

# We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,500

Open access books available

176,000

International authors and editors

190M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index  
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?  
Contact [book.department@intechopen.com](mailto:book.department@intechopen.com)

Numbers displayed above are based on latest data collected.  
For more information visit [www.intechopen.com](http://www.intechopen.com)



# End-to-End Benchmarking of Chiplet-Based In-Memory Computing

*Gokul Krishnan, Sumit K. Mandal, A. Alper Goksoy,  
Zhenyu Wang, Chaitali Chakrabarti, Jae-sun Seo,  
Umit Y. Ogras and Yu Cao*

## Abstract

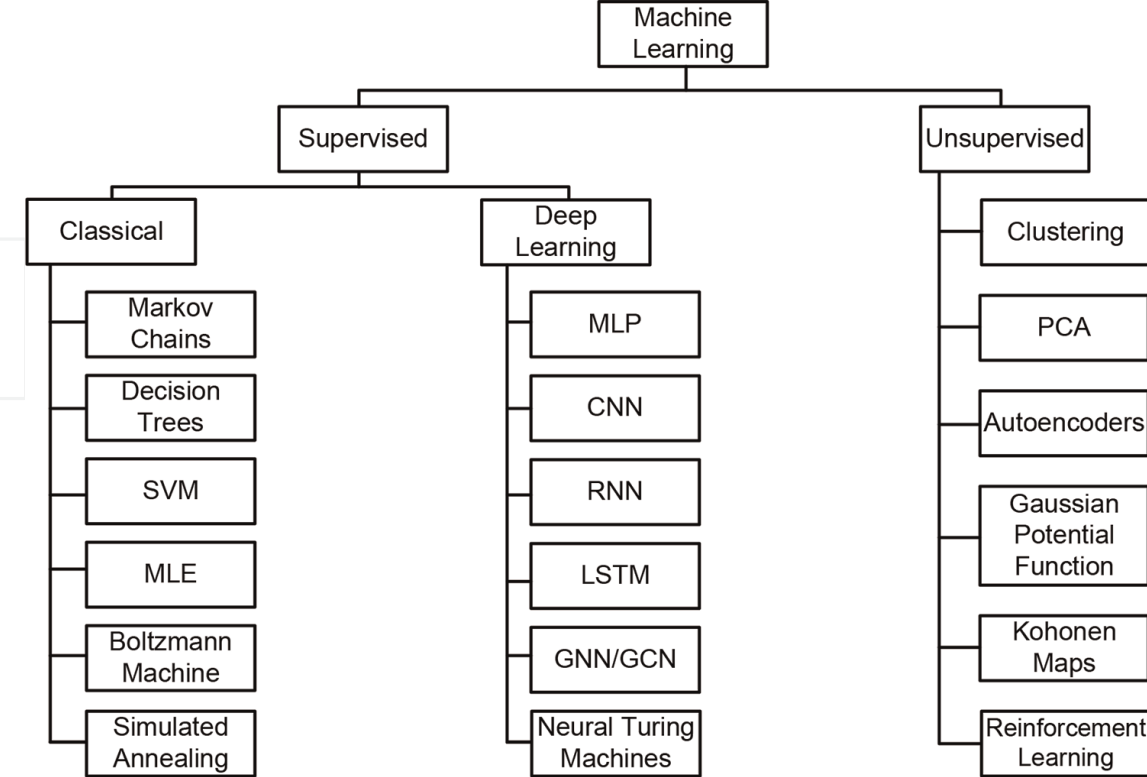
In-memory computing (IMC)-based hardware reduces latency and energy consumption for compute-intensive machine learning (ML) applications. Several SRAM/RRAM-based IMC hardware architectures to accelerate ML applications have been proposed in the literature. However, crossbar-based IMC hardware poses several design challenges. We first discuss the different ML algorithms recently adopted in the literature. We then discuss the hardware implications of ML algorithms. Next, we elucidate the need for IMC architecture and the different components within a conventional IMC architecture. After that, we introduce the need for 2.5D or chiplet-based architectures. We then discuss the different benchmarking simulators proposed for monolithic IMC architectures. Finally, we describe an end-to-end chiplet-based IMC benchmarking simulator, SIAM.

**Keywords:** in-memory compute, SRAM, RRAM, network-on-chip, network-on-package, convolutional neural networks, artificial intelligence

## 1. Introduction

### 1.1 Modern-day AI algorithms

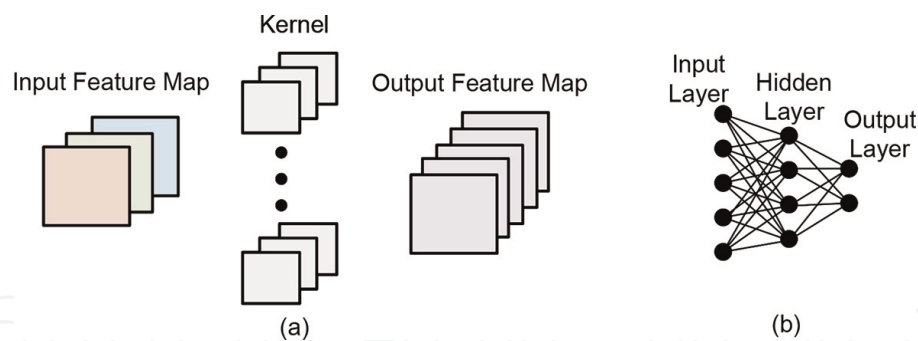
Machine learning (ML) and artificial intelligence (AI) significantly impacted society. AI algorithms, such as deep neural networks (DNNs), achieved accuracy that exceeded human-level perception for various applications, including medical imaging, natural language processing, and computer vision [1–3]. The popularity of AI algorithms was mainly driven by the availability of big datasets (for various applications like image classification and object detection) [1, 4, 5], as well as the increased computing power provided by the next-generation ML hardware accelerators and general-purpose computing platforms. **Figure 1** illustrates the taxonomy of ML algorithms, which could be broadly categorized into supervised and unsupervised



**Figure 1.**  
*Taxonomy showing different ML with different learning techniques.*

learning. Unsupervised learning involved extracting features from a distribution without data annotation, including selecting samples, denoising data, and clustering data into groups. The unsupervised learning algorithm aimed to find the optimal representation of the data that preserved maximum information about the input data  $x$  while ensuring the representation was simpler than the data itself, utilizing constraints such as lower-dimensional, sparse, or independent representation [6, 7]. Some popular unsupervised learning techniques were clustering, principal component analysis (PCA), autoencoders, and Gaussian potential functions.

Supervised learning involves training the ML model using a set of labeled training data and then testing it using a labeled testing set. There are two types of supervised learning: classical approaches and deep learning. Classical approaches use conventional techniques that employ a probabilistic model to determine the next state based on a set of parameters. Some popular classical techniques are decision trees, support vector machines (SVM), Markov chains, and maximum likelihood estimation (MLE) [8–11]. However, classical techniques had limitations such as difficulty in scaling, lack of generalization, and the need for significant data engineering for each algorithm. Classical techniques serve as the foundation for deep learning algorithms, which overcome their limitations. This chapter focuses on deep learning techniques used in supervised learning. Convolutional neural networks (CNNs) stand out for their superior performance in various machine-learning tasks, including computer vision, object detection, and object segmentation. Additionally, recurrent neural networks (RNNs) excel in processing temporal data, while graph convolutional networks (GCNs) combine graphs and neural networks for various applications. The chapter covers recent advancements in CNNs, RNNs, and GCNs, discussing their structures, training methods, and execution efficiency for training and inference operations.



**Figure 2.**  
 (a) Convolution layer consisting of the input feature map (IFM), kernel, and the output feature map (OFM),  
 (b) FC layer in a CNN.

Conventional CNNs consist of layers connected either sequentially or with skip connections. Besides convolutional layers, ReLU, pooling, and batch-normalization methods are commonly used to enhance performance. **Figure 2** illustrates the typical structure of a convolutional and fully connected (FC) layer. Sequential layers usually involve a stack of convolution (conv) layers that extract features from the input. Convolution layer kernels may include  $7 \times 7$ ,  $5 \times 5$ ,  $3 \times 3$ , and  $1 \times 1$ . Additionally, as proposed in MobileNet [12], depthwise convolutions divide a given  $N \times N$  convolution into two parts: first, a  $N \times 1$  convolution is performed, followed by a  $1 \times N$  convolution. Depth-wise convolution yields better accuracy and lower hardware complexity. Pooling layers are periodically utilized to reduce the feature map size and eliminate noisy input. In order to perform classification on extracted features, a set of FC layers or classifier layers are utilized. These layers, along with the Conv layers, have a set of weights that are trained to achieve the best accuracy. Popular CNN structures, such as AlexNet [1], GoogleNet [13], ResNet [14], DenseNet [15], MobileNet [12], and SqueezeNet [16], utilize a combination of convolutional, pooling, and FC layers to achieve high accuracy in a variety of ML tasks. Additionally, CNNs like DenseNet and ResNet utilize skip connections from previous layers to create a highly branched structure, which aims to improve the feature extraction process. These skip connections are present within the conv layers only. In contrast, conventional CNNs face several limitations, such as the vanishing gradient problem, higher hardware costs during training and inference, and over-parameterization [17–19]. To address these issues, network architecture search (NAS) was introduced to automatically find the optimal neural network architecture based on the specific design point for the target application. Different design points, such as higher accuracy, better generalization, higher hardware efficiency, and lower memory access, have been proposed for NAS. Popular techniques like NasNet [20], FBNet [21], AmoebaNet [22], PNAS [23], ECNAS [24], and MNasNet [25] have been developed to address these issues.

RNN is a commonly used deep learning technique that offers an effective solution for modeling data with sequential or temporal structures and variable length inputs and outputs in various applications [26–28]. It processes sequential data one element at a time, using a connectionist model that selectively passes information. This enables RNNs to model input and/or output data consisting of a sequence of dependent elements. Additionally, RNNs can simultaneously model both sequential and time dependencies at different scales. They employ a feedforward network that includes edges connecting adjacent time steps, which introduces time to the model. While conventional edges do not have cycles, recurrent edges that connect adjacent time

steps can form cycles. Modern RNN architectures can be categorized into two main types. The first is long-short-term memory (LSTM), which includes a memory cell, a computation unit that replaces traditional nodes in the hidden layer of a network [29]. The second variant of RNNs is bi-directional RNNs (BRNNs), as proposed in [30].

As more applications rely on graphs to represent data, using CNNs and RNNs to capture hidden patterns within Euclidean data becomes limited. For instance, in e-commerce, a graph-based learning system can use interactions between users and products to recommend highly accurate products. However, graphs' complexity and irregularities pose significant challenges to existing DNNs. To address this issue, Graph Neural Networks (GNNs) were introduced and categorized into three types: Recurrent GNNs (RecGNNs) [31–33], Convolutional GNNs (CGNNs) [34–36], and Graph Autoencoders (GAE) [37–39]. RecGNNs use recurrent neural architectures to learn node representations, while CGNNs generalize convolution operations to graph data by aggregating features from neighboring nodes. GAEs map nodes into a latent feature space, preserving the node's topological information with a low-dimensional vector. Finally, GAEs learn network embeddings using an encoder and a decoder to enforce the preservation of graph topological information.

