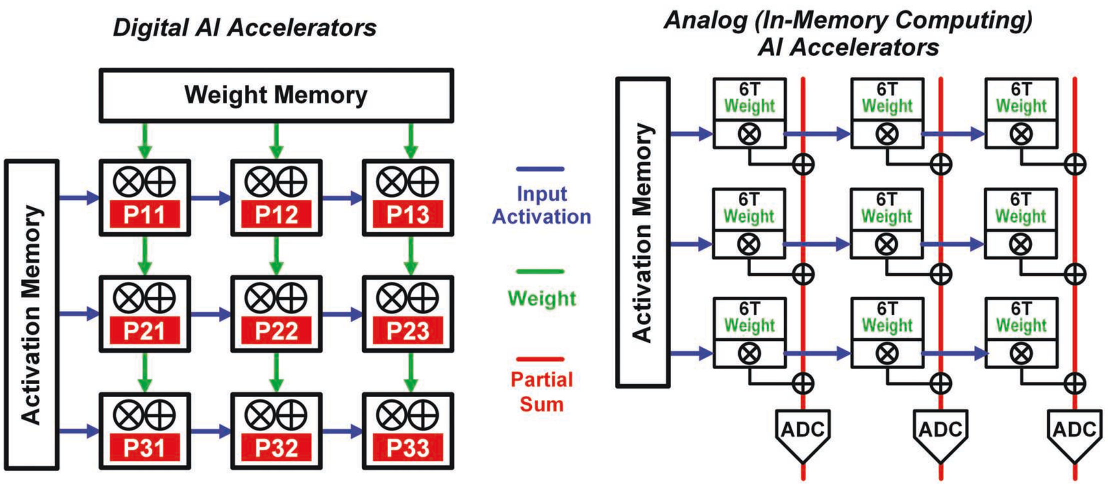


Jae-sun Seo, Jyotishman Saikia, Jian Meng, Wangxin He, Han-sok Suh, Anupreetham, Yuan Liao, Ahmed Hasssan, and Injune Yeo

Digital Versus Analog Artificial Intelligence Accelerators

Advances, trends, and emerging designs

For state-of-the-art artificial intelligence (AI) accelerators, there have been large advances in both all-digital and analog/mixed-signal circuit-based designs. This article presents a practical overview and comparison of recent digital and analog AI accelerators. We first introduce hardware-efficient AI algorithms, which have been targeted for many AI hardware designs. Next, we present a survey of 1) all-digital AI accelerators, including designs with new dataflow, low precision, and sparsity, and 2) analog/mixed-signal AI accelerators featuring switch-capacitor circuits and in-memory computing (IMC) with ADCs. Recent advances of AI accelerators in both digital and analog design approaches are summarized, and emerging AI accelerator designs are discussed.

Digital Object Identifier 10.1109/MSSC.2022.3182935
Date of current version: 24 August 2022

Background

AI and deep neural network (DNN) algorithms have been very successful across many tasks, including computer vision, natural language processing, and medical diagnosis. While AI algorithms have largely improved over recent years to achieve high accuracy with a smaller number of operations and fewer weights, state-of-the-art ImageNet algorithms still require billions of MAC operations per single image inference and storage for millions of weights [1].

It is challenging to map such complex algorithms onto various hardware platforms, especially with divergent power and area constraints, from embedded systems to mobile smartphones and wearable devices. To that end, both hardware-efficient algorithms and custom AI accelerators that can efficiently map such algorithms are required:

To benchmark AI inference accelerators, there are several well-known metrics that have been used throughout the literature [2], such as power, throughput, and energy efficiency. Power is the rate at which you do computations times the energy per inference. The energy per inference for a given application is the ultimate metric we care about (1). The energy per inference has two components: 1) how many operations you have per inference and 2) the energy per such operation. In the machine learning community, there have been many works on tuning the first knob by using a smaller number of necessary operations per inference, compressing the model via exploiting sparsity, and so on. The hardware community has presented various methods for tuning the second knob, with new hardware fabrics and process technologies.

$$\underbrace{\frac{\textit{Energy}}{\textit{Inference}}}_{\substack{\text{Ultimate}\\ \text{Care About}}} = \underbrace{\frac{\textit{Number of Operations}}{\textit{Inference}}}_{\substack{\text{Compress Model}\\ \text{Exploit Sparsity}}} \times \underbrace{\frac{\textit{Energy}}{\textit{Operation}}}_{\substack{\text{Improve Hardware Fabric}\\ \text{Process Technology}}} \qquad (1)$$

Hardware-Efficient Algorithms

Hardware-efficient algorithms include approaches such as quantization [3], pruning [4], compact model transformation [5], tensor decomposition [6], and so on. Arguably, quantization and pruning have been the two main approaches for making AI algorithms hardware efficient. Quantization aims to use fewer bits per each weight and/or activation, and this is done by discretizing each weight to a finite number of specific values. Pruning exploits the fact that zero weights/activations do not change the MAC result and the final DNN result. To that end, we can prune out zero weights and activations without affecting the DNN outcome. By using fewer total weights, the storage requirement is largely alleviated. Both of these techniques can significantly reduce the area and energy of the resultant AI accelerator.

Low-Precision Quantization

Conventionally, DNNs are first trained with 32- or 64-b floating-point precision, and then the trained DNN is quantized with fixed-point precision. For posttraining quantization, dynamic quantization [7] or hardware-aware quantization [8] can be performed, where a different number of integers versus fractional bits are used for different layers within the same model to improve the accuracy-versus-precision tradeoff. However, the DNN accuracy of such posttraining quantization works sharply worsens for <6-b precision.

To achieve high DNN accuracy with even lower precision values, in-training quantization, or quantization-aware training [9], has been proposed. The key idea is that the target low precision for quantization should be incorporated in the DNN training process so that the DNN weights can be trained, reflecting the low-precision quantization. In particular, during the forward/backward passes of DNN training, the same target low fixed-point precision for inference will be used, while, for the weight update, high floating-point precision will still be used. Since the forward pass of DNN training is the same as DNN inference, after training is complete, the DNN inference can achieve high accuracy with the target low-precision quantization. The extreme case of low-precision quantization is binarizing both weights and inputs (1-b precision), where the multiplication between weights and inputs become a simple XNOR operation, and accumulate becomes the bitcount operation of XNOR outputs [10]. A large amount of memory as well as computation can be saved.

In-training quantization has been further optimized with wider DNN models in [11], where it has been reported that wider DNNs perform in-training quantization effectively with lower precision. By trading off a higher number of raw compute operations with an aggressively reduced precision of weights and activations for isoaccuracy, the overall compute cost becomes lower with wider DNN models.

In [12], for various DNNs for ImageNet, low-precision quantization from 8, 4, and down to 2 b has been evaluated. In another recent work [13], the accuracy of 4-b MobileNet models has been optimized. As shown in Figure 1, the variants of ResNet and VGG models for ImageNet data set show reasonable accuracy down to 3–4-b precision. For more compact models, such as MobileNet and SqueezeNet, it is more difficult to achieve low precision, as noticeable accuracy degradation is observed below 4-b precision.

Pruning

The conventional pruning method is magnitude-based pruning [4], where the weights whose magnitudes are close to zero are pruned out from the DNN model. As we prune the

DNNs, the DNN accuracy needs to be carefully monitored. One of the recent works that further improved the accuracy-versus-sparsity tradeoff is lookahead pruning [14], which extends the layerwise approximation of magnitude-based pruning to a block of layers by looking ahead at the impact of pruning on neighboring layers. While high sparsity can be achieved, these elementwise sparsity techniques require indexes to store the location of the remaining nonzero weights after pruning, and this can be a larger burden for low-precision weights. In addition, the sparsity with elementwise pruning is typically formed in an irregular manner, which can hurt the memory access and efficient hardware acceleration.

To address such challenges of elementwise pruning, structured pruning has been proposed [15]–[17], where weights are pruned in rowwise, columnwise, or blockwise manner. Such structured pruning largely reduces the index storage since the index can be shared among the row/column/block structure, and it also enhances regular memory access and hardware acceleration. If we apply rowwise or columnwise structured sparsity for general matrix multiplication, we can remove the entire rows or columns, effectively reducing the size of the matrix. Group least absolute shrinkage and selection operator (LASSO) regularization was performed in [15] to also remove 2D filters in convolutional neural networks (CNNs) to achieve channelwise or filterwise structured sparsity.

When we try to naively combine structured pruning with low-precision quantization, nonnegligible accuracy loss occurs because group LASSO pruning acts like normal regularization, forcing all weights toward smaller values. A weight penalty clipping technique with a self-adapting threshold was presented in [18]. Essentially, if the overall group weight is large in terms of L2 norm, the group is not pruned by group LASSO, and only when the group weight is small are the weights pruned in a groupwise manner. By optimally combining large structured compression with ternary weights for DNNs, a lower accuracy drop was achieved.

All-Digital AI Accelerators

To efficiently execute the complex AI algorithms that accompany the quantization and/or pruning techniques discussed in the "Hardware-Efficient Algorithms" section, many custom ASIC accelerators have been presented in the literature. These include chips from major industry companies (Google TPU, Tesla Dojo, and so on), a number of recent start-up companies (Sambanova, Graphcore, Groq, and so on), and new prototype chips from many research groups in academia.

MAC operations occupy >90% of DNN workloads. To compute such a large number of MAC operations, all-digital DNN accelerators typically employ a large number of parallel processing engines (PEs), where each PE performs one or several MAC computations. Considering different memory hierarchies, off-chip DRAM access consumes higher energy by a couple orders of magnitude than MAC computation or local register file access. To that end, DNN accelerators are designed to support specialized processing dataflows that leverage this memory hierarchy.

Many digital AI accelerators have commonly employed a 2D systolic array of PEs or MAC engines with specific dataflows to reuse input activations and/or weights while keeping the weight, partial sum, or input stationary in each PE (Figure 2). In the weight-stationary dataflow (WSD), each filter weight remains stationary, input activations get loaded and shifted horizontally, and partial sums are accumulated vertically. The static weights can be reused and computed with multiple pixels in the same feature map or with different feature maps. In the output-stationary dataflow (OSD), the accumulation of each output pixel stays stationary in each PE, while the input activations get loaded and shifted horizontally, and the weights get loaded and shifted vertically across the PEs.

