable real-time operation.

E. Fixed Workload Analysis

We now demonstrate how each feature of ANSA impacts overall performance. We will present the effects of each architectural component at a set of design points defined by a limit on the processor area. In addition, we also perform an ablation study to compare our architecture with other competitive works lacking in specific features; specifically, we model NVDLA [19] by analyzing only single core, full layer models with Weight Stationary / Channel Last dataflows; and [7] by analyzing models with only Line Buffer based processing schemes and without Input Stationary dataflows or the shuffle buffer. These architectures were modelled using the same framework as ANSA to normalize the comparison. Eyeriss v2 [4] and Sony's stacked CNN processor [5] are also included in comparison. We note that while many prior works have achieved high performance on comparable workloads, these works were often evaluated at a single, large-area design point, and would therefore make a poor comparison to this work, which targets a different design point. For this reason,

TABLE II

DESIGN POINTS USED TO COMPARE THE ARCHITECTURE PERFORMANCE AS IT SCALES IN SIZE. THE AREA OF THE REFERENCED ACCELERATORS ARE SCALED FOR COMPARISON

Design Point	Area	Reference
XS	$0.15mm^2$	$1.73\mu m$ pixel advanced CMOS sensor, 224^2 resolution
S	$0.35mm^2$	$2.64\mu m$ pixel global shutter CMOS sensor, 224^2 resolution
M	$0.70mm^2$	2x down sampled global shutter CMOS sensor, 316^2 resolution
L	$1.2mm^2$	Common small CNN accelerators: [13], [42], [43]
XL	$2.0mm^2$	Common medium CNN accelerators: [12], [15], [44], [45]
XXL	$5.0mm^2$	Common large CNN accelerators: [46]–[48]

TABLE III

LOWEST ENERGY CONFIGURATIONS AT EACH DESIGN POINT

Design Point	N_{vec}	L_{vec}	PEs/ Tile	Tiles	MACs/ Cycle	Matrix SRAM (Per-PE)	Vector SRAM (Per-PE)	Total SRAM (Full-Chip)
XS	4	8	12	1	384	255 B	3.9 kB	50 kB
S	8	4	28	1	896	1.2 kB	2.6 kB	107 kB
M	16	12	8	1	1536	3.0 kB	33 kB	284 kB
L	32	12	4	1	1536	25 kB	127 kB	608 kB
XL	32	16	2	1	1024	81 kB	503 kB	1.2 MB
XXL	4	16	1	4	256	353 B	775 kB	3.1 MB

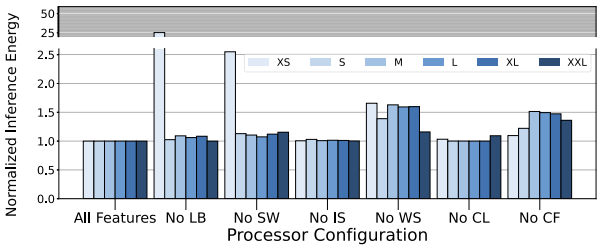


Fig. 15. Inference energy on EfficientNet-B3 for multiple configurations at each design point (LB: Line Buffer, SW: Stream Weights). Darker shades represent larger design points. Energy is normalized to ANSA's full architecture energy at each size. Note the change in scale.

we focus our comparisons with works that scale in order to highlight the features of our design.

1) ANSA Design Points: The specified ANSA design points are shown in TABLE II. Their area ranges from 0.15 mm^2, representing the most constrained near-sensor processor, to 5.0 mm^2 for larger designs comparable to existing popular CNN accelerators. For each design point, we consider all configurations of ANSA within 5% of that area by sweeping over the VMM shape, PE memory capacity, number of PEs per tile, and number of tiles. Each CNN workload is then mapped to each of the designs to estimate inference energy and latency. From the design space exploration, the lowest energy design that satisfies the real-time latency requirement is chosen as the best architecture configuration at that design point. The resulting configurations are shown in TABLE III.