## 1.2 Hardware implications of DNNs

State-of-the-art DNNs, such as CNNs, RNNs, and GCNs, have diverse structures that lead to significant demands on compute and memory resources. Achieving higher accuracy with these machine-learning models requires increased computational complexity and model size, which, in turn, require more memory to store both the weights and activations. The increased model size and complexity also lead to a larger volume of on-chip data movement. For instance, the ImageNet dataset's [1] ResNet-50 [14] requires 50 MB of memory and 4 GFLOPs per inference, while DenseNet-121 [15] requires 110 MB of memory and 8 GFLOPs per inference. Conventional architectures that separate memory and computation lead to a considerable number of external memory accesses, which reduce energy efficiency and performance. The average cost of an external memory access is 1000 times higher than the energy required for computations [40]. When considering the total energy spent on performing inference for VGG-16 and ResNet-50 using conventional von Neumann architectures, a floating-point 32-bit (FP-32) multiplication results in 3.2pJ, while an FP-32 add requires 0.9pJ in the 45 nm technology node [41]. Therefore, performing inference for one image consumes 65 mJ of energy using the VGG-16 CNN, while ResNet-50 consumes 16 mJ. Scaling the computation energy up, for 1000 inference performs, VGG-16 takes 65 J while ResNet-50 consumes 16 J of energy. In conclusion, the higher accuracy achieved by DNNs results in higher computational complexity, increased memory requirements, more off-chip memory access, and lower energy efficiency.

This chapter delves into in-memory computing (IMC) as an alternative to traditional von-Neumann architecture, which offers improved energy efficiency, better performance, and reduced off-chip memory access. IMC has emerged as a promising solution to tackle the memory access, energy efficiency, and performance bottlenecks encountered by DNN applications. Hardware architectures for IMC, such as those based on SRAM and nanoscale nonvolatile memory (e.g., resistive RAM or RRAM), provide a dense and parallel structure to achieve high performance and energy efficiency [42–57]. Additionally, the chapter introduces chiplet-based IMC architectures, as well as a benchmarking simulator for this architecture. The SIAM simulator and the



associated architecture enabled through SIAM are described in detail. Overall, this chapter explores various benchmarking simulators for IMC architectures.

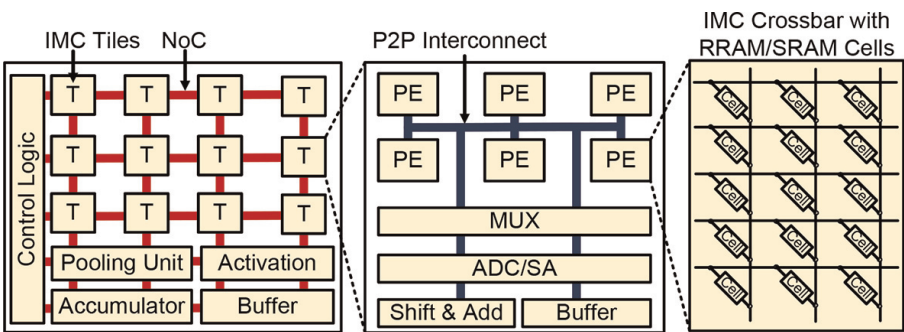
## 2. IMC architectures

In the preceding section, we examined the hardware implications of modern DNNs, specifically the memory and computation complexity associated with von Neumann architectures. For instance, dense structures such as DenseNet require roughly  $2.7 \times 10^7$  off-chip memory accesses to process an image frame [15]. This elevated number of off-chip memory access negatively impacts the energy efficiency of the overall system. IMC architectures provide a promising alternative to traditional von Neumann architectures. **Figure 3** depicts a generic block diagram of an IMC architecture with RRAM/SRAM memory cells. IMC uses analog- or digital-domain computation to carry out multiply-and-accumulate (MAC) operations. The crossbar-based IMC structure efficiently combines memory access and analog-domain computation into a single unit, resulting in faster execution of DNN workloads. The superior energy efficiency is primarily due to a full-custom design, higher density, and higher memory bandwidth [44, 45, 58]. Consequently, IMC-based systems are becoming increasingly popular for implementing compute- and memory-intensive AI applications. This section will examine various IMC architectures in depth using both SRAM and RRAM memory cells.

### 2.1 RRAM/SRAM-based IMC architectures

#### 2.1.1 RRAM Device

RRAM-based IMC architectures consist of an RRAM memory cell at each crosspoint within the IMC crossbar array. RRAM is a two-terminal device with programmable resistance representing the neural network's weights. It has high integration density, fast read speed, high memory accessing bandwidth, and good compatibility with CMOS fabrication technology. For example, the RRAM device stack can include a TiN bottom electrode, HfO<sub>2</sub> mem-resistive switching layer, a PVD Ti oxygen exchange layer (OEL), and  $\sim 40$  nm TiN top electrode [59, 60]. This specific stack is implemented between the M1 and M2 metallization layers using a FEOL-compatible process flow.



**Figure 3.** Generic block diagram of an IMC architecture for DNN acceleration. It consists of an array of IMC tiles connected by an NoC with each tile consisting of a number of IMC arrays [44, 45].

Each RRAM cell can be characterized by the number of resistance levels accessed within them. Broadly, RRAM can be classified into single-level cells (SLC) and multi-level cells (MLC). SLC only has two resistance levels, that is, they can store only binary data. On the other hand, MLC cells have multiple resistance levels that represent higher precision data. The number of available resistance levels is governed by the ratio of the off resistance ( $R_{\text{off}}$ ) to the on resistance ( $R_{\text{on}}$ ) ratio [61]. The ratio provides the range of resistances accessible for the given RRAM device. The overall resistance range can be divided into two main states, a low resistance state (LRS) and a high resistance state (HRS). LRS deals with the lower spectrum within the resistance band, while the HRS deals with the upper band of resistance of the RRAM device.

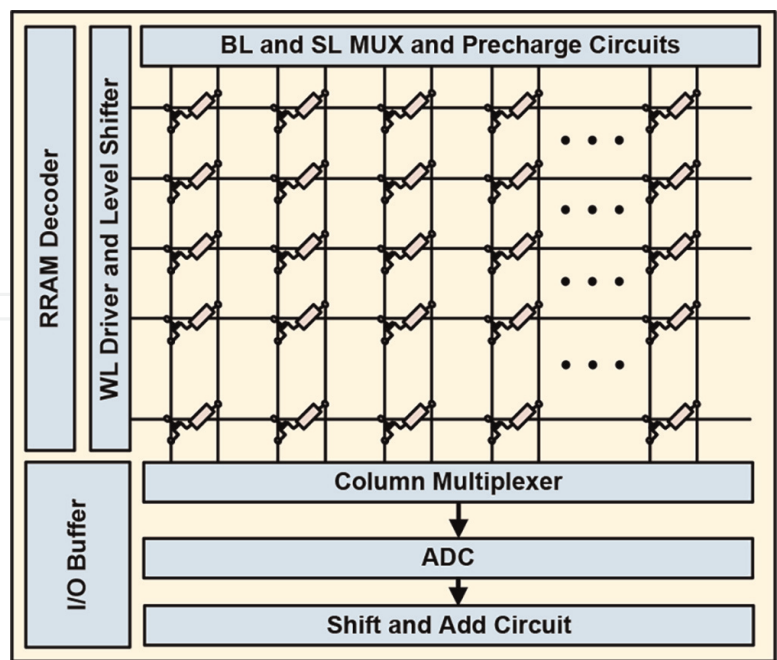
To program the RRAM device, a series of steps need to be followed [62]. First, the RRAM device is formed by applying a large voltage across the two terminals. This process breaks the barrier and allows for electron flow across the terminals. Next, the RRAM is programmed to the required resistance by passing a specific current (compliance current) through the two electrodes. The RRAM can be programmed at different resistances depending on the compliance current. Furthermore, different resistance levels can be achieved depending on the RRAM device (SLC or MLC). Finally, once the RRAM device is programmed, we can perform a read by applying a voltage across the device electrodes. For the RRAM device proposed in [59, 60], a read voltage of up to 0.4 V can be sustained by the RRAM device. Applying a higher voltage damages the device or goes into the write state, changing the programmed resistance level.

### 2.1.2 IMC Architecture

Studies involving crossbar architectures have demonstrated that a  $100\times$  to  $1000\times$  improvement in energy efficiency is achieved as compared to traditional CPU and GPU architectures [44, 45, 47, 49, 50, 52, 63–67]. **Figure 3** shows the block diagram of a IMC architecture with an RRAM/SRAM memory cell. The architecture consists of an array of IMC tiles connected by a network-on-chip (NoC). The architecture also consists of a global pooling unit, nonlinear activation unit, accumulator, and input/output buffers. A global control logic performs the architecture's overall handling of the blocks.

Each tile consists of an array of processing elements (PEs), where each PE is an IMC crossbar array with either an SRAM or an RRAM cell. Each IMC crossbar array consists of a set of peripheral circuits that enable the MAC computations.

**Figure 4** shows the generic block diagram for a single RRAM-based IMC crossbar array. In the case of RRAM IMC, a transistor connects the gate to the wordline (WL) of the IMC crossbar array [60]. The WL connects to the access transistors for the SRAM-based IMC with a conventional 6 T structure. The IMC crossbar arrays consist of a wordline (WL) decoder, WL driver, a column multiplexer, analog-to-digital converter (ADC) or a sense amplifier, shifter and add circuit, control logic, and input/output buffers. The WL decoder turns on and off the WL for the IMC crossbar array. Meanwhile, the WL driver and level shifter ensure the driver can turn on the memory cell. Next, for an  $N \times N$  IMC crossbar array,  $M$  columns are shared across the read-out circuit. The read-out circuit consists of the ADC, shift and add circuit, and the precharge circuit for the read operation. To enable the sharing of  $M$  columns, a column multiplexer is used. Finally, a custom control logic is utilized to drive the control signals during the operation of the IMC crossbar array. We will now go over the operation for both the SRAM and RRAM-based IMC architectures. First, we will



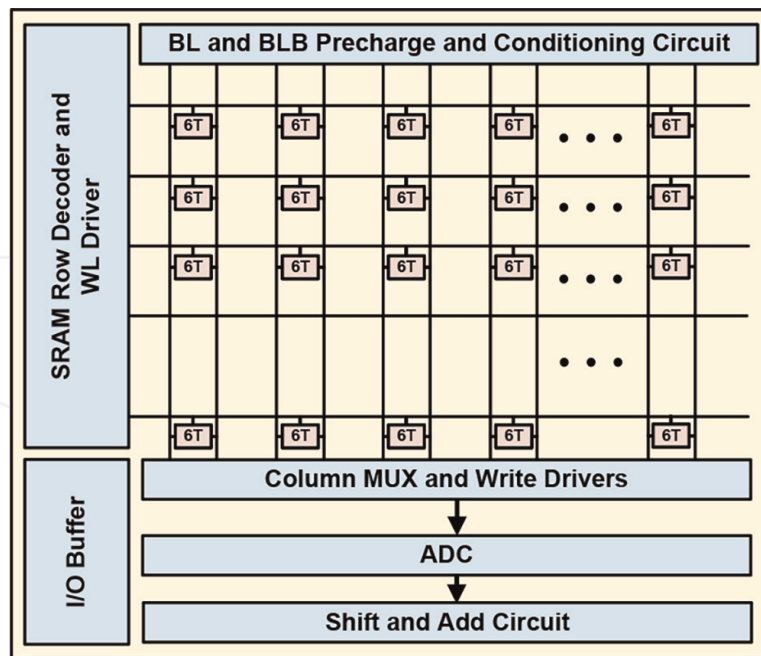
**Figure 4.** Block diagram of a RRAM-based IMC crossbar array. An array of RRAM cells form the IMC crossbar array. Peripheral circuits such as bitline (BL)/select line (SL)/column multiplexer (MUX), precharge circuit, wordline (WL) decoder and driver, buffers, level shifters, ADC, and shift and add circuit complete the RRAM-based IMC.

detail the working of the RRAM-based IMC architecture. **Figure 4** shows the generic block diagram for a single RRAM-based IMC crossbar array. The RRAM devices are programmed by connecting the two terminals to a given voltage. To facilitate this, the terminals are connected to the bitline (BL) and the selectline (SL). By applying a voltage across the BL and the SL, forming, programming, and reading operations are performed cell-by-cell. Each cell is chosen during the write state of the IMC, and then the write is performed. During the compute state, the RRAM undergoes the read operation. Two kinds of read-out are performed, parallel and serial. During the parallel read-out, all/multiple WLs are turned on simultaneously, and the output is accumulated across the BL. Two kinds of input schemes are employed for single and multi-bit inputs. The first method uses a digital-to-analog converter (DAC) to convert the input vector to an analog voltage and performs the computation in the charge domain [44]. The second method is to perform bit-serial computing, where each bit in the input vector is computed one at a time. Each input vector's bit significance is handled using a shift and add circuit [45, 49, 63].