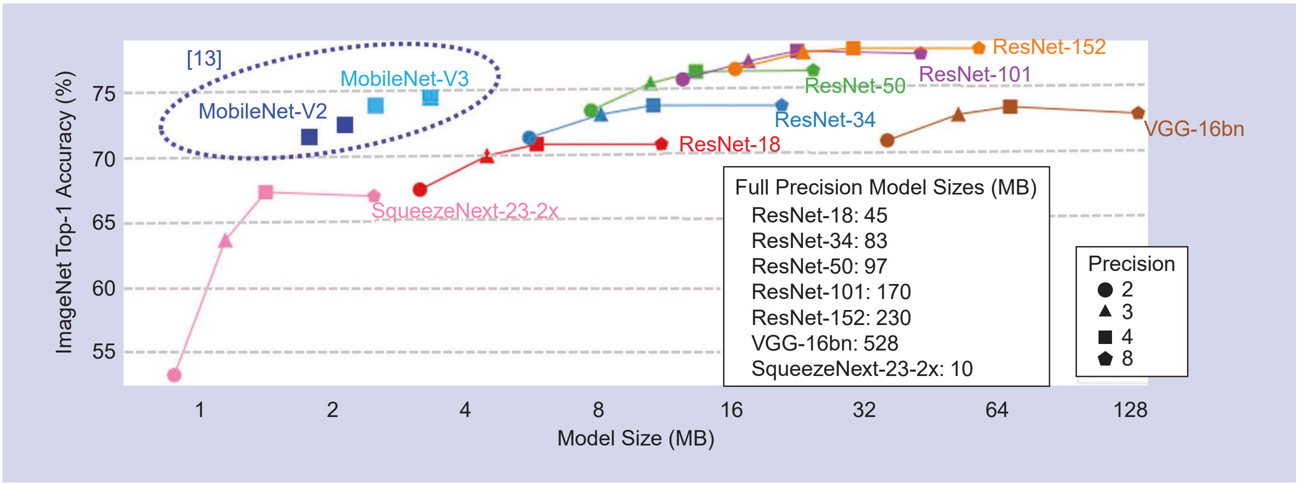


FIGURE 1: Low-precision DNN algorithms and ImageNet accuracy [12], [13].

The OSD minimizes the read/write energy of the partial sum, which requires higher precision than the input or weights.

Digital AI Accelerators Featuring Low Precision

Different DNN models have different optimal weight precisions, and different layers in a given DNN model have different optimal precisions. Therefore, to obtain the optimal energy–accuracy tradeoff, supporting variable weight precision is important for DNN accelerators.

In the UNPU accelerator [22], variable weight precision from 1 to 16 b is supported by bit serial processing, where MAC operations with *N*-b weight precision are computed sequentially from the LSB to the MSB of the weights for *N* cycles by shifting and accumulating the partial sums.

Higher energy efficiency is achieved for lower-precision weights, while the accuracy degradation could occur depending on the DNN. In addition, by efficiently reusing input feature maps, the same UNPU chip can be fully shared for convolution, recurrent, and fully connected layers.

In [23], Intel presented a digital binary neural network (BNN) accelerator in 10-nm CMOS. When *N*-b precision is reduced to 1-b precision, the memory storage is reduced linearly, while the compute complexity is reduced quadratically (e.g., 8-b MAC energy is 10–100× higher than 1-b MAC energy). To that end, the proposed BNN accelerator employs much higher parallelism of very-low-precision MACs and data reuse to amortize the cost of memory access and data movement across many operations. The accelerator chip has 131,000 binary MAC units or XNOR gates split between a total of 128 memory execution units. At 0.37 V, 617 TOPS/W energy efficiency is achieved.

IBM presented a 7-nm AI chip [24] that supports both fixed-point precision inference and floating-point precision training. This chip consists of four AI cores, where each core has two corelets with a private L0 and a shared L1 scratchpad.

Each corelet contains an 8×8 array of mixed-precision engines (MPEs), where MPEs implement separate compute pipelines for various precisions. Inference workloads can be executed with 2- or 4-b fixed-point precision and training workloads can be operated with hybrid 8- or 16-b floating-point precision to meet diverse application demands for both AI inference and training. For inference with 4-b fixed-point precision, with a 0.55-V core and 0.7-V SRAM supply, 16.5 TOPS/W are reported. A scaled-up chip with 32 cores achieves >60% utilization for ResNet-50 and >70% utilization for the Google Neural Machine Translation model.

Digital AI Accelerators Featuring Pruning/Sparsity

The STICKER-T accelerator [25] employed block-circulant weights as a structured compression technique, where each row vector circulantly rotates one element to the right side to generate the next row vector. Therefore, the first row includes all information in this matrix block, leading to $N\times$ storage reduction. By training weights in the block-circulant matrix format, the matrix–vector multiplication can be performed with frequency-domain elementwise production, and FFT operations can reduce the computation complexity from $O(n^2)$ to $O(n\log n)$.

With the block-circulant technique, STICKER-T has a frequency-domain 16×16 MAC array with bit serial processing to flexibly support 1–12-b precision. Sixteen activations are shared by the same PE row, and 16 weights are shared by the same PE

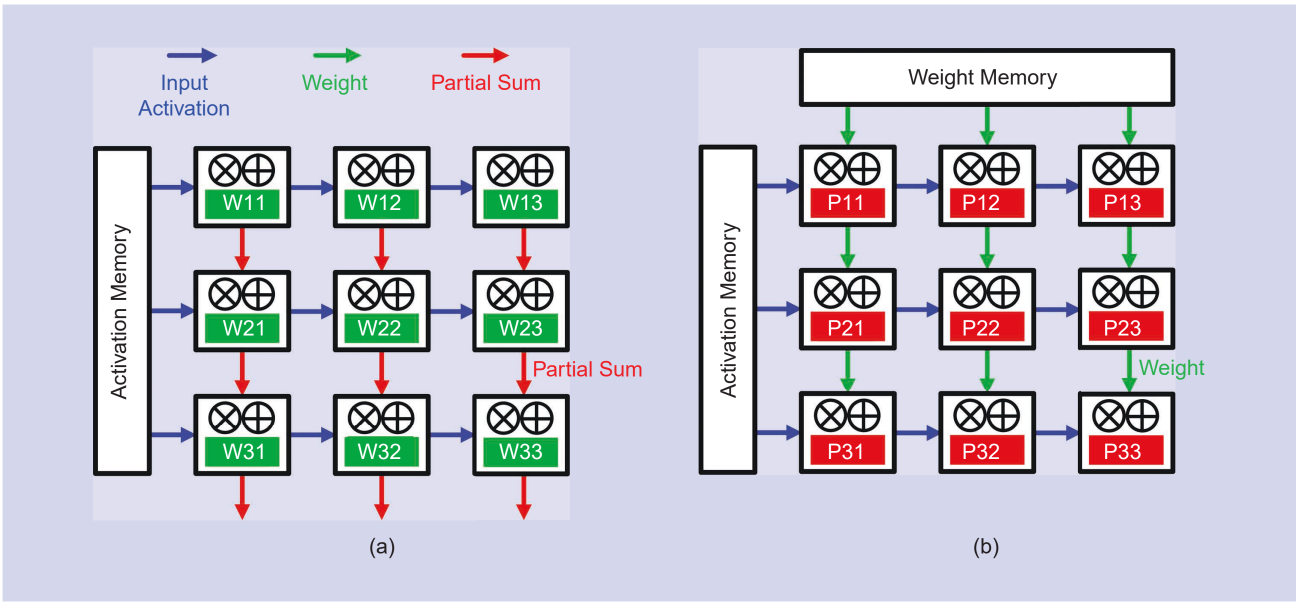


FIGURE 2: (a) The weight-stationary dataflow [19], [20] and (b) output-stationary dataflow [21]. P: partial sum; W: weight.

column. An energy efficiency of up to 140 TOPS/W is reported for 1-b precision, while this decreases progressively for higher-bit precision. At the same bit precision, a larger block size leads to higher energy efficiency due to the reduced FFT and MAC operations, while this is achieved at moderately lower DNN accuracy. Different types of neural networks, such as tiny YOLO CNN, RNN, and so on for different data sets have been demonstrated, using different block sizes and precision.

Long short-term memory (LSTM) is a type of RNN that is widely used for speech applications, but LSTMs pose difficulties for efficient hardware due to the large number of weights and amount of computation complexity. In [17], a new hierarchical coarse-grain sparsity (HCGS) scheme is presented that structurally compresses the weights of LSTM models. HCGS removes block weights for weight matrices in LSTMs in a hierarchical blockwise manner. Within the first-level coarse block sparsity, a second-level fine block sparsity is applied recursively. With 16× hierarchical blockwise sparsity, all weights for a three-layer LSTM fully fit on chip with 288 kB of memory, and only 8.5 kB of memory are employed for index and bias. Since HCGS ensures a regular blockwise sparse weight structure and access, the MAC engines exhibit a high utilization ratio of >98% throughout the LSTM operation. The HCGS accelerator achieves an average energy efficiency of 8.9/7.2 TOPS/W for LSTMs for TIMIT/TED-LIUM data sets while performing real-time speech recognition.

Based on recent works that jointly optimize pruning and low-precision quantization, the relative cost of index storage for elementwise pruning will be even higher for DNNs with low-precision weights. To that end, structured compression works become more important with low-precision quantization since it can share the index per block, substantially reduce the index overhead, and also make the weight memory access regular.

Samsung presented an activation sparsity-aware neural processing unit (NPU) for mobile SoCs in 5-nm CMOS [20]. This NPU chip consists of three cores, where each core has 2,048 8-b MACs that employ a WSD to maximize the reuse of weights. The convolution engine needs to maintain a high utilization factor for diverse convolutions with different parameters, such as dilation, stride, and kernel sizes. Since most DNN layers have many channels, 16 channels are computed in parallel so that the MAC utilization remains high across diverse convolutions. The NPU performs zero skipping for activation sparsity by selecting only a set of nonzero values to form a dense tensor, and the weight matrix is adjusted accordingly to improve the compute efficiency. By optimizing zero skipping, reconfiguration, and multithreading, the overall inference throughput for the Inception-V3 model is improved to 623 inferences/s at 1.2-GHz frequency. Including DMA power, an energy efficiency of 13.6 TOPS/W was measured at 0.6 V, where MAC utilization reached 84%.

The recent trends of digital AI accelerators can be summarized as follows:

- First, in many digital accelerators, the MAC array exhibits a high degree of reconfiguration capability, e.g., suitably reconfiguring the dataflow, stationary scheme, or MAC computation has been reported to achieve high utilization and energy efficiency.
- Second, high to low precision is flexibly supported for both activations and weights for AI inference. The surveyed accelerators supported variable precision from 16 b down to 1 b in the chip design.
- Third, sparsity-aware hardware design has been incorporated in many accelerator designs. The static weight sparsity can be formed in both elementwise and structured sparsity manners, which have tradeoffs in the achievable level of sparsity, index storage, and regularity of memory access. On the other hand, to efficiently handle the dynamic activation sparsity, zero skipping and forming a dense activation tensor with only nonzero activations have been investigated.

Analog/Mixed-Signal AI Accelerators

From the digital AI accelerators in the literature that reported power/energy breakdown [27], [28] (Figure 3), it can be seen that data access energy to/from on-chip SRAMs constitute two thirds or more of the total system power/energy. This is because large amounts of memory access and data communication (from the memory to compute engine) are required to perform the computation. To address such bottlenecks, colocating memory and compute and performing computation in the analog domain have been proposed as the remedies.