The lowest energy configurations for ANSA at these design points tend to have one or a few tiles with either many PEs per tile, or a few PEs with large VMMs. As the number of NVM interfaces is tied to the number of tiles, this suggests that only one NVM interface is sufficient to support computation.

2) Architectural Feature Analysis: In order to assess the impact of each architectural feature on performance, we perform the same sweep while disabling that feature. We perform the search in this manner in order to not exclude an architecture configuration which is able to compensate for a loss of performance due to removing that architectural feature. Here, we consider disabling support for one processing scheme or dataflow at a time.

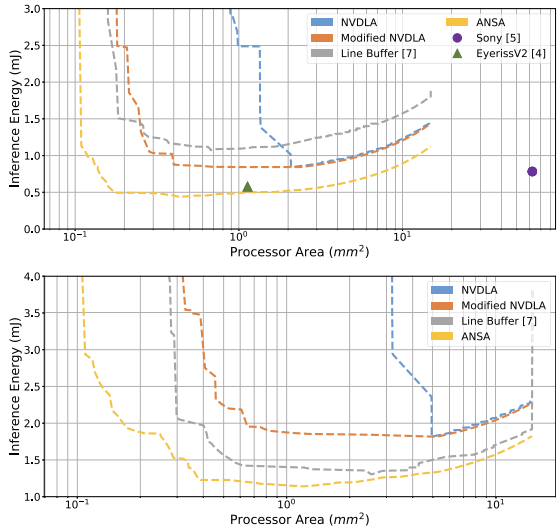


Fig. 16. Minimum energy consumption for processing various backbones on ANSA and compared works at a range of processor sizes. Top: Comparison using the MobileNet backbone with other scalable adaptive works [4] and [5]. Bottom: Comparison using the EfficientNet-B3 backbone. Note the change in vertical range between (a) and (b).

Fig 15 shows the energy consumption of the best configuration at each processor size. Energy is normalized to the full ANSA architecture at each size (see Table III). We can see that the LB schemes are key to enabling processing at the smallest design point. At this size there is not enough SRAM capacity to fit most of model layers, and the LB schemes are needed to reduce the size of the input activations in memory. There is a similar case when the Stream Weights scheme is disabled, though to a lesser extent.

Disabling support for the IS dataflow has minimal impact on the energy consumption. However, disabling support for the WS dataflow has a substantial impact across all but the largest design point. This suggests that without a sufficiently large on-chip memory, the WS dataflow is very important for processing efficiency. Similarly, disabling the CL dataflow has less impact on energy than disabling the CF dataflow, as the CL dataflow is rarely used due to its redundant weight storage requirement.

3) Comparisons: In Fig 16, we show a comparison of ANSA with prior works at a range of processor sizes. We perform a design space sweep of the architecture configurations for each architecture on multiple workloads. We evaluate on the MobileNetV1 workload in the top half of Fig 16 to enable fair comparison with other designs, and we also evaluate our design on EfficientNet-B3 in the bottom half to demonstrate performance across a range of workload sizes. In general, the NVDLA architecture performs poorly at most small processor sizes. This is largely because NVDLA is a single core architecture, whose utilization is reduced by the limited space for on-chip caching, resulting in high latency and high energy consumption. For fairness of comparison, we also evaluate a *Modified NVDLA* architecture using multiple NVDLA cores. We also consider an architecture which closely matches the design presented in [7], with LB only processing schemes.

As shown in Fig 16, ANSA achieves competitive performance with large processor configurations such as Eyeriss v2 [4]. ANSA also achieves superior energy efficiency

to [7] due to the introduction of flexible processing schemes and dataflows. In particular, ANSA maintains efficient processing even at much tighter area constraints, due to its lesser reliance on on-chip cache to enable efficient computation. This scaling is more pronounced with larger and more memory intensive workloads, as seen in Fig 16 when evaluating with EfficientNet-B3. The presence of many large intermediate activations prevents NVDLA architectures from efficiently processing at small sizes, while the presence of many smaller layers prevents [7] from operating at high energy efficiency. The adoption of both techniques in ANSA allows for efficient processing at smaller scales.