Depending on the resistance stored in the RRAM, an output current/charge is generated by the product of the voltage and resistance (conductance). This operation is analogous to the multiply with the MAC. This current/charge is then accumulated across all rows for a given column to perform the addition in the MAC. In the case of the serial read-out, row-wise access of the IMC array is performed for MAC computations. Overall, the final MAC output is generated by accumulating across all rows of the IMC crossbar array.

**Figure 5** shows the generic block diagram for a single SRAM-based IMC crossbar array. Next, we will discuss the operation for an SRAM-based IMC architecture [48, 49, 51, 52, 68–70]. Depending on the SRAM bit-cell type and the degree of parallelism, the IMC design can be largely divided into three categories [71]: 6 T bit-cell with parallel computing, 6 T bit-cell with local computing, and (6 T + extra-T) bit-cell with parallel computing. Initially, SRAM-based IMC architecture employed the



**Figure 5.**

Block diagram of a SRAM-based IMC crossbar array. An array of SRAM cells 6 T or 6 T + additional circuit) form the IMC crossbar array. Peripheral circuits such as bitline (BL) and bitline bar (BLB) precharge and conditioning circuits, row decoder and WL driver, column multiplexers, buffers, write drivers, ADC, and shift and add circuit complete the SRAM-based IMC.

6 T bit-cell with a parallel computation [72, 73]. The parallel computation was achieved by turning on all the WL together to perform the MAC operations. The WL is driven by the input vector where a one means it turns on that cell, while a zero means the cell is turned off. Next, a 6 T bit-cell with a local compute structure is utilized where a special compute engine is designed to perform the MAC operation [70]. Here, the MAC operation is performed row-by-row, similar to the serial read-out in RRAM-based IMC. Finally, in addition to the 6 T cell, extra transistors are added in each bit-cell to perform parallel compute [51, 68, 69]. In addition to the bit-cell structure, peripheral circuits such as precharge circuits, ADCs, write drivers, column multiplexers, row decoders, and row drivers are used.

### 2.1.3 Challenges with IMC Architectures

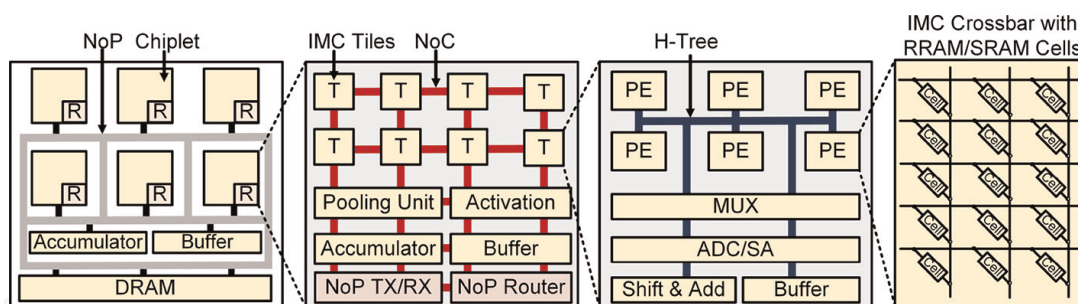
IMC architectures are known for their improved energy efficiency and throughput, but they have some drawbacks. One such drawback is the limited precision of the IMC crossbar array, particularly in the memory cell and ADC, which can affect the accuracy of DNN inference [74, 75]. Additionally, noise within analog computation can also harm DNN inference accuracy. The challenges associated with an RRAM-based IMC architecture are discussed next. RRAM devices suffer from several non-idealities, including limited resistance levels, device-to-device write variations, stuck-at-faults, and limited  $R_{\text{off}}/R_{\text{on}}$  ratio, which make it difficult to design reliable RRAM-based IMC architectures [60, 61, 76–84]. These non-idealities can cause programmed weight values (resistance value) to deviate, significantly reducing post-mapping accuracy for DNNs. Moreover, the limited array size of the IMC crossbar structure necessitates the splitting of large convolution (conv) or FC layers into partial operations, which can introduce additional errors due to the limited precision of the peripheral circuits (ADC and shift and add) of the RRAM-based IMC crossbar.

Previous research has proposed several methods to address the post-mapping accuracy loss associated with RRAM-based in-memory acceleration of DNNs. Two methods that have been proposed are Closed-Loop-on-Device (CLD) and Open-Loop-off-Device (OLD), which involve iterative read-verify-write (R-V-W) operations at the RRAM device until the resistance converges to the desired value [85, 86]. Other methods, such as in References [78, 87], utilize variation-aware-training (VAT) based on known device variation ( $\sigma$ ) characterized from RRAM devices. In [76], VAT is combined with dynamic precision quantization to mitigate the post-mapping accuracy loss. Another approach proposed in [75] involves injecting RRAM macro measurement results that include variability and noise during the DNN training process to improve the DNN accuracy of the RRAM IMC hardware. [88] proposes post-mapping training by selecting a random subset of weights and mapping them to an on-chip memory to recover the accuracy. [79] utilizes knowledge distillation and online adaptation for accuracy mitigation, using an SRAM-based IMC as the parallel network, while [88] proposes to use a register file and a randomization circuit. Finally, [77, 80] propose a custom unary mapping scheme by mapping the most significant bit (MSB) and least significant bit (LSB) of the weights to RRAM devices based on individual cell variations and bit significance.

Next, we discuss the challenges associated with SRAM-based IMC architectures. A compromise between parallelism and reliability is employed for best performance. In a conventional 6 T SRAM IMC architecture, parallel computation is achieved by turning on all or multiple rows. The higher parallelism raises the critical issue of read disturbance, resulting in the WL voltage being driven with a lower voltage [72, 73, 89]. To mitigate this, reduced parallelism is employed by exploiting the local compute engine [70]. The reduced parallelism results in reduced throughput for DNN inference. [51, 68, 69] proposes to utilize additional transistors that isolate the bit-cell and employ parallel computation. Such a solution comes at the cost of increased area overhead, thus limiting the density of the SRAM-based IMC architecture. The additional transistor solution is typically implemented using a resistance or a capacitance. The resistive IMC method implements a multi-bit MAC operation by utilizing a resistive pull-up/down using transistors [51, 68, 69]. The pull-up/down characteristics of the transistors exhibit a non-linear behavior for the read bitline (RBL) transfer curve across different voltage ranges, thus reducing reliability. At the same time, the capacitive SRAM-based IMC utilizes a capacitor per bit-cell and charge sharing and capacitive coupling to perform the MAC operations [52]. The capacitive SRAM IMC exhibits a more linear transfer characteristic on the RBL but at the cost of a capacitor per bit-cell. Finally, the limited precision of the ADC and the noise on the bitline (BL) requires careful algorithm design to achieve the best inference accuracy [89].

## 2.2 2.5D/Chiplet-based AI accelerators

The area of monolithic hardware accelerators increases with the increasing number of parameters of AI algorithms. The higher silicon area of a single monolithic system reduces the yield, increasing fabrication cost [46]. The chiplet-based system solves the issue of higher fabrication cost by integrating multiple small chips (known as chiplets) on a single die. Since the area of each chiplet in the system is considerably lower than a monolithic chip (for the same AI algorithm), the yield of the chiplet-based system increases, which reduces the fabrication cost. The communication between chiplets is performed through network-on-package (NoP), as shown in **Figure 6**. Several works



**Figure 6.**

Chiplet-based IMC architecture [46, 64] that includes an NoP for on-package communication, NoC for on-chip communication within each chiplet, and a point-to-point network like H-Tree for within tile communication.

in the literature propose NoP for chiplet-based systems considering different performance objectives (e.g., latency, energy) [42, 46, 90–92].

Kite is a family of NoP proposed in [90], mainly targeted for general-purpose processors. In this work, three topologies are proposed—Kite-Small, Kite-Medium, and Kite-Large. First, an objective function is constructed with a combination of the average delay between the source and destination, diameter, and bisection bandwidth of the NoP. Experimental evaluations on synthetic traffic show that the proposed Kite topologies reduce latency by 7% and improve the peak throughput by 17% with respect to other well-known interconnect topologies. A chiplet-based system with a 96-core processor, INTACT, is proposed in [91]. The chiplets are connected through a generic chiplet-interposer interface (called 3D-plugs in the paper). 3D-plugs consist of micro-bump arrays. However, both Kite and INTACT are not specific to AI workloads.

Shao et al. designed and fabricated a 36-chiplet system called SIMBA for deep learning inference [92]. The chiplets in the system are connected through a mesh NoP. Ground-referenced signaling (GRS) is used for intra-package communication. The NoP follows a hybrid wormhole/cut-through flow control. The NoP bandwidth is 100 GBps/chiplet, and the latency for one hop is 20 ns. Extensive evaluation on the fabricated chip shows up to 16% speed up compared to baseline layer mapping for ResNet-50. A simulator for the chiplet-based systems, SIAM, is proposed in [46], targeting AI workloads. In this simulator, a mesh topology is considered for NoP. It is shown that up to 85% of the total system area is contributed by NoP. In this work, multiple studies were performed by varying NoP parameters. For example, it is shown that increasing the NoP channel width increases the energy-delay product of the NoP for ResNet-110. This phenomenon is demonstrated for systems with 25 and 36 chiplets. However, none of the prior works considered any workload-aware optimization for the NoP. Therefore, there is ample opportunity for future research considering NoP optimization for AI accelerators.