Switched-Capacitor Circuit-Based Accelerators

In an early work [29], an analog MAC engine design based on switched-capacitor circuits was presented. The 40-nm chip exploited 300-aF unit fringe capacitors for efficient charge-domain processing with local memory and achieved 8.7 TOPS/W at 1 GHz.

A mixed-signal binary CNN processor was presented in [30]. Digital XNOR gates were employed for the multiplication operation of BNNs, and the wide accumulation of XNOR results were implemented with low-energy switched-capacitor circuits. By adapting the BNN algorithm, such as fixing the number of channels to be 256 and the convolution kernel size to be 2×2, the customized accelerator stores all weights on chip and exploits data locality and reuse, demonstrating a low 3.8-μJ energy per inference for 86% CIFAR-10 accuracy.

IMC Scheme

To make computation more immersed with the weight storage and to alleviate the memory bottlenecks, an IMC scheme was proposed in recent years, where we aim to embed computing inside the memory. In conventional digital AI accelerators, we read out weights from the memory row by row and convey them to a separate compute or MAC engine. In IMC, we turn on multiple or all rows of the subarray simultaneously and perform the MAC computation inside the memory, typically along the bitline in an analog manner, which can largely reduce the data transfer since there is no separate compute engine. By asserting all rows together, compute parallelism is also largely increased. These advantages can lead to high energy efficiency.

However, IMC exhibits some challenges, including analog variability, potential DNN accuracy loss, large-scale integration, and making it programmable to flexibly map various AI workloads. While there have been different memory technologies presented for IMC, in this work, we mostly focus on the SRAM-based IMC designs due to their robustness and viability for large-scale integration in any CMOS node.

Figure 4 categorizes analog SRAM IMC schemes based on the bitcell design and location/method of analog computation, focusing on the SRAM column slice. Some early works employed the conventional six-transistor (6T) SRAM and pursued activating multiple/all rows of a subarray [31]–[34] [Figure 4(a)], thereby performing analog computation along the bitline. The 6T SRAM is, evidently, the densest SRAM bitcell. However, analog computation results can drive the bitline to a very low or high voltage, and, with all wordlines turned on, this could result in a read disturb by flipping the storage value of a bitcell. To prevent a read disturb, the wordline voltages in some of these works were lowered to ~0.4 V [32], [34].

Some other works dedicated analog compute engines that can be shared by a group of rows [35]–[37] [Figure 4(b)], for example, 16 rows in a 256-row subarray [37]. Within a group, the SRAM access and computation occur in a row-by-row manner, but different groups in the subarray are computed in parallel. The local analog compute circuits add some area overhead, but the smallest 6T cells can be used for the bitcell, while some works used larger bitcells. Overall, the area overhead is relatively small, but the parallelism is reduced due to the row-by-row operation within a group.

To be more aggressive on the parallelism, a number of works pursued activating all rows, but, to eliminate the read disturb issue, a new bitcell was employed with at least a couple of additional transistors per bitcell on top of the 6T [38]–[43] [Figure 4(c)]. The new bitcell could be foundry 8T [38], a custom 8T1C design [40], [43], and so on. Considering this, an area overhead exists compared to 6T SRAMs. However, by turning on all of the rows, the elementwise multiplication computed in each bitcell can be accumulated at once by connecting the read bitline (RBL) together, and this results in very high parallelism and low latency. We will go through representative design examples for this highly parallel SRAM IMC scheme.

Resistive and Capacitive SRAM IMC

The analog SRAM IMC can be largely categorized into two types, namely, resistive IMC and capacitive IMC.

Resistive SRAM IMC

Let's look into the resistive IMC works by examining two representative papers [39], [42]. In the IMC bitcell itself, bitwise multiplication between the input activation and weight is performed [Figure 5(a)]. In [39], the input and weight are represented by "1" or "0," and the multiplication between these two can be done by pulling down the RBL only when both the input and weight are "1" and leaving it as is when either of them is "0." In the XNOR SRAM work [42], several additional transistors are employed in the bitcell to implement the

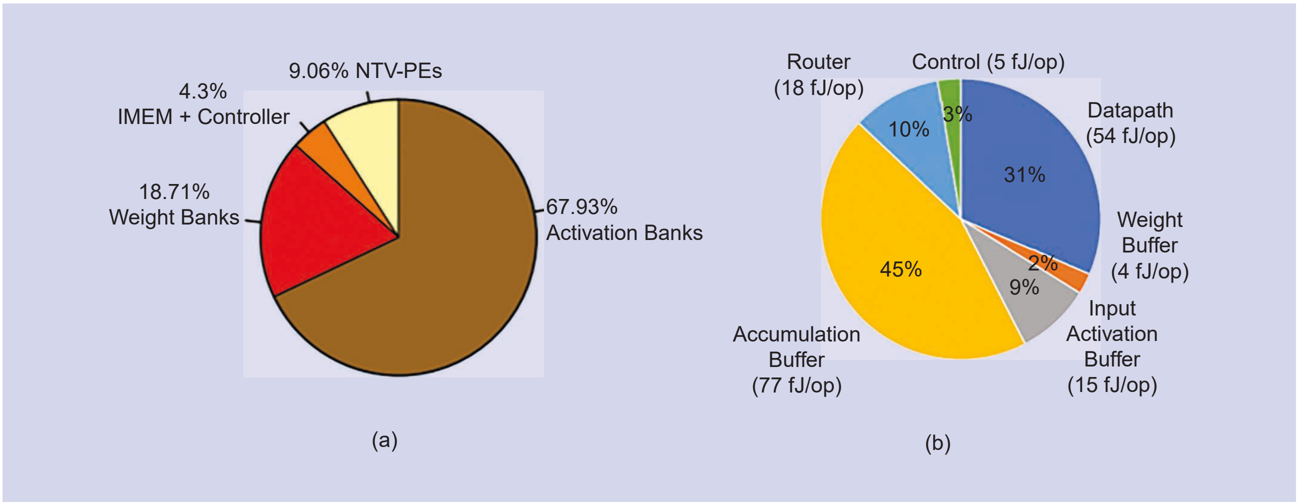


FIGURE 3: The power/energy breakdown of digital AI accelerators: the (a) EOS chip power for AlexNet [27], and (b) MCM chip PE energy [28]. EOS: enhanced output stationary; IMEM: instruction memory; NTV: near-threshold voltage; op: operation.

XNOR-based multiplication for BNNs. If the input and weight have the same polarity, the RBL will be pulled up, and, if the input and weight have different polarities, the RBL will be pulled down.

Figure 5(b) shows the analog accumulation along the column in resistive IMC designs. By activating *N* rows simultaneously, *N* cells in the SRAM column connect to the same RBL. In [39], the precharged RBL will discharge with different strengths depending on how many bitcells are pulling down. The analog RBL voltage will be evaluated after a specific amount of time, which can make the evaluation time sensitive. In [42], depending on how many bitcells pull up versus pull down the RBL, a robust resistive divider is formed without time dependency, but, depending on the RBL voltage level, a crowbar current could flow.

Figure 5(c) shows the MAC computation of resistive IMC designs. By turning on *N* rows together, an *N*-input binary MAC could be performed in a single cycle. The transfer curves of the two resistive IMC designs exhibit a monotonic relationship between the partial MAC value and the RBL voltage.

To make computation more immersed with the weight storage and to alleviate the memory bottlenecks, an IMC scheme was proposed in recent years, where we aim to embed computing inside the memory.

Capacitive SRAM IMC

For analog capacitive IMC designs, we describe the design of two representative works [40], [43]. In [40], the modified SRAM bitcell includes two additional PMOS transistors and a capacitor, where the XNOR multiplication result between the activation and a weight of –1 or +1 will discharge the capacitor to the ground or charge the capacitor to *Vdd*. In C3SRAM [43], the bitcell consists of two additional NMOS transistors and a serially connected capacitor, where an XNOR multiplication result of –1 or +1 will drive the middle node *Vc* with 0 V or *Vdd*.

Figure 6(b) shows the analog accumulation along the column in capacitive IMC designs. In [40], as *N* bitcells are activated simultaneously, the individual cell's capacitor discharged at 0 V or charged at *Vdd* gets connected together and goes through a charge-sharing operation. In [43], *N* bitcells connect to the same MAC bitline (*MBL*) through the series capacitor, and capacitive coupling is performed based on the bitcell multiplication result. As a result, a capacitive divider is formed, where some bitcells' capacitors will sit between the *MBL* and *Vdd*, and others will be between the *MBL* and ground.

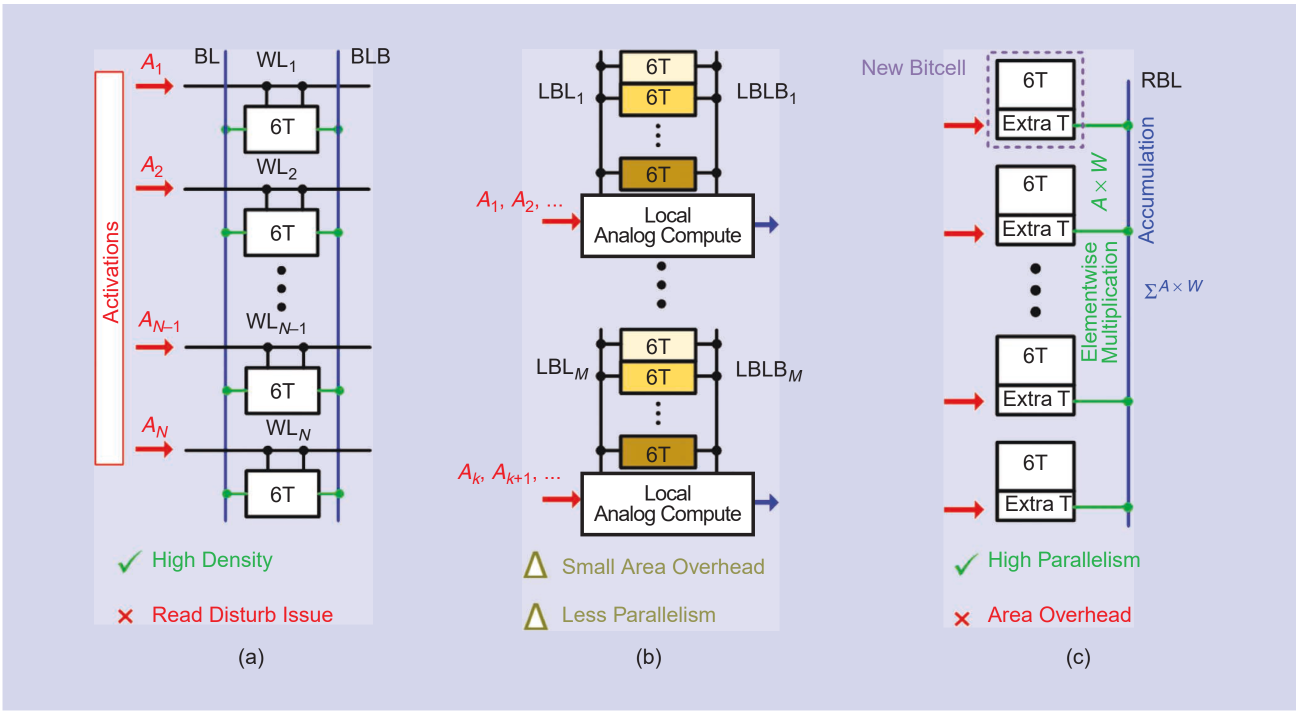


FIGURE 4: The categorization of SRAM IMC schemes: (a) 6T bitcell plus parallel compute [31]–[34], (b) 6T bitcell plus local compute [35], [36] and 10T plus local compute [37], and (c) (6 + extra)T bitcell parallel compute [38]–[43]. BL and BLB represent differential bitlines. LBL and LBLB represent local differential bitlines. 6T: six-transistor; 10T: 10-transistor; T: transistor.