F. Dynamic Workload Analysis

In this section we consider ANSA's performance across multiple workloads, a situation common in AR/VR applications where the CNN workload can vary between frames. AR-Net has four backbone networks[1] (TABLE I) which operate on different image resolutions. AR-Net can choose to skip some frames' computation. The smallest backbone, in this case MobileNetV2, is also run for all non-skipped frames. We include this inference energy in the total for the three EfficientNet backbones.

Our goal is to minimize the average runtime inference energy for these dynamic workloads. This will be dependent on the frequency each backbone is used. We consider two representative usage frequency distributions. In the equal distribution, each of the four backbones has equal chance of being used and no frames are skipped. In the light distribution, EfficientNet-B3, EfficientNet-B1, EfficientNet-B0, MobileNet, and frame skips are used in 1%, 5%, 15%, 70%, and 9% of frames, respectively.

We perform a design space sweep of the architecture configurations for each of the backbones and each workload. In this sweep, only configurations which can run all backbones are considered. Each Pareto curve of energy versus area was extracted, as shown in Fig 17. This curve provides insight into how the energy consumption changes with area. Overall, the curves have a bathtub shape with a region in the center along which the energy consumption is nearly constant. For large designs, energy consumption is dominated by static power draw of the SRAMs. For small designs, energy consumption is dominated by NVM access energy due to limited on-chip cache and decreased data reuse.

For each workload distribution, the lowest-energy configuration was found. The energy consumption for each backbone with this minimum configuration is annotated in Fig 17 by each vertical set of four points. Note that some workloads may be computed above the Pareto minimum for each distribution, as the architecture must optimize for each workload simultaneously according to their useage frequency. Points for ANSA are shown alongside three other architectures: LB only processing scheme from [7], the modified NVDLA architecture, and DepFiN [49].

[1]We analyzed both versions of AR-Net with ResNet and EfficientNet backbones, however we focus here on the results from the EfficientNet backbones since they are better suited for edge computing. The results and takeaways from the ResNet are similar.

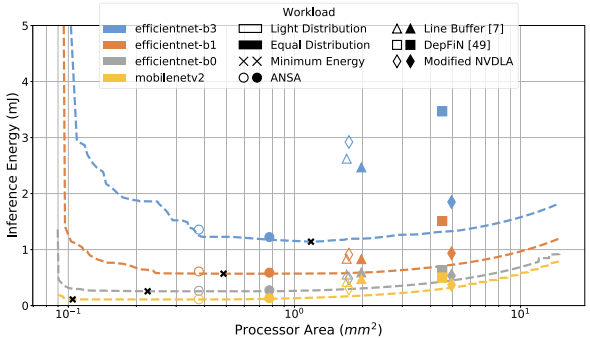


Fig. 17. Pareto curves for the energy consumption of processing each network to meet 30 FPS performance versus area. The black **x** on each curve shows the minimum energy for that backbone.

TABLE IV
OPTIMAL ANSA CONFIGURATIONS FOR EACH WORKLOAD DISTRIBUTION

Workload	N_{vec}	L_{vec}	PEs/ Tile	Tiles	MACs/ Cycle	Matrix SRAM (Per-PE)	Vector SRAM (Per-PE)	Total SRAM (Full-Chip)	Area (28nm) (mm^2)	Average Power (mW)	Peak Power (mW)
Equal	16	16	8	1	2048	978 B	35 kB	284 kB	0.77	16.624	36.69
Light	4	16	16	1	1024	452 B	8 kB	133 kB	0.38	4.860	40.697

TABLE V
COMPARISON OF ANSA AND OTHER ARCHITECTURES

Architecture	Workload	Area (mm^2)	SRAM	Avg Inference Energy
Full Architecture	Light	0.378	133 kB	**0.162 mJ**
	Equal	0.775	284 kB	**0.554 mJ**
LB Only [7]	Light	1.702	1.02 MB	**0.447 mJ (2.76 ×)**
	Equal	1.976	1.02 MB	**1.091 mJ (1.97 ×)**
DepFiN [49]	Light	4.500	1.03 MB	**0.555 mJ (3.42 ×)**
	Equal			**1.528 mJ (2.76 ×)**
NVDLA [19]	Light	4.952	3.02 MB	**0.405 mJ (2.50 ×)**
	Equal			**0.922 mJ (1.66 ×)**
Mod. NVDLA	Light	1.741	1.02 MB	**0.362 mJ (2.23 ×)**
	Equal	4.952	3.02 MB	**0.922 mJ (1.66 ×)**