### 3. Benchmarking simulators

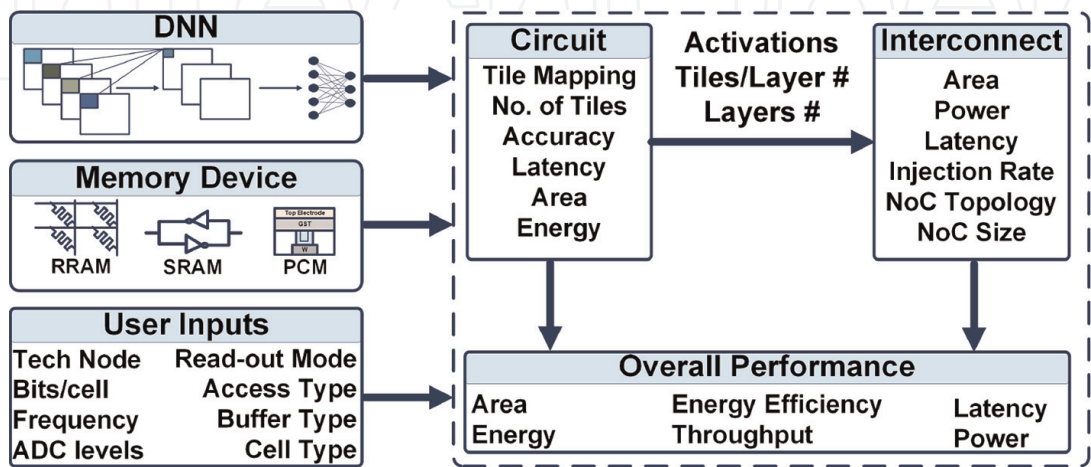
#### 3.1 Monolithic AI benchmarking simulators

Numerous researchers have put forth assessment frameworks for evaluating the performance of IMC-based AI accelerators. One such framework is NeuroSim [93], the first simulator designed to evaluate the performance of IMC-based AI accelerators. It includes various performance evaluation metrics such as area, latency, and power



consumption of an IMC system under a given workload of DNN. NeuroSim is highly flexible and allows users to assess the performance of IMC-based AI accelerators under different system specifications, considering both traditional CMOS-based memory technology (such as SRAM) and emerging nonvolatile memory technologies (like ReRAM and STTMRAM) for the in-memory compute elements. NeuroSim assumes a tile-based architecture [44], consisting of multiple tiles where processing elements (PEs) and IMC-based crossbar arrays reside. Predictive Technology Model (PTM) [94] is used to simulate lower-level components, such as buffers, ADC, and multiplexers, and verified against circuit simulation (e.g., SPICE), reaching more than 90% accuracy. The interface between NeuroSim and popular ML frameworks such as PyTorch and TensorFlow has also been created to make it more user-friendly [95]. However, one major drawback of NeuroSim is that it assumes H-Tree based bus interconnect for inter-tile communication, which can consume up to 90% of the total energy consumption of DNN inference [96]. To overcome this issue, Krishnan et al. [97] (**Figure 7**) proposed an evaluation framework for IMC-based AI accelerators that considers cycle-accurate network-on-chip (NoC) simulation [98]. Similarly, MNSIM [99] also evaluates the performance of IMC-based systems to execute AI applications like NeuroSim. In addition, MNSIM integrates software-hardware co-design techniques in the evaluation framework. GeneiX, proposed by Chakraborty et al. [78], is another evaluation framework for crossbar-based IMC accelerators considering the non-idealities in the memory elements.

While it is essential to evaluate hardware performance under AI workloads, it is also important to assess the accuracy of the AI workload when implemented on-chip. Memory imperfections can lower the accuracy of DNNs. RxNN [100] and SpikeSim [66] are frameworks that evaluate the accuracy of DNN workloads in the presence of memory imperfections. These techniques focus on evaluating the performance of IMC systems executing DNN inference. However, emerging edge devices require online learning, which involves training the DNN. Therefore, evaluating the performance of AI accelerators while executing DNN inference alone is insufficient. An evaluation framework for IMC-based AI accelerators with on-chip training is introduced in [101]. The authors incorporate non-linearity, asymmetry, device-to-device, and cycle-to-cycle variation of weight update into the Python wrapper and peripheral circuits for error/weight gradient computation in NeuroSim core for a given AI



**Figure 7.** Block diagram on an IMC benchmarking simulator proposed in [97]. The simulator consists of a circuit part and an interconnect part that perform system-level benchmarking of IMC architectures.



workload. The training framework is based on the authors' prior work [102], where they proposed an SRAM-based transposable function. Since SRAM-based arrays can perform write operations quickly while consuming minimal energy, the weight-gradient computation function is implemented through SRAM-based arrays rather than other nonvolatile memory technologies.

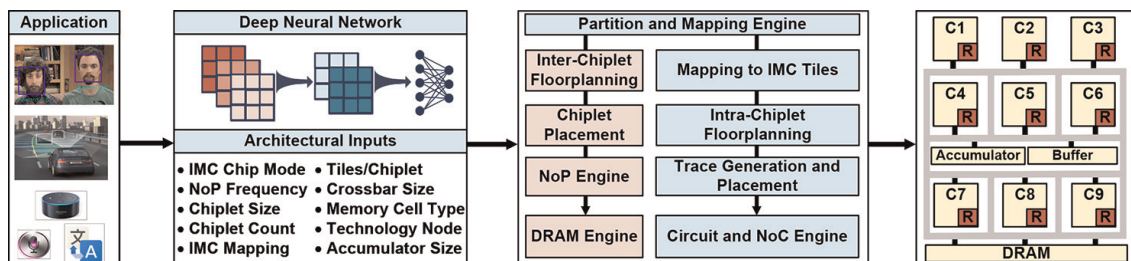
### 3.2 SIAM: Chiplet-based AI benchmarking simulator

This section introduces SIAM [46], a benchmarking simulator for IMC architectures based on chiplets. SIAM supports both generic or homogeneous and custom chiplet-based IMC architectures. A homogeneous architecture has a fixed number of chiplets that the user determines. On the other hand, a custom architecture comprises a specific number of chiplets required to map the DNN under consideration. In both cases, the chiplet structure contains a fixed number of user-defined IMC crossbar arrays. **Figure 6** illustrates a homogeneous chiplet-based IMC architecture SIAM uses.

**Figure 8** illustrates how the comprehensive framework provided by SIAM can be used to benchmark the performance of chiplet-based in-memory compute (IMC) architectures. SIAM generates a chiplet-based IMC architecture based on user-defined inputs. It assesses the corresponding hardware performance by computing various performance metrics, including area, energy, latency, energy efficiency, power, leakage energy, and IMC utilization. The SIAM framework is developed using Python and C++ programming languages and has a top-level Python wrapper that integrates the different components of the simulator. Additionally, SIAM is designed to interface with widely used deep learning frameworks, such as PyTorch and TensorFlow, and supports a range of network structures in current literature. Therefore, SIAM can be used to explore neural architecture search (NAS) techniques. **Table 1** describes the user inputs and their associated parameters for the SIAM benchmarking tool. SIAM consists of four engines:

- Partition and mapping engine (Python)
- Circuit and NoC engine (C++)
- NoP engine (Python and C++)
- DRAM engine (Python and C++)

The individual engines within SIAM operate independently on various subsets of user inputs and communicate with each other through the top-level Python wrapper. **Figure 8** provides an overview of the simulation flow used by SIAM to provide a



**Figure 8.**  
Block diagram of the chiplet-based IMC architecture simulator SIAM [46].

User input	Description	User input	Description
Intra-chiplet architecture		Inter-chiplet architecture	
ADC resolution	Bit-precision of flash ADC	Chiplet size	Number of IMC tiles within each chiplet
Read-out method	Sequential or parallel	Total chiplet count	Fixed count or DNN specific custom count
Crossbar size	IMC crossbar array size	Chip mode	Monolithic or chiplet-based IMC architecture
Buffer type	SRAM or register file	Chiplet structure	Homogeneous or custom chiplet structure
NoC width	Number of channels in the NoC	NoP frequency	Frequency of the NoP driver and interconnect
NoC topology	Mesh or tree	Global accumulator size	Size of global accumulator
Frequency	Frequency of operation	NoP channel width	Number of parallel links for TX and RX
DNN algorithm		Device and technology	
Data precision	Weights and activation precision	Memory cell	RRAM or SRAM
Network structure	DNN network structure	Tech node	Technology node for fabrication
Sparsity	DNN layer-wise sparsity	Bits/cell	Number of levels in RRAM

**Table 1.**  
*Definition of the user inputs to SIAM.*

better understanding of the framework. Firstly, the partition and mapping engine is used to perform layer partition and mapping onto the chiplets and IMC crossbars, which generates the IMC architecture structure, the number of required chiplets and IMC tiles per layer, the IMC architecture utilization, the volume of intra-chiplet and inter-chiplet data movement, and the number of global accumulator accesses. Next, the circuit and NoC engine evaluate the intra-chiplet and global circuit performance, respectively, providing hardware performance metrics such as area, energy, and latency. Meanwhile, the NoP engine evaluates the cost of chiplet-to-chiplet data movement. Lastly, the DRAM engine assesses the memory access cost, providing energy and latency performance metrics. Except for the partition and mapping engine, all engines operate concurrently, resulting in shorter simulation times. Additionally, SIAM can be used to benchmark traditional monolithic IMC architectures. In the following sections, we provide detailed information on the four engines that comprise SIAM’s core functionality.

3.3 Partition and mapping engine

The operational steps of the partition and mapping engine are explained in Algorithm 1. The engine is responsible for partitioning the DNN layers and mapping them onto the IMC chiplets and crossbar arrays. The partitioning and mapping are carried out layer by layer for the entire DNN. The engine takes various user inputs such as

DNN structure, the precision of DNN weights, the mapping scheme for IMC chiplets, the size of IMC chiplets, and the size of IMC crossbars.

To begin with, we will describe the IMC mapping approach that is employed in SIAM. Suppose we have a layer  $i$  with weight matrix  $W_i$  represented by  $Kx_i \times Ky_i \times Nif_i \times Nof_i$ , where  $Kx$  and  $Ky$  indicate the kernel size,  $Nif$  refers to the number of input features, and  $Nof$  denotes the number of output features. SIAM uses the same mapping scheme presented in [44, 45]:

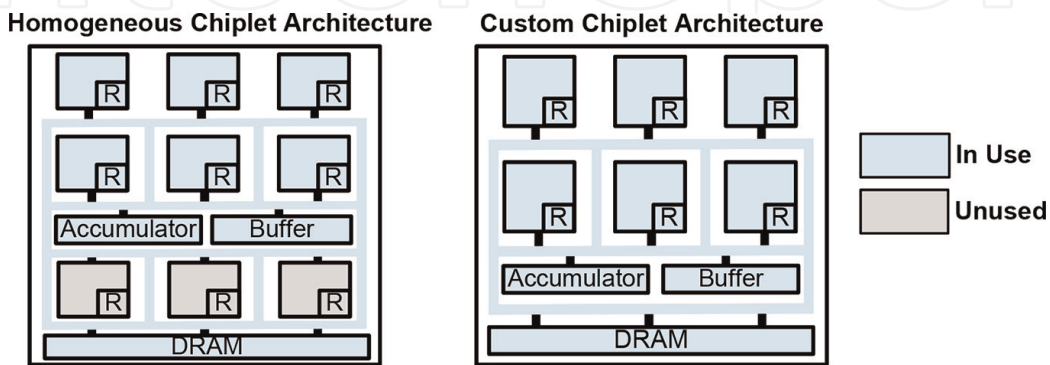
$$N_i^r = \left\lceil \frac{Kx_i \times Ky_i \times Nif_i}{(PE_x)} \right\rceil; N_i^c = \left\lceil \frac{Nof_i \times N_{bits}}{(PE_y)} \right\rceil \quad (1)$$

The equation given above calculates the required number of rows and columns for the IMC crossbars to map layer  $i$  of the DNN.  $N_i^r$  and  $N_i^c$  represent the number of rows and columns required, while  $N_{bits}$ ,  $PE_x$ , and  $PE_y$  denote the DNN weight precision and the number of rows and columns in the IMC crossbar array. To find the total number of required IMC crossbar arrays to map layer  $i$  of the DNN, one can multiply  $N_i^r$  and  $N_i^c$  together, which gives  $N_i^{Total}$  (line 7 of Algorithm 1).