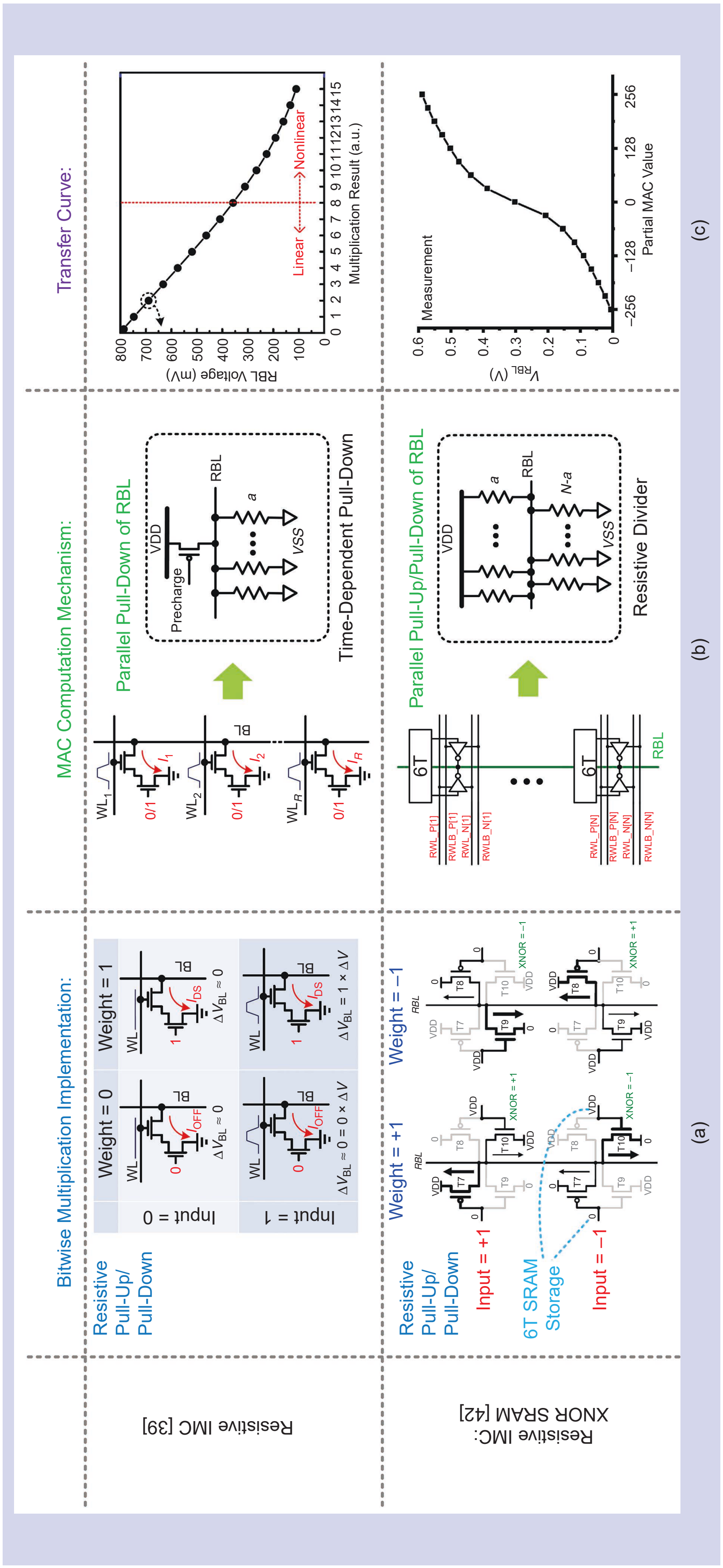


FIGURE 5: The resistive IMC circuits and operation: (a) the IMC SRAM bitcell design, (b) N bitcells connected to the same RBL, and (c) the N-input MAC computation. a.u.: arbitrary units.

Figure 6(c) shows the transfer curves of the MAC computation of the capacitive IMC designs when N rows are activated together. Due to the nature of the charge sharing or capacitive divider, capacitive IMC designs exhibit a more linear relationship between the partial MAC value and the RBL voltage compared to their resistive IMC counterparts.

IMC System Designs

Figure 7 shows the trend from IMC macro designs toward IMC system designs in the literature. SRAM IMC designs started from single-macro IMC designs, including CONV-SRAM [37], XNOR SRAM [42], Twin-8T SRAM [38], C3SRAM [43], and the 7-nm IMC macro by TSMC [39]. Evidently, there is a large gap between single IMC macros and an end-to-end accelerator, so, more recently, researchers have presented IMC system designs where a small to large number of IMC macros have been integrated [16], [41], [44]–[47], up to several megabits of IMC SRAM. Here, we describe two of the largest SRAM IMC system integrations to date [46], [47].

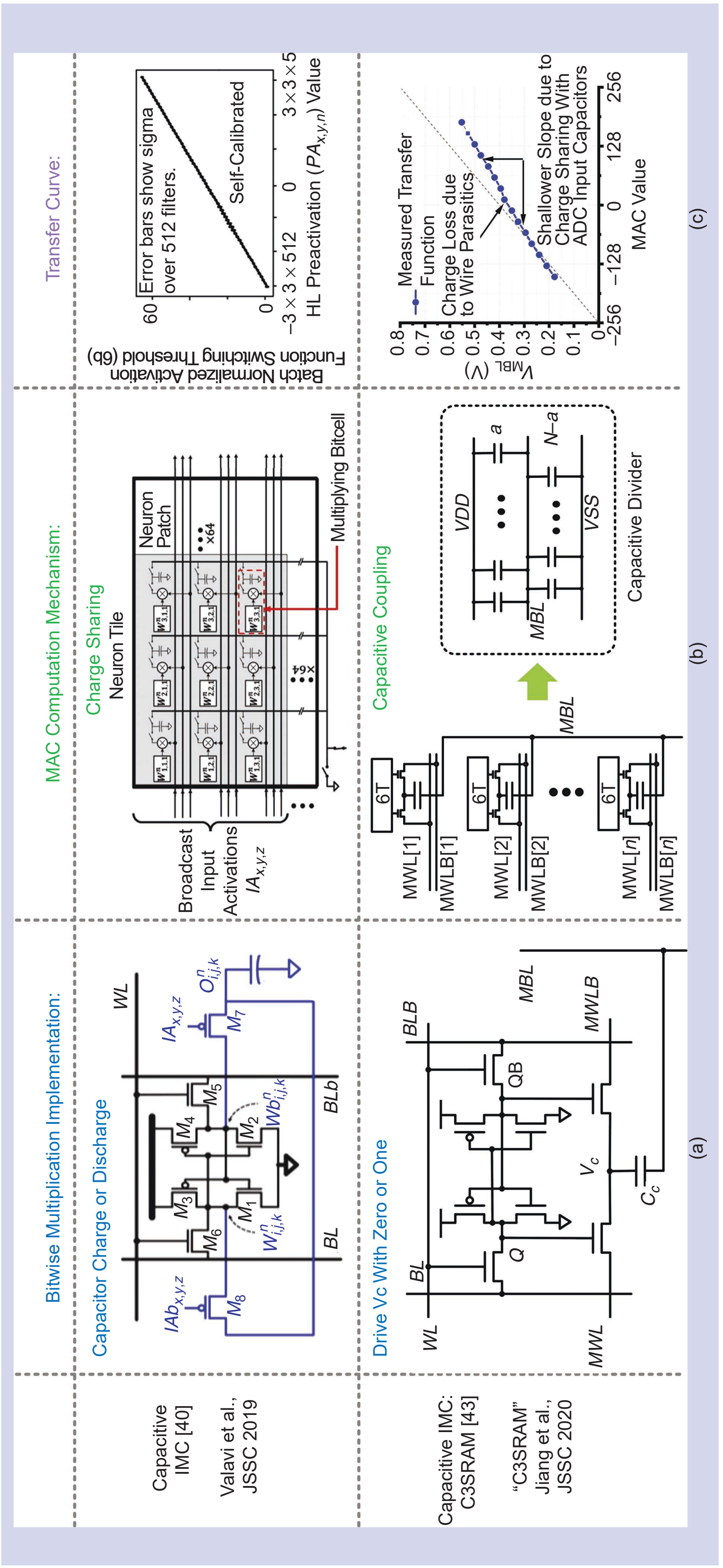
In [46], a 16-nm scalable IMC accelerator design was presented, occupying a 25-mm^2 area. This IMC accelerator includes a 4×4 array of compute-in-memory unit (CIMU) cores, an on-chip network between cores, buffers, control circuits, and off-chip interfaces. The CIMU consists of 1) the compute-in-memory array (CIMA) for IMC operations, 2) a near-memory computing digital SIMD with a custom instruction set for flexible elementwise operations, and 3) buffering and control for enabling a range of dataflows. The IMC engine is a capacitive CIMA with 1,152 rows and 256 columns, which executes fully row/column-parallel analog computation, which is digitized by an 8-b ADC. The capacitive IMC allows linearity between the partial MAC value and ADC output, and it achieved high CNN accuracy for both the CIFAR-10 and ImageNet data sets. For 4-b activation and weight precision, an energy efficiency of

121 TOPS/W and a MAC compute density of 2.67 TOPS/mm^2 are reported.

In [47], a 28-nm programmable IMC accelerator chip with 20 mm^2 has been presented. Using a capacitive coupling-based SRAM IMC bitcell, a 256 × 128 SRAM IMC macro is designed with a column-parallel 4-b ADC, and 108 such macros are integrated for the total chip. A total of 3.4 Mb of memory was dedicated for SRAM IMC, and 1.5 Mb of off-the-shelf activation memory was also employed. A custom instruction set architecture (ISA) was developed for programmability support with IMC and SIMD operations, where the hardware loop control feature was able to reduce the total number of instructions by 4×. A peak system-level energy efficiency of 437 TOPS/W and peak throughput of 4.9 TOPS for binary precision were demonstrated. These energy numbers include all components of the overall accelerator chip, such as the activation memory, 256-way SIMD unit for vector operations, and on-chip communication.