For the equal workload, the lowest-energy ANSA processor size is close to that of the medium design point in TABLE II at 0.78 mm^2. For this configuration, all operating points are very close to the Pareto minimum. For the light workload, the lowest-energy processor size is reduced to 0.38 mm^2. All but the heaviest backbone can be run at a point near the Pareto minimum. For this configuration, sacrificing inefficient compute on the heaviest backbone is acceptable due to its low usage frequency. The specific architecture configurations for the two workloads are shown in TABLE IV.

All compared architectures have higher area and average inference energy compared with ANSA as shown in TABLE V. ANSA achieves a 42% lower energy consumption with a 84% reduction in area over to the modified NVDLA architecture for the equal workload. ANSA uses the flexibility in processing schemes and dataflows to vastly reduce the required area of the processor while maintaining efficiency. The addition of power gating moves the operating points even closer to the Pareto minimum, reducing energy by 11.3% and 10.2% for the equal and light workloads, respectively.

TABLE IV details the ANSA configuration optimized for the equal and light workloads. As expected, the light workload yields a smaller processor in terms of SRAM capacity and overall chip area. The equal workload design features higher compute in the form of a few large PEs, while the light workload uses many smaller PEs. Both designs cluster all their PEs in a single tile, suggesting only one NVM interface is sufficient to support real-time computation in both cases. For

both configurations, the vast majority of the on-chip SRAM is dedicated to the vector SRAM. This is likely because the Stream Weights processing scheme with WS dataflow is used for the largest layers which allows for a single filter to be cached on the chip at a time.

VIII. Conclusion

This work presents ANSA, a near-sensor processor for AR/VR devices. ANSA targets stacked-chip integration with the sensor and NVMs to achieve efficient computation over a range of dynamic workloads and at compact processor size constraints. We present ANSA's tile-based architecture alongside the processing schemes and computational dataflows it supports. We demonstrate how when combined with layer-by-layer methods the multi-layer LB technique can enable efficient CNN computation on memory-constrained processors. Many of the presented techniques have the potential to improve the performance of existing architectures with small adjustments to support multiple processing schemes and dataflows. Our architecture demonstrates the benefits of dynamically switching between processing schemes and dataflows on a per-layer basis to support efficient CNN processing on resource-constrained systems, as well as the ability to maintain processing efficiency through these schemes for very small area constraints, achieving 2.76× greater energy efficiency than [7] at 4.5× smaller area. We also demonstrate how combined with power gating this enables our architecture to adapt to the dynamic workloads at runtime, demonstrating a 40% and 55% reduction in energy consumption, respectively, on two AR-Net dynamic workloads.

References

[1] 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.*, New York, NY, USA, 2014, pp. 1–7, doi: 10.1145/2637166.2637230.

[2] *Introducing Meta Quest Pro, An Advanced VR Device for Collaboration and Creation*. Accessed: Oct. 28, 2022. [Online]. Available: https://www.oculus.com/blog/meta-quest-pro-price-release-date/

[3] 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, pp. 10.5.1–10.5.4.

[4] 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. Emerging Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, Jun. 2019.

[5] R. Eki et al., "A 1/2.3 inch 12.3 Mpixel with on-chip 4.97 TOPS/W CNN processor back-illuminated stacked CMOS image sensor," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 64, Feb. 2021, pp. 154–156.

[6] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.

[7] K. Goetschalckx and M. Verhelst, "Breaking high-resolution CNN bandwidth barriers with enhanced depth-first execution," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 323–331, Jun. 2019.