SIAM can create homogeneous and custom chiplet-based IMC architectures using two types of chiplet partitions. The partition and mapping engine generates architectures based on the assumption that each DNN layer cannot be divided across multiple chiplets and that each chiplet can support multiple layers for optimal chiplet utilization. To map an entire layer of the DNN, multiple chiplets with IMC crossbar arrays are required due to numerous multi-bit weights in each layer. Dividing a layer across multiple chiplets increases overhead in control logic required for routing inputs, higher chiplet-to-chiplet communication energy and latency, and greater inter-chiplet data communication volume. The engine uniformly divides the layer across multiple chiplets during partitioning to prevent workload imbalance issues. The engine determines the number of chiplets required to map layer  $i$  of the DNN based on the total number of required IMC crossbar arrays,  $N_i^{Total}$ , using the equation

$N_i^{Chiplet} = \left\lceil \frac{N_i^{Total}}{S} \right\rceil$ , where  $S$  represents the total number of IMC crossbar arrays within a chiplet (the chiplet size). The resulting architectures generated by the partition and mapping engine can be seen in **Figure 9**.

After determining the number of required chiplets to map a layer of the DNN, the next step is to determine the total number of chiplets in the architecture. This is done



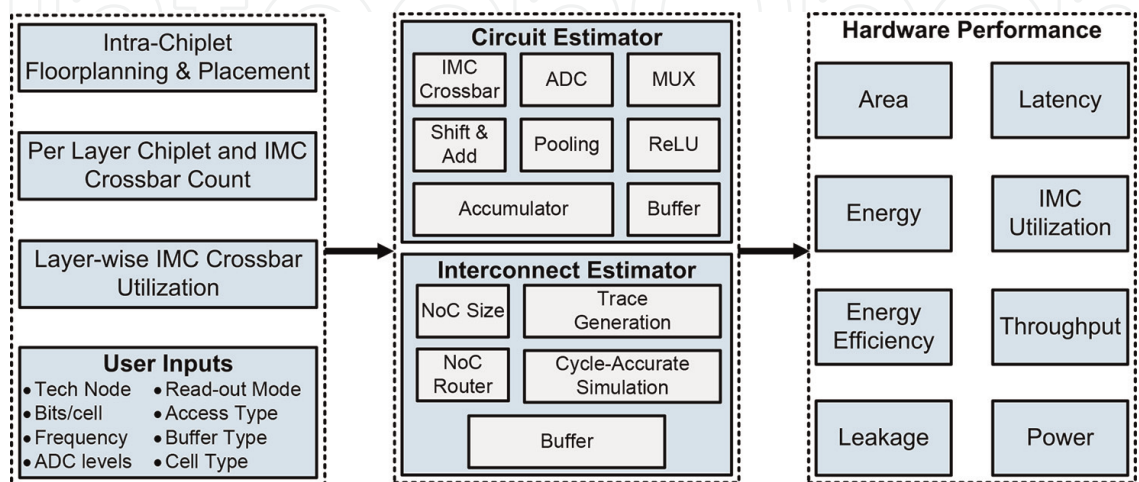
**Figure 9.** The figure illustrates two chiplet-based IMC architectures, namely homogeneous (left) and custom (right), generated by SIAM for the same DNN. The homogeneous architecture is a generic architecture, whereas the custom architecture is tailored for the specific DNN. The NoP router is denoted by  $R$  in both architectures.

at line 9 of Algorithm 1. In the *homogeneous chiplet partition* scheme, the user inputs a fixed number of chiplets to map the DNN. The engine compares the total number of chiplets required to map the DNN ( $N^{Chiplet}$ ) with the maximum available chiplets in the architecture ( $C$ ). If the number of required chiplets exceeds the maximum available chiplets, the engine throws an error and requests an increase in the number of available chiplets. On the other hand, if the number of required chiplets is less than or equal to the maximum available chiplets, the engine continues with the partition and mapping process for the subsequent layers in the DNN.

The custom partition scheme in SIAM creates a chiplet-based IMC architecture tailored to the specific DNN being considered without any upper limit on the number of chiplets used. Each chiplet in this scheme has a consistent structure with a fixed number of IMC tiles containing IMC crossbar arrays and peripheral circuitry. On the other hand, the homogeneous partition scheme uses a fixed number of chiplets (user input) to map the DNN in a generic manner. SIAM allows for the comparison of both architectures on a single platform. After partitioning and mapping the layers onto the IMC chiplets, the engine calculates the total amount of data communicated within and across the chiplets, taking into account instances where a layer is partitioned across multiple chiplets. The global accumulator is used to generate the layer output in such cases. The engine also determines the number of additions performed by the global accumulator and the number of global buffer accesses required. The engine output includes the layer partition across chiplets, the necessary number of chiplets and IMC crossbars, IMC crossbar utilization, intra- and inter-chiplet data movement volume, and the number of the global accumulator and buffer accesses. This information is then used by the circuit, NoC, and NoP engines to evaluate the performance of the chiplet-based IMC architecture.

### 3.4 Circuit and NoC engine

After the partitioning and mapping of the DNN, the next step in SIAM is to arrange the inter- and intra-chiplet components in a floorplan and place them. This process leads to the final design of the chiplet-based IMC architecture. Once the architecture is determined, the circuit and NoC engine estimate the performance of the hardware, as illustrated in **Figure 10**. The engine uses a model-based estimator to



**Figure 10.** Block diagram of the circuit and NoC engine within SIAM. The engine utilizes a separate circuit and NoC simulators that perform the overall hardware performance estimation.



evaluate the circuit aspect, while it employs a trace-based estimator for the interconnect portion.

#### 3.4.1 Circuit estimator

The circuit estimator evaluates the hardware performance of the chiplets, global accumulator, and global buffer in the entire chiplet-based IMC architecture. To perform this evaluation, the engine considers a range of inputs such as the placement of components within and across chiplets, the number of chiplets and IMC crossbars per layer, the IMC utilization per layer, the technology node, the operating frequency, the type of IMC cell, the number of bits per cell, and the ADC precision. The intra-chiplet circuits include the IMC crossbar array, buffer, accumulator, activation unit, and pooling unit. In contrast, the peripheral circuits consist of the ADC, multiplexer circuit, shift and add circuit, and decoders. The circuit estimator is calibrated using NeuroSim [95].

The circuit estimator evaluates the performance of the entire chiplet-based IMC architecture, where each layer of the DNN is considered separately, and each chiplet is responsible for computations for a subset of layers. The partition and mapping engine provides the chiplet count, IMC crossbar count, and IMC utilization values for each layer. The estimator estimates the area, energy, and latency from the device level to the circuit and architecture levels, using user inputs such as technology node, IMC cell type, IMC crossbar size, ADC precision, and read-out mode. For each IMC crossbar within the chiplet, the cost of a single crossbar and its peripheral circuits are evaluated to obtain the total area, energy, and latency of the IMC chiplet, which includes the buffer cost, shift and adder circuitry, accumulator, pooling, and activation units. The global accumulator and global buffer accumulate the partial sum of a layer across chiplets at the chiplet-level. The estimator utilizes the number of additions performed, data volume from each chiplet, and the accumulator size to determine the global accumulator and buffer's area, energy, and latency. Finally, the estimator repeats the estimation for all chiplets required for a given layer of the DNN to determine the overall hardware performance.

#### 3.4.2 NoC Estimator

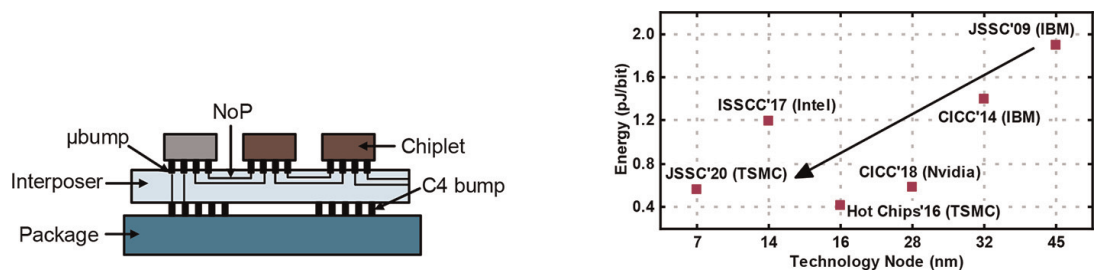
Effective communication is essential for achieving optimal hardware performance in DNN accelerators [63]. In [103], communication-intensive DNN accelerators are extensively discussed. Every layer of a DNN transmits a significant amount of data to other layers. Studies have demonstrated that communication can account for up to 90% of the total inference latency in DNNs [45]. Therefore, it is vital to develop an efficient communication protocol for DNNs. To achieve this goal, we incorporate the cost of communication between multiple layers within a chiplet. As NoC is the standard interconnect fabric used in the SoC-domain [104], we consider using an NoC for intra-chiplet communication. We customize a cycle-accurate NoC simulator, BookSim [98], to evaluate NoC performance. First, we generate a trace file for each chiplet following Algorithm 2. The algorithm considers the number of tiles, input activations, chiplets, layer-to-chiplet mapping, quantization bit-precision, and bus width. We then determine the source and destination tile information for each layer in each chiplet. Next, we calculate the number of packets for each source-destination pair and generate a trace file as a tuple consisting of the source tile ID, destination tile ID, and

timestamp. Finally, we simulate the trace file using BookSim to obtain the area, energy, and latency for on-chip communication within each chiplet.

### 3.5 NoP engine

Using specialized signaling techniques and driver circuits, the NoP handles on-chip data movement between different chiplets. This is achieved through a silicon interposer or organic substrate, as shown in [105, 106]. **Figure 11** (left) shows the cross-sectional image of a 2.5D integration with chiplets and an interposer. However, modeling the NoP's performance can be challenging due to its complex interconnect structure, specialized driver architectures, and corresponding signaling techniques. To accurately estimate performance, the NoP engine models each component of the NoP. **Figure 11** (right) shows different NoP implementations with the corresponding energy-per-bit ( $E_{bit}$ ) proposed in prior works. The NoP performance evaluation comprises two main components: (1) NoP latency estimation and (2) NoP area and power estimation. To estimate NoP latency, the engine uses a cycle-accurate simulator to evaluate the interconnect. It generates the NoP trace using Algorithm 2, similar to the one used for NoCs, based on the chiplet-to-chiplet data volume generated by the partition and mapping engine. The generated traces are simulated using a cycle-accurate simulator or the NoP estimator to determine the latency of the NoP interconnect. To estimate NoP area and power consumption, the engine obtains interconnect parameters such as wire length, pitch, width, and stack-up. The engine then determines the interconnect capacitance and resistance using the PTM interconnect models [107]. Based on these parameters, it generates timing parameters for the interconnect and compares them to the target bandwidth. If the timing parameters do not meet the bandwidth requirements, the NoP engine chooses the maximum allowable bandwidth.

The NoP engine evaluates the NoP transmitter/receiver (TX/RX) circuits, which includes clocking circuitry, based on the energy per bit ( $E_{bit}$ ), number of TX/RX channels, bandwidth, chiplet-to-chiplet data volume, and operating frequency to estimate the energy and latency cost of the TX/RX circuits. The energy calculation for the NoP driver is provided in Algorithm 3. The NoP engine first calculates the number of bits being transferred between chiplets. It then retrieves the energy per bit ( $E_{bit}$ ) from previous research, which is illustrated in **Figure 11** (right). The total energy for the TX/RX channel is computed by multiplying the number of bits and the energy per bit, as indicated in line 9 of Algorithm 3. Afterward, the NoP engine determines the NoP driver area using the NoP driver area cost from previous implementations



**Figure 11.**  
(Left) Cross-sectional image of the NoP interconnect. The NoP is routed within the interposer connecting different chiplets across the architecture. M bumps connect the chiplets to the interposer, (Right) Energy per bit for different NoP driver circuit and signaling techniques proposed in prior works.

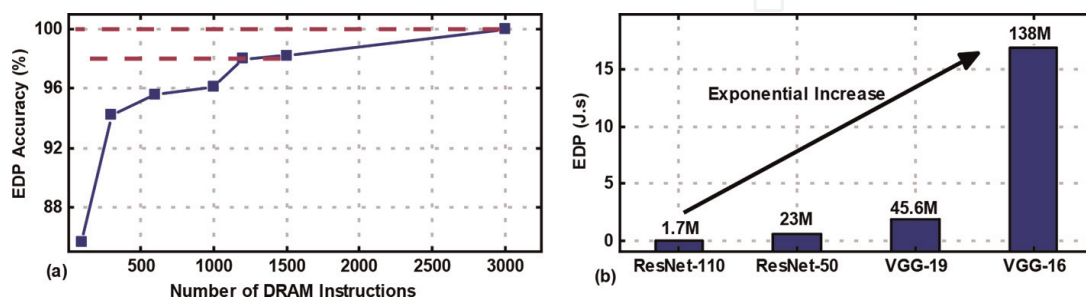
(**Figure 11**). Finally, the engine integrates the interconnect and driver performance metrics to determine the overall NoP performance.

The functional flow of the NoP engine is summarized below:

- NoP trace generation based on the chiplet placement, inter-chiplet data transfer volume, and inter-chiplet layer partition,
- NoP interconnect evaluation using a cycle-accurate simulator to generate energy, latency, and area metrics,
- NoP router modeling and TX/RX driver based on real measurements. Finally, the NoP engine combines the NoP driver and the interconnect metrics to generate the overall NoP performance.

### 3.6 DRAM engine

The architecture of the IMC based on chiplets includes a DRAM chiplet that serves as the external memory for the IMC chiplets. The DRAM engine estimates the external memory access required for this architecture. The assumption is that the DRAM will transfer the entire set of weights to the chiplet only once before performing the inference task. This assumption remains constant across different architectural configurations and inference runs for a specific DNN model. The DRAM engine includes a DRAM request generator, RAMULATOR [108] for estimating the latency for DRAM transactions, and VAMPIRE [109] to estimate the DRAM transaction power. The DRAM choice depends on user input and currently supports DDR3 and DDR4 [110, 111]. For a given DNN model, the DRAM engine generates the required traces and memory requests with timestamps, which include the location within the DRAM memory and the operation. SIAM uses a customized version of the cycle-accurate simulator RAMULATOR and the model-based power analysis tool VAMPIRE. To reduce simulation time, the DRAM engine estimates smaller sets of instructions, which are then multiplied by the total number of sets required to represent all the weights in the DNN. An experiment was performed to calibrate the method (**Figure 12(a)** and **(b)**), which showed that a reduction of 50% of DRAM instructions to the engine results in less than 2% EDP accuracy degradation than that at 100% instructions. The overall EDP for different networks across different datasets for



**Figure 12.**

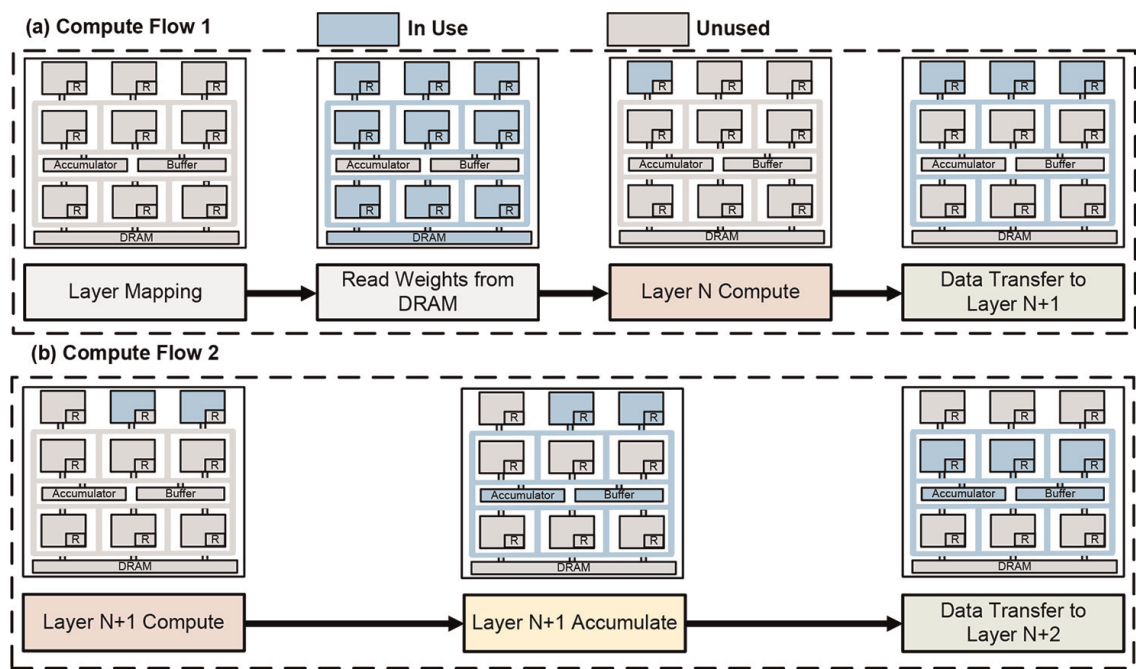
(a) The accuracy of EDP prediction for different numbers of instructions processed to represent 3000 DRAM instructions. Reduction in the number of instructions to half results in less than 2% EDP accuracy degradation for half the simulation time. (b) EDP of DRAM transactions (DDR4) for different DNNs. There is an exponential increase in DRAM cost with an increase in DNN model size.

DDR4 shows an exponential increase in EDP with the increase in the model size of the DNN. The DRAM engine provides a fast and accurate estimation of external memory access for the entire range of DNNs through this method. To summarize, the DRAM engine’s execution involves the following essential steps:

- Generate DRAM requests based on the data precision and DNN model size,
- Calculate the latency cost of DRAM transactions using a customized version of RAMULATOR, and calculate the power consumption by utilizing a customized version of VAMPIRE,
- Combine the results to produce the total cost of DRAM access.

#### 4. SIAM dataflow

This section outlines the default dataflow of the SIAM chiplet-based IMC architecture. The computation dataflow is illustrated in **Figure 13**. Before starting the inference task, the weights are obtained from the DRAM and allocated to the IMC chiplets according to the partition and mapping engine described in Section III-C. There are two possible scenarios based on this: either no layer is spread across several chiplets, or a layer is distributed across multiple chiplets. Let us consider the case where layer N of the DNN is assigned to the first chiplet, as depicted in **Figure 13(a)**. During computation, the entire layer is processed within one chiplet, generating the computed output activations for layer N. The global accumulator and buffer are not utilized during this process and are turned off. When the computation is done, the output activations are sent to the chiplets that execute layer N + 1. If two chiplets are required to map the weights for layer N + 1, the NoP transfers the output activation from layer



**Figure 13.** Computation dataflow within the chiplet-based IMC architecture in SIAM. Two cases arise: (a) no layer is partitioned across two or more chiplets and (b) a layer is partitioned across two or more chiplets.



N to both chiplets, as shown in **Figure 13(a)**. **Figure 13(b)** presents the computation flow for layer  $N + 1$ , where both chiplets execute computations in parallel. The mapping ensures that an equal number of weights are assigned to each chiplet, avoiding any workload imbalances. After the computation is completed, the generated partial sums are aggregated using the global accumulator and buffer. Then, the accumulated outputs from layer  $N + 1$  are transferred to the chiplets that hold the weights for layer  $N + 2$ . This process is repeated until all layers are processed and the final output is generated. The algorithmic implementation of the dataflow used in the SIAM IMC chiplet architecture is explained in Algorithm 4.

## 5. Conclusion

In this chapter, we discussed benchmarking chiplet-based IMC-based AI accelerators. We discuss various IMC architectures proposed in the literature. CMOS- (e.g., SRAM) and memristor- (e.g., RRAM) based IMC architectures are discussed. Although IMC improves the energy efficiency of computing elements, it increases on-chip communication volume. To address this, we discuss chiplet-based in-memory architectures. We also discuss different benchmarking simulators for monolithic and chiplet-based IMC architectures in detail. Finally, we dive deeply into SIAM, a chiplet-based IMC benchmarking simulator. SIAM provides a unified framework for performance benchmarking of chiplet-based IMC architectures. SIAM supports both *homogeneous* (*generic*) and custom chiplet-based IMC architectures. Finally, SIAM interfaces with popular deep learning frameworks such as PyTorch and TensorFlow and can be integrated with modern NAS techniques.

## Author details

Gokul Krishnan<sup>1</sup>, Sumit K. Mandal<sup>2</sup>, A. Alper Goksoy<sup>3</sup>, Zhenyu Wang<sup>1</sup>, Chaitali Chakrabarti<sup>1</sup>, Jae-sun Seo<sup>1</sup>, Umit Y. Ogras<sup>3</sup> and Yu Cao<sup>1\*</sup>


<sup>1</sup> School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, USA

<sup>2</sup> Indian Institute of Science, Bangalore, India

<sup>3</sup> University of Wisconsin-Madison, Madison, USA

\*Address all correspondence to: yu.cao@asu.edu

## IntechOpen

© 2023 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. 