The recent trends of analog/mixed-signal AI accelerators can be summarized as follows:

- Many single-macro IMC designs have been presented in recent years, and, currently, the best IMC designs report >1,000 TOPS/W at the single-macro level [49], [50]. Both resistive IMC and capacitive IMC have been demonstrated, where capacitive IMC shows higher linearity between MAC results and analog voltage values.
- Analog IMC designs also support multibit and flexible precision from 1- to 8-b precision for both activations and weights of DNNs [33], [35], [38], [39], [50], [51]. Some sparsity-aware IMC designs have also been presented [16], [52] to further optimize energy with techniques such as zero skipping.
- More recently, larger-scale IMC accelerators that integrate more than 100 IMC macros have been recently reported in scaled CMOS technologies, where several megabits



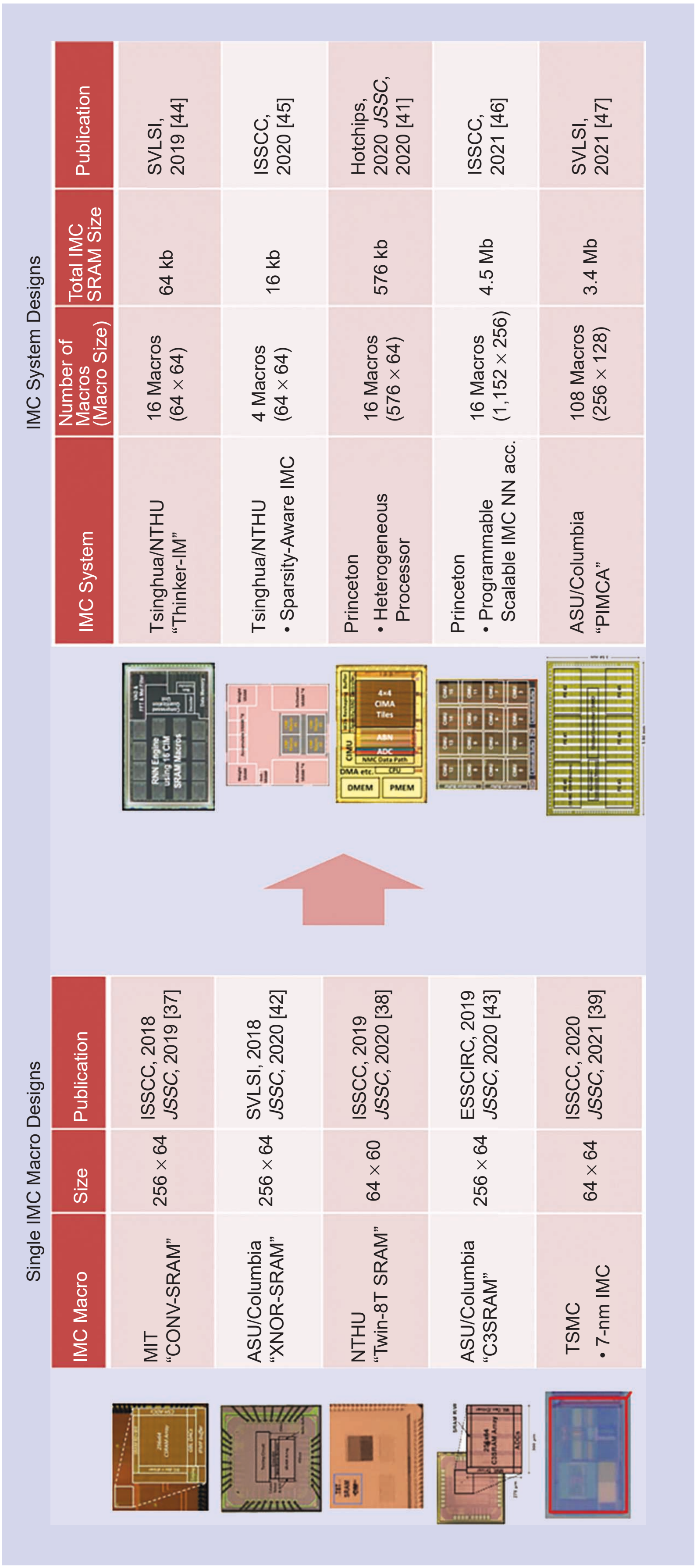
Capacitive IMC [40]
Valavi et al., JSSC 2019
Capacitive IMC: C3SRAM [43]
"C3SRAM" Jiang et al., JSSC 2020
Bitwise Multiplication Implementation:
Capacitor Charge or Discharge
Drive Vc With Zero or One
MAC Computation Mechanism:
Charge Sharing
Neuron Tile
Neuron Patch
Broadcast Input Activations $IA_{x,y,z}$
Multiplying Bitcell
Capacitive Coupling
Capacitive Divider
Transfer Curve:
Error bars show sigma over 512 filters.
Self-Calibrated
Batch Normalized Activation Function Switching Threshold (6b)
HL Preactivation ($PA_{x,y,n}$) Value
Measured Transfer Function
Charge Loss due to Wire Parasitics
Shallower Slope due to Charge Sharing With ADC Input Capacitors
V_{MBL} (V)
MAC Value
(a)
(b)
(c)

FIGURE 6: The capacitive IMC circuits and operation. (a) the IMC SRAM bitcell design, (b) N bitcells connected to the same RBL, and (c) the N-input MAC computation. MWL and MWLB represent differential MAC wordlines. HL: hidden layer.

of IMC SRAM are integrated on chip [46], [47].

- By design, IMC-based accelerators use a weight-stationary scheme, and recent IMC accelerators made efforts to support the programmability with a custom ISA so that various AI models can be flexibly mapped onto the same chip.

In [48], the energy efficiency, throughput, and compute density of digital accelerators and IMC designs based on SRAM and embedded nonvolatile memory (eNVM) have been benchmarked comprehensively, as shown in Figure 8. SRAM-based IMCs show much higher energy efficiency over digital accelerators at the macro level, but the energy efficiency gap noticeably reduces at the processor/system level. Digital accelerators can achieve arbitrarily high throughput via scale-up without compromising on their accuracy. However, more design efforts are required for IMC-based accelerators to preserve accuracy during scale-up.


Single IMC Macro Designs
IMC System Designs

Single IMC Macro Designs

IMC Macro	Size	Publication
MIT “CONV-SRAM”	256 × 64	ISSCC, 2018 *JSSC*, 2019 [37]
ASU/Columbia “XNOR-SRAM”	256 × 64	SVLSI, 2018 *JSSC*, 2020 [42]
NTHU “Twin-8T SRAM”	64 × 60	ISSCC, 2019 *JSSC*, 2020 [38]
ASU/Columbia “C3SRAM”	256 × 64	ESSCIRC, 2019 *JSSC*, 2020 [43]
TSMC • 7-nm IMC	64 × 64	ISSCC, 2020 *JSSC*, 2021 [39]

IMC System Designs

IMC System	Number of Macros (Macro Size)	Total IMC SRAM Size	Publication
Tsinghua/NTHU “Thinker-IM”	16 Macros (64 × 64)	64 kb	SVLSI, 2019 [44]
Tsinghua/NTHU • Sparsity-Aware IMC	4 Macros (64 × 64)	16 kb	ISSCC, 2020 [45]
Princeton • Heterogeneous Processor	16 Macros (576 × 64)	576 kb	Hotchips, 2020 *JSSC*, 2020 [41]
Princeton • Programmable Scalable IMC NN acc.	16 Macros (1,152 × 256)	4.5 Mb	ISSCC, 2021 [46]
ASU/Columbia “PIMCA”	108 Macros (256 × 128)	3.4 Mb	SVLSI, 2021 [47]

FIGURE 7: The trend of IMC macro designs toward IMC system designs. ASU: Arizona State University; ISSCC: IEEE Solid-State Circuits Conference; JSSC: *IEEE Journal of Solid-State Circuits*; NN acc.: neural network accelerator; NTHU: National Tsing Hua University; PIMCA: programmable in-memory computing accelerator; SVLSI: IEEE Symposium on VLSI Circuits.

Addressing Challenges and New AI Accelerator Design Directions

There are several notable challenges of analog SRAM IMC. First, IMC trades off the signal-to-noise ratio (SNR) [53], which makes it more susceptible to process, voltage, and temperature (PVT) variation as well as the ADC offset. These could result in a nonexact computation result and network-level DNN accuracy degradation, especially for complex and larger data sets. Second, the IMC peripheral circuitry requires the ADC to digitize the analog computation. Despite many advances in ADC designs, ADCs still consume a large area and high energy, and the ADC offset due to transistor mismatch can cause errors. Third, since a couple of transistors or passive components, such as capacitors, could be added to each bitcell, the SRAM bitcell can be relatively large compared to the off-the-shelf 6T counterparts, and this brings density and leakage concerns. Fourth, the rigid crossbar structure used in SRAM

IMC is not particularly amenable for fine-grain weight pruning. Here, we introduce specific approaches that have been proposed to tackle each of these challenges.

We need ADCs and DACs because we perform analog computation inside the IMC macro while utilizing digital activation storage and digital communication between IMC macros.

Addressing a Low SNR

Associated with a lower SNR, analog IMC involves inherent intrachip and interchip variability as well as ADC quantization. As shown in a number of IMC chips from the literature, variability can be observed in the ADC outputs for the same ideal MAC value, and accuracy degradation is often reported compared to the digital baseline. To mitigate this accuracy loss, some IMC SRAM works attempted to improve the SNR by limiting the number of activated rows for IMC, e.g., 18 rows in [38], but this reduces the computing parallelism and energy efficiency.

To improve the IMC hardware accuracy, recent works, such as [54], performed customized DNN training with noise injection at the individual weight level, where the injected noise is drawn from Gaussian distributions based on memory bitcell variations. However, such individual weight-level Gaussian noise injections could be suboptimal for actual IMC chips for two reasons: 1) individual weight-level noise does not consider the IMC crossbar structure and other hardware noise, such as bitline or ADC noise, and 2) Gaussian noise does not necessarily represent the actual IMC hardware noise well [55].

Considering these, IMC noise-aware DNN training is performed [55] by injecting the measured IMC hardware noise into the forward pass during DNN training. All MAC operations in DNNs are divided into multiple *N*-input MAC operations, where *N* is the number of rows activated together in the IMC macro. For each *N*-input MAC operation, the noisy and quantized partial sum results from IMC chip measurements are used, where the ADC outputs for given MAC values are randomly sampled based on the conditional probability of ADC output measurements for each MAC value. In [55], injecting actual IMC hardware noise measured from two different IMC chips of XNOR SRAM [42] and C3SRAM [43] has been evaluated. The reported results show that the IMC noise-aware training scheme shows higher inference accuracy for the IMC hardware across multiple DNN models, different precision, and a different amount of noise, and the improvement is especially higher for higher-noise environments (e.g., a low *Vdd* for C3SRAM [43]).

Addressing Area-/ Energy-Hungry ADCs

Why are relatively high-resolution ADCs necessary for IMC? Each IMC macro has a finite number of rows (e.g., 256), which computes the partial sums of a DNN layer. When dividing a large dot product into a number of smaller ones, each partial sum should not be prematurely quantized and should have relatively high precision. With regard to the ADC challenge, different methods have been proposed for possibly removing the ADCs (and DACs) for IMC designs. We need ADCs and DACs because we perform analog computation inside the IMC macro while utilizing digital activation storage and digital communication between IMC macros. Then, we can think again about 1) an all-digital design but with IMC and 2) an all-analog AI accelerator where all operations are performed in the analog domain.

Digital IMC

All-digital IMC designs were recently presented in [56] and [58] [Figure 9(a)]. In analog IMC, the bitwise multiplication results are accumulated in an analog manner along the bitline, which then necessitates an ADC at the column end. On the other hand, the digital IMC design proposes performing the accumulation with

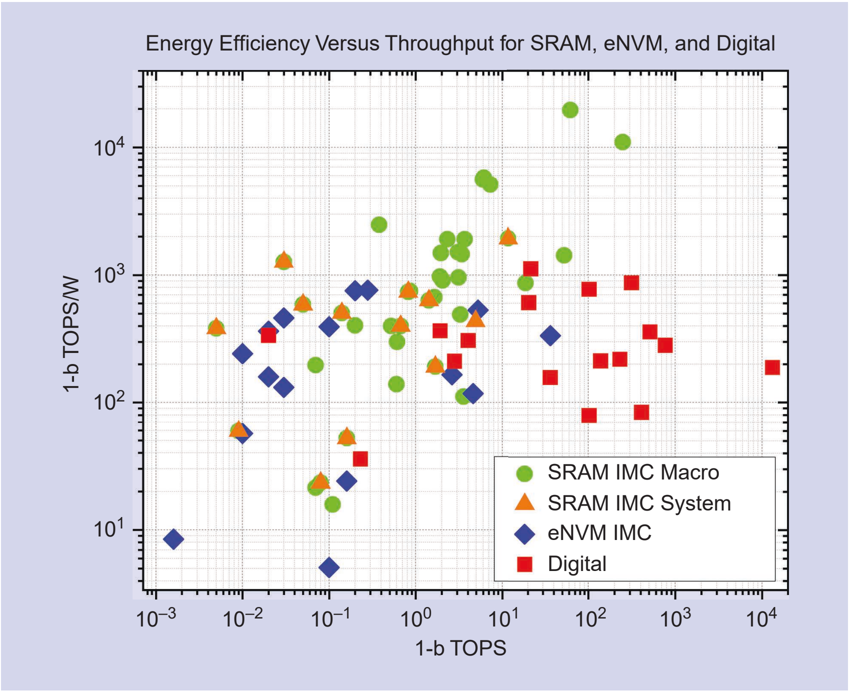


FIGURE 8: The digital and analog accelerator literature. (Adapted from [48].)

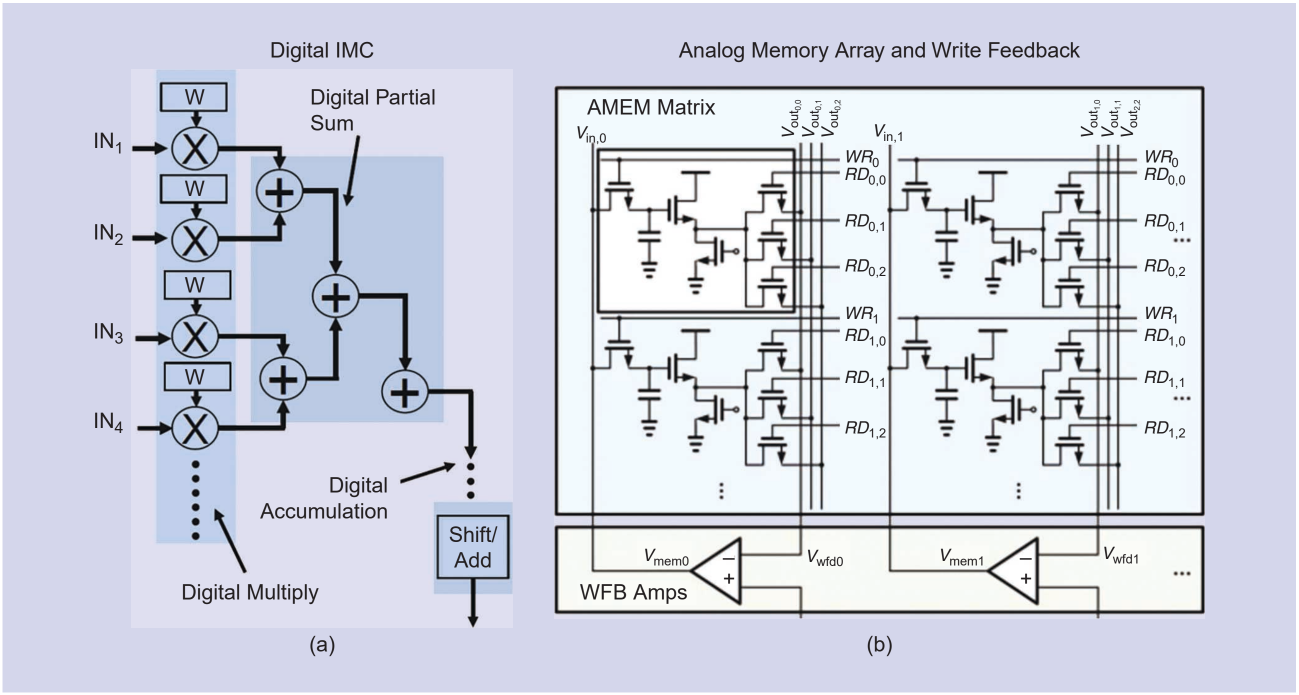


FIGURE 9: Addressing ADC challenges by (a) digital IMC [56] and (b) analog memory-based all-analog design [57]. WFB: write-with-feedback.

digital logic gates while adding such digital accumulation circuitry inside the IMC SRAM macro for every group of columns. Such digital IMC designs eliminate ADCs, since all computations are in the digital domain, and are also advantageous for CMOS scaling and lower *Vdd* operation. However, the digital implementation of the wide adder tree can consume a large area; thus, a compact implementation becomes important.

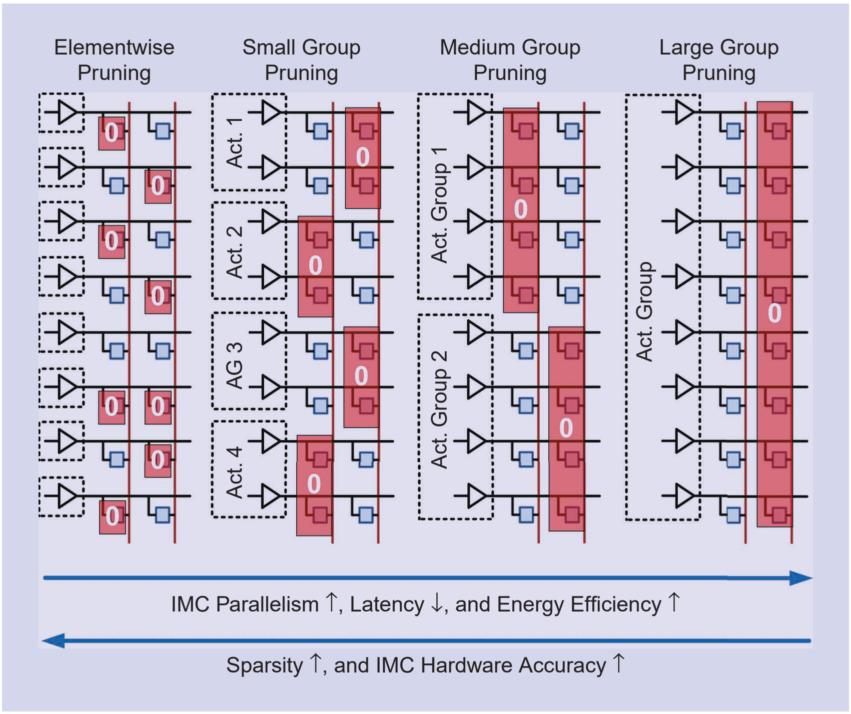


FIGURE 10: The tradeoffs of pruning for IMC designs [65].

All-Analog AI Accelerator

On the other hand, an all-analog DNN accelerator "ARCHON" in 28 nm was presented in [57] [Figure (9b)], which comprises both analog computation and capacitor-based analog memory (AMEM). ARCHON features an analog neuronal computation unit (ANU) based on voltage-to-time converters and weighted current summation as well as an AMEM based on a 6T1C bitcell that can store ~5-b precision across PVT variations. The ANU and AMEM can perform the computations needed for CNNs in the analog domain across layers without data conversions. ARCHON achieves an energy efficiency of 332.7 TOPS/W (the analog datapath) and 19.9 TOPS/W (the processor level).

Addressing Density

To support IMC, the SRAM bitcell and macro areas are both becoming relatively large. To alleviate the density concerns, IMC designs based on eNVM, such as RRAM (with a single-level cell [59] and multilevel cell [60]), PCM [61], MRAM [62], and Flash [63], have been presented. Such eNVMs are denser than SRAM and consume low leakage. Also, with the resistive device, it can naturally support matrix–vector multiplication. On the other hand, the nonidealities of eNVM devices include endurance, variation, drift, and so on. In several recent works, the larger-scale integration of eNVM-based IMC systems has been demonstrated. In [63], an embedded Flash-based IMC system with 79 million 8-b weights was presented, and [64] presented a 2.25-MB RRAM-based IMC system with an embedded ARM processor.

Addressing Pruning for IMC Designs

Applying random sparsity patterns resulted from fine-grain nonstructured pruning to a fixed SRAM IMC array structure can become inefficient [65]. If the IMC operation happens on a column basis, it will be much more efficient to prune out the entire/partial column in a structured manner. However, as shown in Figure 10, smaller group sizes achieve higher sparsity compared to the large-sized groups. On the other hand, a small-sized group will restrict the number of rows that can be activated simultaneously, which requires a higher number of cycles to go through the same crossbar array. Other approaches include sparsity-aware activation/weight processing for IMC macro design [16] and sparsity-optimized IMC bitcell design [66]. Overall, this challenge needs a carefully structured pruning algorithm and supporting IMC hardware co-design.

Conclusion

In this article, we presented how both digital and analog AI accelerators have largely advanced in recent years. The trends of all-digital accelerators include a reconfigurable MAC array, high utilization across various AI models, flexible precision support, and weight/activation sparsity-aware design. Regarding the trends on analog/mixed-signal accelerators, single IMC macro designs are scaled up with a higher level of integration for a many-macro IMC system design. Flexible precision and programmability have been supported in larger-scale IMC systems. The challenges for these designs are being addressed in different ways. For analog IMC designs, improving the DNN accuracy, ADC overhead, density, and sparsity are important. By addressing such challenges, new all-analog, digital IMC- and NVM-based AI accelerators are being further presented in the literature.

Acknowledgments

This work was, in part, supported by National Science Foundation grant 1652866 and the Center for Brain-Inspired Computing, one of six centers in the Joint University Microelectronics Program, a Semiconductor Research Corporation program sponsored by DARPA.

References

[1] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, “Model compression and hardware acceleration for neural networks: A comprehensive survey,” *Proc. IEEE*, vol. 108, no. 4, pp. 485–532, 2020, doi: 10.1109/JPROC.2020.2976475.

[2] G. W. Burr, S. Lim, B. Murmann, R. Venkatesan, and M. Verhelst, “Fair and comprehensive benchmarking of machine learning processing chips,” *IEEE Des. Test*, vol. 39, no. 3, pp. 18–27, 2022, doi: 10.1109/MDAT.2021.3063366.

[3] J. Choi, S. Venkataramani, V. V. Srinivasan, K. Gopalakrishnan, Z. Wang, and P. Chuang, “Accurate and efficient 2-bit quantized neural networks,” in *Proc. Conf. Mach. Learn. Syst. (MLSys)*, 2019, pp. 348–359.

[4] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2016, pp. 1–14.

[5] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, “Efficient architecture search by network transformation,” in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 2787–2794, doi: 10.5555/3504035.3504375.

[6] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, “Tensorizing Neural Networks,” in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2015, vol. 28, pp. 442–450, doi: 10.5555/2969239.2969289.

[7] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, “DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks,” in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2017, pp. 240–241, doi: 10.1109/ISSCC.2017.7870350.

[8] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, “HAQ: Hardware-aware automated quantization with mixed precision,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 8612–8620.

[9] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Quantized neural networks: Training neural networks with low precision weights and activations,” *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, 2018.

[10] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “XNOR-Net: ImageNet classification using binary convolutional neural networks,” in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2016, pp. 525–542, doi: 10.1007/978-3-319-46493-0_32.

[11] A. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marra, “WRPN: Wide reduced-precision networks,” in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2018, pp. 1–11.

[12] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, “Learned step size quantization,” in *Proc. Int. Conf. Learning Representations (ICLR)*, 2020, pp. 1–12.

[13] E. Park and S. Yoo, “PROFIT: A novel training method for sub-4-bit MobileNet models,” in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2020, pp. 430–446, doi: 10.1007/978-3-030-58539-6_26.

[14] S. Park, J. Lee, S. Mo, and J. Shin, “Lookahead: A far-sighted alternative of magnitude-based pruning,” in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2020, pp. 1–20.

[15] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, “Learning structured sparsity in deep neural networks,” in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2016, pp. 2074–2082, doi: 10.5555/3157096.3157329.

[16] J. Yue *et al.*, “A 65nm computing-in-memory-based CNN processor with 2.9-to-35.8TOPS/W system energy efficiency using dynamic-sparsity performance-scaling architecture and energy-efficient inter/intra-macro data reuse,” in *Proc. IEEE Int. Solid- State Circuits Conf. (ISSCC)*, 2020, pp. 234–236, doi: 10.1109/ISSCC19947.2020.9062958.

[17] D. Kadetotad, S. Yin, V. Berisha, C. Chakrabarti, and J. Seo, “An 8.93 TOPS/W LSTM recurrent neural network accelerator featuring hierarchical coarse-grain sparsity for on-device speech recognition,” *IEEE J. Solid-State Circuits*, vol. 55, no. 7, pp. 1877–1887, 2020, doi: 10.1109/JSSC.2020.2992900.

[18] L. Yang, Z. He, and D. Fan, “Harmonious coexistence of structured weight pruning and ternarization for deep neural networks,” in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 4, pp. 6623–6630, 2020, doi: 10.1609/aaai.v34i04.6138.

[19] N. P. Jouppi *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Architecture (ISCA)*, 2017, pp. 1–12, doi: 10.1145/3079856.3080246.

[20] J.-S. Park *et al.*, “A 6K-MAC feature-map-sparsity-aware neural processing unit in 5nm flagship mobile SoC,” in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2021, vol. 64, pp. 152–154, doi: 10.1109/ISSCC42613.2021.9365928.

[21] 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 28nm FDSOI,” in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2017, pp. 246–247, doi: 10.1109/ISSCC.2017.7870353.

[22] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, “UNPU: An energy-efficient deep neural network accelerator with fully variable weight bit precision,” *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, 2019, doi: 10.1109/JSSC.2018.2865489.

[23] P. C. Knag *et al.*, “A 617-TOPS/W all-digital binary neural network accelerator in 10-nm FinFET CMOS,” *IEEE J. Solid-State*

Circuits, vol. 56, no. 4, pp. 1082–1092, 2021, doi: 10.1109/JSSC.2020.3038616.

[24] A. Agrawal *et al.*, "A 7nm 4-core AI chip with 25.6TFLOPS hybrid FP8 training, 102.4TOPS INT4 inference and workload-aware throttling," in *Proc. IEEE Int. Solid- State Circuits Conf. (ISSCC)*, 2021, vol. 64, pp. 144–146, doi: 10.1109/ISSCC42613.2021.9365791.

[25] J. Yue *et al.*, "STICKER-T: An energy-efficient neural network processor using block-circulant algorithm and unified frequency-domain acceleration," *IEEE J. Solid-State Circuits*, vol. 56, no. 6, pp. 1936–1948, 2021, doi: 10.1109/JSSC.2020.3030264.

[26] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017, doi: 10.1109/JSSC.2016.2616357.

[27] J. Sim, S. Lee, and L.-S. Kim, "An energy-efficient deep convolutional neural network inference processor with enhanced output stationary dataflow in 65-nm CMOS," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 1, pp. 87–100, 2020, doi: 10.1109/TVLSI.2019.2935251.

[28] 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, 2020, doi: 10.1109/JSSC.2019.2960488.

[29] E. H. Lee and S. S. Wong, "A 2.5GHz 7.7TOPS/W switched-capacitor matrix multiplier with co-designed local memory in 40nm," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2016, pp. 418–419, doi: 10.1109/ISSCC.2016.7418085.

[30] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An always-On 3.8 μJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 158–172, 2019, doi: 10.1109/JSSC.2018.2869150.

[31] M. Kang, M.-S. Keel, N. R. Shanbhag, S. Eilert, and K. Curewitz, "An energy-efficient VLSI architecture for pattern recognition via deep embedding of computation in SRAM," in *Proc. IEEE Int. Conf. Acoust, Speech Signal Process. (ICASSP)*, 2014, pp. 8326–8330, doi: 10.1109/ICASSP.2014.6855225.

[32] J. Zhang, Z. Wang, and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6T SRAM array," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, 2017, doi: 10.1109/JSSC.2016.2642198.

[33] M. Kang, S. K. Gonugondla, A. Patil, and N. R. Shanbhag, "A multi-functional in-memory inference processor using a standard 6T SRAM array," *IEEE J. Solid-State Circuits*, vol. 53, no. 2, pp. 642–655, 2018, doi: 10.1109/JSSC.2017.2782087.

[34] J. Kim *et al.*, "Area-efficient and variation-tolerant in-memory BNN computing using 6T SRAM array," in *Proc. IEEE Symp. VLSI Circuits*, 2019, pp. C118–C119, doi: 10.23919/VLSIC.2019.8778160.

[35] J.-W. Su *et al.*, "A 28nm 64Kb inference-training two-way transpose multibit 6T SRAM compute-in-memory macro for AI edge chips," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2020, pp. 240–242, doi: 10.1109/ISSCC19947.2020.9062949.

[36] X. Si *et al.*, "A 28nm 64Kb 6T SRAM computing-in-memory macro with 8b MAC operation for AI edge chips," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2020, pp. 246–248, doi: 10.1109/ISSCC19947.2020.9062995.

[37] A. Biswas and A. P. Chandrakasan, "CONV-SRAM: An energy-efficient SRAM with in-memory dot-product computation for low-power convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 217–230, 2019, doi: 10.1109/JSSC.2018.2880918.

[38] X. Si *et al.*, "A twin-8T SRAM computation-in-memory unit-macro for multibit CNN-based AI edge processors," *IEEE J. Solid-State Circuits*, vol. 55, no. 1, pp. 189–202, 2020, doi: 10.1109/JSSC.2019.2952773.

[39] Q. Dong *et al.*, "A 351TOPS/W and 372.4GOPS compute-in-memory SRAM macro in 7nm FinFET CMOS for machine-learning applications," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2020, pp. 242–244, doi: 10.1109/ISSCC19947.2020.9062985.

[40] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A 64-tile 2.4-Mb in-memory-computing CNN accelerator employing charge-domain compute," *IEEE J. Solid-State Circuits*, vol. 54, no. 6, pp. 1789–1799, 2019, doi: 10.1109/JSSC.2019.2899730.

[41] H. Jia, H. Valavi, Y. Tang, J. Zhang, and N. Verma, "A programmable heterogeneous microprocessor based on bit-scalable in-memory computing," *IEEE J. Solid-State Circuits*, vol. 55, no. 9, pp. 2609–2621, 2020, doi: 10.1109/JSSC.2020.2987714.

[42] S. Yin, Z. Jiang, J. Seo, and M. Seok, "XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks," *IEEE J. Solid-State Circuits*, vol. 55, no. 6, pp. 1733–1743, 2020, doi: 10.1109/JSSC.2019.2963616.

[43] Z. Jiang, S. Yin, J. Seo, and M. Seok, "C3SRAM: An in-memory-computing SRAM macro based on robust capacitive coupling computing mechanism," *IEEE J. Solid-State Circuits*, vol. 55, no. 7, pp. 1888–1897, 2020, doi: 10.1109/JSSC.2020.2992886.

[44] R. Guo *et al.*, "A 5.1pJ/neuron 127.3us/inference RNN-based speech recognition processor using 16 computing-in-memory SRAM macros in 65nm CMOS," in *Proc. IEEE Symp. VLSI Circuits*, 2019, pp. C120–C121, doi: 10.23919/VLSIC.2019.8778028.

[45] J. Yue *et al.*, "A 2.75-to-75.9TOPS/W computing-in-memory NN processor supporting set-associate block-wise zero skipping and ping-pong CIM with simultaneous computation and weight updating," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2021, vol. 64, pp. 238–240, doi: 10.1109/ISSCC42613.2021.9365958.

[46] H. Jia *et al.*, "A programmable neural-network inference accelerator based on scalable in-memory computing," in *Proc. IEEE Int. Solid- State Circuits Conf. (ISSCC)*, 2021, vol. 64, pp. 236–238, doi: 10.1109/ISSCC42613.2021.9365788.

[47] S. Yin *et al.*, "PIMCA: A 3.4-Mb programmable in-memory computing accelerator in 28nm for on-chip DNN inference," in *Proc. IEEE Symp. VLSI Technol.*, 2021, pp. 1–2.

[48] N. R. Shanbhag and S. K. Roy, "Comprehending in-memory computing trends via proper benchmarking," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, 2022, pp. 1–7, doi: 10.1109/CICC53496.2022.9772817.

[49] I. A. Papistas *et al.*, "A 22 nm, 1540 TOP/s/W, 12.1 TOP/s/mm^2 in-memory analog matrix-vector-multiplier for DNN acceleration," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, 2021, pp. 1–2, doi: 10.1109/CICC51472.2021.9431575.

[50] J. Lee, H. Valavi, Y. Tang, and N. Verma, "Fully row/column-parallel in-memory computing SRAM macro employing capacitor-based mixed-signal computation with 5-b inputs," in *Proc. IEEE Symp. VLSI Circuits*, 2021, pp. 1–2, doi: 10.23919/VLSICircuits52068.2021.9492444.

[51] Z. Chen *et al.*, "CAP-RAM: A charge-domain in-memory computing 6T-SRAM for accurate and precision-programmable CNN inference," *IEEE J. Solid-State Circuits*, vol. 56, no. 6, pp. 1924–1935, 2021, doi: 10.1109/JSSC.2021.3056447.

[52] C. Yu, K. T. C. Chai, T. T.-H. Kim, and B. Kim, "A zero-skipping reconfigurable SRAM in-memory computing macro with binary-searching ADC," in *Proc. IEEE 51st Eur. Solid-State Device Res. Conf. (ESSDERC)*, 2021, pp. 131–134, doi: 10.1109/ESSDERC53440.2021.9631785.

[53] N. Verma *et al.*, "In-memory computing: Advances and prospects," *IEEE Solid State Circuits Mag.*, vol. 11, no. 3, pp. 43–55, 2019, doi: 10.1109/MSSC.2019.2922889.

[54] V. Joshi *et al.*, "Accurate deep neural network inference using computational phase-change memory," *Nature Commun.*, vol. 11, no. 1, pp. 1–13, 2020, doi: 10.1038/s41467-020-16108-9.

[55] S. K. Cherupally *et al.*, "Improving DNN hardware accuracy by in-memory computing noise injection," *IEEE Des. Test*, early access, 2021, doi: 10.1109/MDAT.2021.3139047.

[56] Y.-D. Chih *et al.*, "An 89TOPS/W and 16.3TOPS/mm^2 all-digital SRAM-based full-precision compute-in memory macro in 22nm for Machine-learning edge applications," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2021, pp. 252–254, doi: 10.1109/ISSCC42613.2021.9365766.

[57] J.-O. Seo, M. Seok, and S. Cho, "ARCHON: A 332.7TOPS/W 5b variation-tolerant analog CNN processor featuring analog neuronal computation unit and analog memory," in *Proc. IEEE Int. Solid- State Circuits Conf. (ISSCC)*, 2022, pp. 258–260, doi: 10.1109/ISSCC42614.2022.9731654.

[58] D. Wang, C.-T. Lin, G. K. Chen, P. Knag, R. K. Krishnamurthy, and M. Seok, "DIMC: 2219TOPS/W 2569F^2/b digital in-memory computing macro in 28nm based on approximate arithmetic hardware," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2022, pp. 266–268, doi: 10.1109/ISSCC42614.2022.9731659.

[59] J.-M. Hung *et al.*, "An 8-Mb DC-current-free binary-to-8b precision ReRAM nonvolatile computing-in-memory macro using time-space-readout with 1286.4-21.6TOPS/W for edge-AI devices," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2022, vol. 65, pp. 1–3, doi: 10.1109/ISSCC42614.2022.9731715.

[60] W. He *et al.*, "2-bit-per-cell RRAM-based in-memory computing for area-/energy-efficient deep learning," *IEEE Solid-State Circuits Lett.*, vol. 3, pp. 194–197, Jul. 2020, doi: 10.1109/LSSC.2020.3010795.

[61] P. Narayanan *et al.*, "Fully on-chip MAC at 14nm enabled by accurate row-wise programming of PCM-based weights and parallel vector-transport in duration-format," in *Proc. IEEE Symp. VLSI Technol.*, 2021, pp. 1–2.

[62] S. Jung *et al.*, "A crossbar array of magnetoresistive memory devices for in-memory computing," *Nature*, vol. 601, no. 7892, pp. 211–216, 2022, doi: 10.1038/s41586-021-04196-6.

[63] L. Fick, S. Skrzyniarz, M. Parikh, M. B. Henry, and D. Fick, "Analog matrix processor for edge AI real-time video analytics," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2022, pp. 260–262, doi: 10.1109/ISSCC42614.2022.9731773.

[64] M. Chang *et al.*, "A 40nm 60.64TOPS/W ECC-capable compute-in-memory/digital

2.25MB/768KB RRAM/SRAM system with embedded cortex M3 microprocessor for edge recommendation systems," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2022, pp. 1–3, doi: 10.1109/ISSCC42614.2022.9731679.

[65] J. Meng, L. Yang, X. Peng, S. Yu, D. Fan, and J. Seo, "Structured pruning of RRAM crossbars for efficient in-memory computing acceleration of deep neural networks," *IEEE Trans. Circuits Syst., II, Exp. Briefs*, vol. 68, no. 5, pp. 1576–1580, 2021, doi: 10.1109/TCSII.2021.3069011.

[66] B. Zhang *et al.*, "A 177 TOPS/W, capacitor-based in-memory computing SRAM macro with stepwise-charging/discharging DACs and sparsity-optimized bitcells for 4-bit deep convolutional neural networks," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, 2022, pp. 1–2, doi: 10.1109/CICC53496.2022.9772781.

About the Authors

Jae-sun Seo (jaesun.seo@asu.edu) received his Ph.D. degree from the University of Michigan in 2010. He is an associate professor in the School of Electrical, Computer, and Energy Engineering at Arizona State University (ASU), Tempe, Arizona, 85281, USA. He worked at IBM Research from 2010 to 2013 before joining ASU. His research interests include the efficient hardware design of machine learning algorithms and neuromorphic computing. He was a recipient of the IBM Outstanding Technical Achievement Award in 2012, National Science Foundation CAREER Award in 2017, Intel Outstanding Researcher Award in 2021, and IEEE Transactions on VLSI Systems Best Paper Award in 2022.

Jyotishman Saikia (jsaikia@asu.edu) received his B.Tech. degree in electronics and telecommunication engineering from Kalinga Institute of Industrial Technology University, Bhubaneswar, India, in 2016 and his M.S. degree in computer engineering from Arizona State University (ASU) in 2019. He is currently pursuing a doctoral degree in electrical engineering at ASU, Tempe, Arizona, 85281, USA. His research interests include the design of memory systems, low-power design, and the in-memory computation-based implementation of deep neural network algorithms.

Jian Meng (jmeng15@asu.edu) received his B.S. degree from Portland State University, Portland, USA, in 2019. He is currently pursuing the Ph.D. degree with the School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, Arizona, 85281, USA. His research interests include deep neural network compression optimization, hardware–software co-design with neuromorphic hardware acceleration, and neuromorphic algorithm design for event-based vision.

Wangxin He (wangxinh@asu.edu) received his B.E. in microelectronics at Xiamen University, Xiamen, China, in 2017 and his M.S. degree in electrical engineering from the University of Illinois at Chicago, Chicago, Illinois, USA, in 2019. He is currently pursuing a Ph.D. degree in electrical engineering at Arizona State University, Tempe, Arizona, 85281, USA. His research interests include NVM mixed-signal system design and testing for energy-efficient artificial intelligence hardware and neuromorphic computing acceleration, with a focus on RRAM-related research.

Han-sok Suh (hsuh6@asu.edu) graduated from Inha University summa cum laude, and he received his master's degree from the University of Southern California Viterbi School of Engineering in computer engineering. At Arizona State University, Tempe, Arizona, 85281, USA, he is pursuing a Ph.D. degree and working on computer architecture design on FPGAs. His research interests include software–hardware co-design and energy-efficient computing for machine learning applications.

Anupreetham (anolas11@asu.edu) received his B.Tech. degree in electronics and communication engineering from the National Institute of Technology Karnataka, India, and his M.S. degree in computer engineering from Arizona State University. He is a Ph.D. candidate at the School of Electrical, Computer, and Energy Engineering at Arizona State University (ASU), Tempe, Arizona, 85281, USA, supervised by Prof. Jae-sun Seo. He currently works as a graduate research associate at the Seo Lab at ASU. His research interests include FPGA design for real-time machine learning applications and spiking neural network hardware implementation.

Yuan Liao (yliao59@asu.edu) received his B.S. and M.S. degrees, both in electrical engineering, from the University of Washington, Seattle, in 2019 and 2021, respectively. He joined the Seo Lab in August 2021 and currently is a Ph.D. student in electrical, computer, and energy engineering at Arizona State University, Tempe, Arizona, 85281, USA. His research interests include efficient hardware design for machine learning algorithms.

Ahmed Hasssan (ahasssan@asu.edu) received his B.S. degree in electrical engineering from COMSATS University Islamabad, Pakistan, in 2015 and his M.S. degree in electrical engineering from Government College University Lahore, Pakistan, in 2019. He served as lecturer in the Department of Electrical Engineering, Sharif College of Engineering and Technology, Lahore, from 2017 to 2020. He is currently enrolled in the Ph.D. degree program at the Ira Fulton School of Engineering, Arizona State University, Tempe, Arizona, 85281, USA. He has published papers in peer-reviewed journals and conferences. His research interests include neuromorphic computing-based hardware design. In particular, he is working on low-precision sparse neural networks for dynamic vision sensors. He is a Member of IEEE Young Professionals.

Injune Yeo (iyeo3@asu.edu) received his B.S. degree in semiconductor science from Dongguk University, Seoul, South Korea, in 2011 as well as his master's degree in mechatronics engineering and Ph.D. degree in electrical engineering from Gwangju Institute of Science and Technology, Gwangju, South Korea, in 2014 and 2020, respectively. He is now a postdoctoral scholar in the School of Electrical, Computer, and Energy Engineering at Arizona State University, Tempe, Arizona, 85281, USA. His research interests include ADCs and in-memory computing with NVM.