[8] W. Hua, Y. Zhou, C. De Sa, Z. Zhang, and G. E. Suh, "Boosting the performance of CNN accelerators with dynamic fine-grained channel gating," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2019, pp. 139–150.

[9] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "MCUNetV2: Memory-efficient patch-based inference for tiny deep learning," 2021, *arXiv:2110.15352*.

[10] H. Wu, Q. Liu, and X. Liu, "A review on deep learning approaches to image classification and object segmentation," *Comput. Mater. Contin*, vol. 1, no. 1, pp. 1–5, 2018.

[11] C. Kawatsu et al., "Gesture recognition for robotic control using deep learning," in *Proc. NDIA Ground Vehicle Syst. Eng. Technol. Symp.*, 2017, pp. 1–7.

[12] 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.

[13] 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.

[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] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "Envision: A 0.26-to-10 TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28 nm FDSOI," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 246–247.

[16] Y. S. Shao et al., "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2019, pp. 14–27.

[17] 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.

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

[19] NVIDIA. (2018). *NVIDIA Deep Learning Accelerator (NVDLA)*. [Online]. Available: http://nvdla.org/

[20] F. Zhang et al., "Mediapipe hands: On-device real-time hand tracking," 2020, *arXiv:2006.10214*.

[21] S. Choi, J. Lee, K. Lee, and H.-J. Yoo, "A 9.02 mW CNN-stereo-based real-time 3D hand-gesture recognition processor for smart mobile devices," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 220–222.

[22] J. Chen, J. Meng, X. Wang, and J. Yuan, "Dynamic graph CNN for event-camera based gesture recognition," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5.

[23] C. Choy, J. Gwak, and S. Savarese, "4D spatio–temporal ConvNets: Minkowski convolutional neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 3075–3084.

[24] K. Tateno, F. Tombari, I. Laina, and N. Navab, "CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6243–6252.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, *arXiv:1512.03385*.

[26] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

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

[28] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.

[29] Y. Meng et al., "AR-Net: Adaptive frame resolution for efficient action recognition," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 86–104.

[30] D. U. Lee et al., "A 1.2 V 8 Gb 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.

[31] 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.

[32] W. J. Gallagher et al., "22 nm STT-MRAM for reflow and automotive uses with high yield, reliability, and magnetic immunity and with performance and shielding options," in *IEDM Tech. Dig.*, Dec. 2019, pp. 2.7.1–2.7.4.

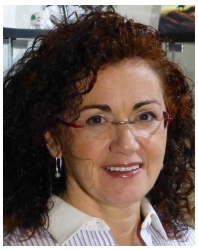
[33] Y. Shih et al., "A reflow-capable, embedded 8 Mb STT-MRAM macro with 9 ns read access time in 16 nm FinFET logic CMOS process," in *IEDM Tech. Dig.*, Dec. 2020, pp. 11–14.

[34] Y.-H. Chen et al., "A 16 nm 128 Mb SRAM in high-k metal-gate FinFET technology with write-assist circuitry for low-VMIN applications," *IEEE J. Solid-State Circuits*, vol. 50, no. 1, pp. 170–177, Sep. 2015.