## References

- [1] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*. 2012;**1**:1097-1105
- [2] Deng L, Hinton G, Kingsbury B. New types of deep neural network learning for speech recognition and related applications: An overview. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. Vancouver, Canada: IEEE; 2013. pp. 8599-8603
- [3] Litjens G, Kooi T, Bejnordi BE, Setio AAA, Ciompi F, Ghafoorian M, et al. A survey on deep learning in medical image analysis. *Medical Image Analysis*. 2017;**42**:60-88
- [4] Lin T-Y, Maire M, Belongie S, Hays J, Perona P, Ramanan D, et al. Microsoft coco: Common objects in context. In: *European Conference on Computer Vision*. Springer; 2014. pp. 740-755
- [5] Hamilton W, Ying Z, Leskovec J. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*. 2017; **30**:1025-1035
- [6] Liu B, Chen Y, Liu S, Kim H-S. Deep learning in latent space for video prediction and compression. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021:701-710
- [7] Rubinstein R, Bruckstein AM, Elad M. Dictionaries for sparse representation modeling. *Proceedings of the IEEE*. 2010;**98**(6):1045-1057
- [8] Gagniuc PA. Markov Chains: From Theory to Implementation and Experimentation. John Wiley & Sons; 2017
- [9] Kotsiantis SB. Decision trees: A recent overview. *Artificial Intelligence Review*. John Wiley & Sons Publisher; 2013; **39**(4):261-283
- [10] Pisner DA, Schnyer DM. Support vector machine. In: *Machine Learning*. Elsevier; 2020. pp. 101-121
- [11] Goodfellow I, Bengio Y, Courville A. *Deep Learning*. MIT Press; 2016
- [12] Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv Preprint arXiv:1704.04861*. 2017
- [13] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, et al. Going deeper with convolutions. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Boston. 2015. pp. 1-9
- [14] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Las Vegas. 2016. pp. 770-778
- [15] Huang G, Liu Z, Van Der Maaten L, Weinberger KQ. Densely connected convolutional networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Hawaii. 2017. pp. 4700-4708
- [16] Iandola FN, Han S, Moskewicz MW, Ashraf K, Dally WJ, Keutzer K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 0.5 mb model

size. arXiv Preprint arXiv:1602.07360. 2016

[17] Krishnan G, Ma Y, Cao Y. Small-world-based structural pruning for efficient FPGA inference of deep neural networks. In: 2020 IEEE 15th International Conference on Solid-State & Integrated Circuit Technology (ICSICT). IEEE; 2020. pp. 1-5

[18] Krishnan G, Du X, Cao Y. Structural pruning in deep neural networks: A small-world approach. arXiv Preprint arXiv:1911.04453. 2019

[19] Du X, Krishnan G, Mohanty A, Li Z, Charan G, Cao Y. Towards efficient neural networks on-a-chip: Joint hardware-algorithm approaches. In: 2019 China Semiconductor Technology International Conference (CSTIC). Shanghai, China: IEEE; 2019. pp. 1-5

[20] Zoph B, Vasudevan V, Shlens J, Le QV. Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, Utah. 2018. pp. 8697-8710

[21] Wu B, Dai X, Zhang P, Wang Y, Sun F, Wu Y, et al. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, California. 2019. pp. 10734-10742

[22] Real E, Aggarwal A, Huang Y, Le QV. Regularized evolution for image classifier architecture search. Proceedings of the AAAI Conference on Artificial Intelligence. 2019;33(01): 4780-4789

[23] Liu C, Zoph B, Neumann M, Shlens J, Hua W, Li L-J, et al. Progressive

neural architecture search. In: Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany. 2018. pp. 19-34

[24] Zhou D, Zhou X, Zhang W, Loy CC, Yi S, Zhang X, et al. Ecnas: Finding proxies for economical neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020. pp. 11396-11404

[25] Tan M, Chen B, Pang R, Vasudevan V, Sandler M, Howard A, et al. Mnasnet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, California. 2019. pp. 2820-2828

[26] Jordan MI. Serial order: A parallel distributed processing approach. *Advances in Psychology*. 1997;121: 471-495

[27] Sutskever I, Vinyals O, Le QV. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*. 2014;2: 3104-3112

[28] Lipton ZC, Berkowitz J, Elkan C. A critical review of recurrent neural networks for sequence learning. arXiv Preprint arXiv:1506.00019. 2015

[29] Hochreiter S, Schmidhuber J. Long short-term memory. *Neural Computation*. 1997;9(8):1735-1780

[30] Schuster M, Paliwal KK. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*. 1997;45(11):2673-2681

[31] M. Gori, G. Monfardini, and F. Scarselli, A new model for learning in

- graph domains. In: Proceedings 2005 IEEE International Joint Conference on Neural Networks. Vol. 2. Montreal, Canada: IEEE; 2005. pp. 729–734
- [32] Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G. The graph neural network model. *IEEE Transactions on Neural Networks*. 2008; **20**(1):61-80
- [33] Gallicchio C, Micheli A. Graph echo state networks. In: The 2010 International Joint Conference on Neural Networks (IJCNN). Barcelona, Spain: IEEE; 2010. pp. 1-8
- [34] Liu Z, Chen C, Li L, Zhou J, Li X, Song L, et al. Geniepath: Graph neural networks with adaptive receptive paths. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2019;**33**(01): 4424-4431
- [35] Xu K, Hu W, Leskovec J, Jegelka S. How powerful are graph neural networks? *arXiv Preprint arXiv: 1810.00826*. 2018
- [36] Chiang W-L, Liu X, Si S, Li Y, Bengio S, Hsieh C-J. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Anchorage, Alaska. 2019. pp. 257-266
- [37] Simonovsky M, Komodakis N. Graphvae: Towards generation of small graphs using variational autoencoders. In: *International Conference on Artificial Neural Networks*. Springer; 2018. pp. 412-422
- [38] Ma T, Chen J, Xiao C. Constrained generation of semantically valid graphs via regularizing variational autoencoders. *arXiv Preprint arXiv: 1809.02630*. 2018
- [39] De Cao N, Kipf T. Molgan: An implicit generative model for small molecular graphs. *arXiv Preprint arXiv: 1805.11973*. 2018
- [40] Horowitz M. Computing's energy problem (and what we can do about it). *IEEE ISSCC*. 2014:10-14
- [41] Gholami A, Kim S, Dong Z, Yao Z, Mahoney MW, Keutzer K. A survey of quantization methods for efficient neural network inference. *arXiv Preprint arXiv:2103.13630*. 2021
- [42] Krishnan G, Goksoy AA, Mandal SK, Wang Z, Chakrabarti C, Seo J-s, et al. Big-little chiplets for in-memory acceleration of DNNS: A scalable heterogeneous architecture. In: *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, San Diego, California. 2022. pp. 1-9
- [43] Wang Z, Nair GR, Krishnan G, Mandal SK, Cherian N, Seo J-s, et al. AI computing in light of 2.5 d interconnect roadmap: Big-little chiplets for in-memory acceleration. In: *2022 International Electron Devices Meeting (IEDM)*. San Francisco, California: IEEE; 2022. pp. 23-26
- [44] Shafiee A et al. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News*. 2016; **44**(3):14-26
- [45] Krishnan G, Mandal SK, Chakrabarti C, Seo J-s, Ogras UY, Cao Y. Interconnect-aware area and energy optimization for in-memory acceleration of DNNS. *IEEE Design & Test*. 2020; **37**(6):79-87
- [46] Krishnan G, Mandal SK, Pannala M, Chakrabarti C, Seo J-s, Ogras UY, et al.



SIAM: Chiplet-based scalable in-memory acceleration with mesh for deep neural networks. *ACM Transactions on Embedded Computing Systems (TECS)*. 2021;**20**(5s):1-24

[47] Song L, Qian X, Li H, Chen Y. Pipelayer: A pipelined ReRAM-based accelerator for deep learning. In: 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), Austin, Texas. 2017. pp. 541-552

[48] Valavi H, Ramadge PJ, Nestler E, Verma N. A 64-tile 2.4-mb in-memory-computing CNN accelerator employing charge-domain compute. *IEEE Journal of Solid-State Circuits*. 2019;**54**(6): 1789-1799

[49] Yin S, Zhang B, Kim M, Saikia J, Kwon S, Myung S, et al. Pimca: A 3.4-MB programmable in-memory computing accelerator in 28 nm for on-chip DNN inference. In: 2021 Symposium on VLSI Technology. Kyoto, Japan: IEEE; 2021. pp. 1-2

[50] Yin S, Jiang Z, Kim M, Gupta T, Seok M, Seo J-s. Vesti: Energy-efficient in-memory computing accelerator for deep neural networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*. 2019;**28**(1):48-61

[51] Yin S, Jiang Z, Seo J-s, Seok M. XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks. *IEEE Journal of Solid-State Circuits*. 2020;**55**(6):1733-1743

[52] Jiang Z, Yin S, Seo J-s, Seok M. C3SRAM: An in-memory-computing SRAM macro based on robust capacitive coupling computing mechanism. *IEEE Journal of Solid-State Circuits*. 2020; **55**(7):1888-1897

[53] Chih Y-D, Lee P-H, Fujiwara H, Shih Y-C, Lee C-F, Naous R, et al. An 89tops/

w and 16.3 tops/mm<sup>2</sup> all-digital SRAM-based full-precision compute-in memory macro in 22nm for machine-learning edge applications. In: 2021 IEEE International Solid-State Circuits Conference (ISSCC). Vol. 64. San Francisco, California: IEEE; 2021. pp. 252-254

[54] Kim H, Yoo T, Kim TT-H, Kim B. Colonnade: A reconfigurable SRAM-based digital bit-serial compute-in-memory macro for processing neural networks. *IEEE Journal of Solid-State Circuits*. 2021;**56**(7):2221-2233

[55] Yue J, Liu Y, Yuan Z, Feng X, He Y, Sun W, et al. Sticker-IM: A 65 nm computing-in-memory NN processor using block-wise sparsity optimization and inter/intra-macro data reuse. *IEEE Journal of Solid-State Circuits*. 2022;**57**(8):2560-2573

[56] Fujiwara H, Mori H, Zhao W-C, Chuang M-C, Naous R, Chuang C-K, et al. A 5-nm 254-tops/w 221-tops/mm<sup>2</sup> fully-digital computing-in-memory macro supporting wide-range dynamic-voltage-frequency scaling and simultaneous mac and write operations. In: 2022 IEEE International Solid-State Circuits Conference (ISSCC). Vol. 65. San Francisco, California: IEEE; 2022. pp. 1-3

[57] Spetalnick SD, Chang M, Crafton B, Khwa W-S, Chih Y-D, Chang M-F, et al. A 40nm 64kb 26.56 tops/w 2.37 mb/mm<sup>2</sup> rram binary/compute-in-memory macro with 4.23 x improvement in density and 75% use of sensing dynamic range. In: 2022 IEEE International Solid-State Circuits Conference (ISSCC). Vol. 65. San Francisco, California: IEEE; 2022. pp. 1-3

[58] Mao M et al. MAX2: An ReRAM-based neural network accelerator that maximizes data reuse and area

utilization. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*. 2019;9(2):398-410

[59] Liehr M, Hazra J, Beckmann K, Rafiq S, Cady N. Impact of switching variability of 65nm CMOS integrated hafnium dioxide-based ReRAM devices on distinct level operations. In: *IIRW*. IEEE; 2020. pp. 1-4

[60] Krishnan G, Sun J, Hazra J, Du X, Liehr M, Li Z, et al. Robust RRAM-based in-memory computing in light of model stability. In: *IRPS*. IEEE; 2021. pp. 1-5

[61] Krishnan G, Yang L, Sun J, Hazra J, Du X, Liehr M, et al. Exploring model stability of deep neural networks for reliable RRAM-based in-memory acceleration. *IEEE Transactions on Computers*. 2022;71(11):2740-2752

[62] He W, Yin S, Kim Y, Sun X, Kim J-J, Yu S, et al. 2-bit-per-cell RRAM-based in-memory computing for area-/energy-efficient deep learning. *IEEE Solid-State Circuits Letters*. 2020;3:194-197

[63] Mandal SK, Krishnan G, Chakrabarti C, Seo J-s, Cao Y, Ogras UY. A latency-optimized reconfigurable NOC for in-memory acceleration of DNNs. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*. 2020;10(3):362-375

[64] Krishnan G, Wang Z, Yang L, Yeo I, Meng J, Joshi RV, et al. IMC architecture for robust DNN acceleration. In: *2022 IEEE 16th International Conference on Solid-State & Integrated Circuit Technology (ICSICT)*. IEEE; 2022. pp. 1-4

[65] Krishnan G, Wang Z, Yeo I, Yang L, Meng J, Liehr M, et al. Hybrid RRAM/ SRAM in-memory computing for robust DNN acceleration. *IEEE Transactions on*

*Computer-Aided Design of Integrated Circuits and Systems*. 2022;41(11):4241-4252

[66] Moitra A, Bhattacharjee A, Kuang R, Krishnan G, Cao Y, Panda P. Spikesim: An end-to-end compute-in-memory hardware evaluation tool for benchmarking spiking neural networks. *arXiv Preprint arXiv:2210.12899*. 2022

[67] Krishnan G. Energy-Efficient In-Memory Acceleration of Deep Neural Networks Through a Hardware-Software Co-Design Approach [Technical Report]. Arizona State University; 2022

[68] Si X, Chen J-J, Tu Y-N, Huang W-H, Wang J-H, Chiu Y-C, et al. 24.5 a twin-8t SRAM computation-in-memory macro for multiple-bit CNN-based machine learning. In: *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. San Francisco, California: IEEE; 2019. pp. 396-398

[69] Dong Q, Sinangil ME, Erbagci B, Sun D, Khwa W-S, Liao H-J, et al. 15.3 a 351tops/w and 372.4 GOPS compute-in-memory SRAM macro in 7 nm finfet CMOS for machine-learning applications. In: *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. San Francisco, California: IEEE; 2020. pp. 242-244

[70] Su J-W, Si X, Chou Y-C, Chang T-W, Huang W-H, Tu Y-N, et al. 15.2 a 28nm 64kb inference-training two-way transpose multibit 6t sram compute-in-memory macro for ai edge chips. In: *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. San Francisco, California: IEEE; 2020. pp. 240-242

[71] Seo JS, Saikia J, Meng J, He W, Suh HS, Liao Y, et al. Digital Versus Analog Artificial Intelligence Accelerators:

Advances, trends, and emerging designs. IEEE Solid-State Circuits Magazine. 2022;**14**(3):65-79

[72] Kang M, Kim Y, Patil AD, Shanbhag NR. Deep in-memory architectures for machine learning—accuracy versus efficiency trade-offs. IEEE Transactions on Circuits and Systems I: Regular Papers. 2020;**67**(5):1627-1639

[73] Zhang J, Wang Z, Verma N. In-memory computation of a machine-learning classifier in a standard 6t sram array. IEEE Journal of Solid-State Circuits. 2017;**52**(4):915-924

[74] Krishnan G, Hazra J, Liehr M, Du X, Beckmann K, Joshi RV, et al. Design limits of in-memory computing: Beyond the crossbar. In: 2021 5th IEEE Electron Devices Technology & Manufacturing Conference (EDTM). Chengdu, China: IEEE; 2021. pp. 1-3

[75] Cherupally SK, Meng J, Rakin AS, Yin S, Yeo I, Yu S, et al. Improving the accuracy and robustness of rram-based in-memory computing against rram hardware noise and adversarial attacks. Semiconductor Science and Technology. 2022;**37**(3):034001

[76] Long Y, She X, Mukhopadhyay S. Design of reliable DNN accelerator with un-reliable ReRAM. In: DATE. Grenoble France: IEEE; 2019. pp. 1769-1774

[77] Ma C et al. Go unary: A novel synapse coding and mapping scheme for reliable Reram-based neuromorphic computing. In: DATE. Grenoble France: IEEE; 2020. pp.1432-1437

[78] Chakraborty I, Ali MF, Kim DE, Ankit A, Roy K. Geniex: A generalized approach to emulating non-ideality in memristive Xbars using neural networks. In: 2020 57th ACM/IEEE Design

Automation Conference (DAC), San Francisco, California. 2020. pp. 1-6

[79] Charan G et al. Accurate inference with inaccurate RRAM devices: Statistical data, model transfer, and on-line adaptation. In: DAC. San Francisco, California: IEEE; 2020. pp. 1-6

[80] Sun Y et al. Unary coding and variation-aware optimal mapping scheme for reliable ReRAM-based neuromorphic computing. TCAD. 2021;**40**(12):2495-2507

[81] Zhou C, Kadambi P, Mattina M, Whatmough PN. Noisy machines: Understanding noisy neural networks and enhancing robustness to analog hardware errors using distillation. arXiv Preprint arXiv:2001.04974. 2020

[82] Yang X et al. Multi-objective optimization of ReRAM crossbars for robust DNN inferencing under stochastic noise. In: ICCAD. IEEE/ACM; 2021. pp. 1-9

[83] Joshi V et al. Accurate deep neural network inference using computational phase-change memory. Nature Communications. 2020;**11**(1):2473

[84] Charan G, Mohanty A, Du X, Krishnan G, Joshi RV, Cao Y. Accurate inference with inaccurate RRAM devices: A joint algorithm-design solution. IEEE Journal on Exploratory Solid-State Computational Devices and Circuits. 2020;**6**(1):27-35

[85] Hu M, Li H, Chen Y, Wu Q, Rose GS. BSB training scheme implementation on memristor-based circuit. In: IEEE CISDA. Singapore: IEEE; 2013. pp. 80-87

[86] Liu B et al. Reduction and IR-drop compensations techniques for reliable



- neuromorphic computing systems. In: ICCAD. San Jose, CA: IEEE; 2014. pp. 63-70
- [87] Chen L et al. Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar. In: DATE. Lausanne, Switzerland: IEEE; 2017. pp. 19-24
- [88] Mohanty A et al. Random sparse adaptation for accurate inference with inaccurate multi-level RRAM arrays. In: IEDM. San Francisco: IEEE; 2017. pp. 3-6
- [89] Saikia J, Yin S, Cherupally SK, Zhang B, Meng J, Seok M, et al. Modeling and optimization of sram-based in-memory computing hardware design. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE; 2021. pp. 942-947
- [90] Bharadwaj S, Yin J, Beckmann B, Krishna T. Kite: A family of heterogeneous interposer topologies enabled via accurate interconnect modeling. In: 2020 57th ACM/IEEE Design Automation Conference (DAC). San Francisco, California: IEEE; 2020. pp. 1-6
- [91] Vivet P, Guthmuller E, Thonnart Y, Pillonnet G, Fuguet C, Miro-Panades I, et al. IntAct: A 96-core processor with six chiplets 3D-stacked on an active interposer with distributed interconnects and integrated power management. IEEE Journal of Solid-State Circuits. 2020;56(1):79-97
- [92] Shao YS, Clemons J, Venkatesan R, Zimmer B, Fojtik M, Jiang N, et al. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, Columbus, Ohio. 2019. pp. 14-27
- [93] Chen P-Y, Peng X, Yu S. Neurosim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2018; 37(12):3067-3080
- [94] Zhao W, Cao Y. New generation of predictive technology model for sub-45 nm early design exploration. IEEE Transactions on Electron Devices. 2006; 53(11):2816-2823
- [95] Peng X, Huang S, Luo Y, Sun X, Yu S. DNN+ NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies. In: 2019 IEEE International Electron Devices Meeting (IEDM), San Francisco, California. 2019. pp. 32-35
- [96] Krishnan G, Mandal SK, Chakrabarti C, Seo J-s, Ogras UY, Cao Y. Impact of on-chip interconnect on in-memory acceleration of deep neural networks. ACM Journal on Emerging Technologies in Computing Systems (JETC). 2021;18(2):1-22
- [97] Krishnan G, Mandal SK, Chakrabarti C, Seo J-s, Ogras UY, Cao Y. Interconnect-centric benchmarking of in-memory acceleration for DNNs. In: 2021 China Semiconductor Technology International Conference (CSTIC). Shanghai, China: IEEE; 2021. pp. 1-4
- [98] Jiang N et al. A detailed and flexible cycle-accurate network-on-chip simulator. In: 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). Austin, Texas: IEEE; 2013. pp. 86-96
- [99] Zhu Z, Sun H, Qiu K, Xia L, Krishnan G, Dai G, et al. MNSIM 2.0: A behavior-level modeling tool for memristor-based neuromorphic



computing systems. In: Proceedings of the 2020 on Great Lakes Symposium on VLSI, Beijing, China. 2020. pp. 83-88

[100] Jain S, Sengupta A, Roy K, Raghunathan A. RxNN: A framework for evaluating deep neural networks on resistive crossbars. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 2020;**40**(2):326-338

[101] Peng X, Huang S, Jiang H, Lu A, Yu S. DNN+ NeuroSim V2. 0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 2020;**40**(11): 2306-2319

[102] Jiang H, Huang S, Peng X, Su J-W, Chou Y-C, Huang W-H, et al. A two-way SRAM array based accelerator for deep neural network on-chip training. In: 2020 57th ACM/IEEE Design Automation Conference (DAC), San Francisco, California. 2020. pp. 1-6

[103] Nabavinejad SM, Baharloo M, Chen K-C, Palesi M, Kogel T, Ebrahimi M. An overview of efficient interconnection networks for deep neural network accelerators. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*. 2020;**10**(3):268-282

[104] Jeffers J et al. Intel Xeon Phi Processor High Performance Programming. Knights Landing Edition; 2016

[105] Poulton JW et al. A 0.54 pJ/b 20Gb/s ground-referenced single-ended short-haul serial link in 28nm CMOS for advanced packaging applications. In: 2013 IEEE ISSCC. San Francisco, California: IEEE; 2013. pp. 404-405

[106] Lin M-S et al. A 7-nm 4-GHz Arm<sup>1</sup>-core-based CoWoS<sup>1</sup> chiplet design

for high-performance computing. *IEEE Journal of Solid-State Circuits*. 2020; **55**(4):956-966

[107] Sinha S, Yeric G, Chandra V, Cline B, Cao Y. Exploring sub-20nm FinFET design with predictive technology models. In: DAC 2012. San Francisco, California: IEEE; 2012. pp. 283-288

[108] Kim Y, Yang W, Mutlu O. RAMULATOR: A fast and extensible DRAM simulator. *IEEE Computer Architecture Letters*. 2015;**15**(1):45-49

[109] Ghose S et al. What your DRAM power models are not telling you: Lessons from a detailed experimental study. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*. 2018; **2**(3):1-41

[110] MICRON, Datasheet for DDR3 model, 2011. Available at: [https://media-www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr3/2gb\\_ddr3l-rs.pdf?rev=f43686e89394458caff410138d9d2152](https://media-www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr3/2gb_ddr3l-rs.pdf?rev=f43686e89394458caff410138d9d2152) (Accessed March 29, 2021).

[111] MICRON, Datasheet for DDR4 model. 2014. Available at: [https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr4/4gb\\_ddr4\\_dram\\_2e0d.pdf](https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr4/4gb_ddr4_dram_2e0d.pdf) [Accessed March 29, 2021].