[35] 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.
[36] Omnivision. *1 Megapixel and Below*. Accessed: Nov. 23, 2021. [Online]. Available: https://www.ovt.com/image-sensors/1-megapixel-and-below
[37] G. Park et al., "A 2.2 μm stacked back side illuminated voltage domain global shutter CMOS image sensor," in *IEDM Tech. Dig.*, Dec. 2019, pp. 16.4.1–16.4.4.
[38] Sony. *Sony to Release World's First Intelligent Vision Sensors With AI Processing Functionality*. Accessed: Nov. 23, 2021. [Online]. Available: https://www.sony.net/SonyInfo/News/Press/202005/20-037E/
[39] T. Chen et al., "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 269–284, 2014.
[40] J. Fowers et al., "A configurable cloud-scale DNN processor for real-time AI," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 1–14.
[41] D. Sanchez, G. Michelogiannakis, and C. Kozyrakis, "An analysis of on-chip interconnection networks for large-scale chip multiprocessors," *ACM Trans. Archit. Code Optim.*, vol. 7, no. 1, pp. 1–28, May 2010, doi: 10.1145/1736065.1736069.
[42] J. Yan, S. Yin, F. Tu, L. Liu, and S. Wei, "GNA: Reconfigurable and efficient architecture for generative network acceleration," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2519–2529, Nov. 2018.
[43] D. Im, D. Han, S. Choi, S. Kang, and H.-J. Yoo, "DT-CNN: An energy-efficient dilated and transposed convolutional neural network processor for region of interest based image segmentation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 10, pp. 3471–3483, Oct. 2020.
[44] H. Mo et al., "A 28 nm 12.1 TOPS/W dual-mode CNN processor using effective-weight-based convolution and error-compensation-based prediction," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 64, Feb. 2021, pp. 146–148.
[45] Z. Yuan et al., "Sticker: A 0.41–62.1 TOPS/W 8 bit neural network processor with multi-sparsity compatible convolution arrays and online tuning acceleration for fully connected layers," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 33–34.
[46] J.-F. Zhang, C.-E. Lee, C. Liu, Y. S. Shao, S. W. Keckler, and Z. Zhang, "SNAP: A 1.67–21.55 TOPS/W sparse neural acceleration processor for unstructured sparse deep neural network inference in 16 nm CMOS," in *Proc. Symp. VLSI Circuits*, Jun. 2019, pp. C306–C307.
[47] I. Miro-Panades et al., "SamurAI: A 1.7 MOPS-36 GOPS adaptive versatile IoT node with 15,000× peak-to-idle power reduction, 207 ns wake-up time and 1.3 TOPS/W ML efficiency," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2020, pp. 1–2.
[48] N. Shah, L. I. G. Olascoaga, S. Zhao, W. Meert, and M. Verhelst, "PIU: A 248 GOPS/W stream-based processor for irregular probabilistic inference networks using precision-scalable posit arithmetic in 28 nm," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2021, pp. 150–152.
[49] K. Goetschalckx and M. Verhelst, "DepFiN: A 12 nm, 3.8 TOPs depth-first CNN processor for high res. Image processing," in *Proc. Symp. VLSI Circuits*, Jun. 2021, pp. 1–2.

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 Meta Reality Labs, Redmond, WA, 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.

Jack Erhardt (Member, IEEE) received the B.S. degree in electrical engineering from the University of Minnesota, Minneapolis, MN, USA, in 2020, and the M.S.E. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2022, where he is currently pursuing the Ph.D. degree. His current research interests include edge compute systems and hardware-software codesign for low power applications.



Barbara De Salvo (Fellow, IEEE) is currently the Director of Research with the Meta Reality Labs Research, responsible for Silicon Strategy and Foundry Engineering. Before joining Meta, she was a Chief Scientist and the Deputy Director of CEA-LETI, driving the path-finding strategy, in 2019. From 2013 to 2015, she was the Manager and a Visiting Scholar at IBM-Albany-NY in the frame of the sub-10 nm CMOS International Technology Alliance, where several of her research works have led to product technologies for novel logic ICs (as silicon-on-insulator, finfet, and stacked nanowire technology platforms). In CEA-LETI, she founded and led the advanced memory technology division (2008–2013), where she promoted the introduction of disruptive memory technologies, such as phase-change memories, resistive oxide-based and conductive-bridge memories. She pioneered neuromorphic hardware solutions based on emerging technologies for ultra-low-power cognitive systems. She has authored more than 350 referred articles, ten book chapters, a monography on silicon non-volatile memories edited by Wiley and Sons. She is an Active Member of the IEEE Women in Engineering network. She is currently serving as the General Chair for IEEE IEDM 2022, as well as the Chair of the IEEE Corporation Award Committee.

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 the Meta 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 with 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 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 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 an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS (2013–2015) and the IEEE IEEE RANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS (2014–2015). Since 2015, he has been an Associate Editor of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS.