A Survey on Machine Learning Accelerators and Evolutionary Hardware Platforms

Sathwika Bavikadi and Abhijitt Dhavlle
Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA 22030 USA

Amlan Ganguly
Department of Computer Engineering, Rochester Institute of Technology, Rochester, NY 14623 USA

Anand Haridass
Intel Corporation, Bengaluru 560103, India

Hagar Hendy and Cory Merkel
Department of Computer Engineering, Rochester Institute of Technology, Rochester, NY 14623 USA

Vijay Janapa Reddi
John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 USA

Purab Ranjan Sutradhar
Department of Computer Engineering, Rochester Institute of Technology, Rochester, NY 14623 USA

Arun Joseph
IBM Electronic Design Automation, Bengaluru 560045, India

Sai Manoj Pudukotai Dinakarrao
Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA 22030 USA

Editor's notes:
Advanced computing systems have long been enablers for breakthroughs in artificial intelligence (AI) and machine learning (ML) algorithms, either through sheer computational power or form-factor miniaturization. However, as AI/ML algorithms become more complex and the size of data sets increases, existing computing platforms are no longer sufficient to bridge the gap between algorithmic innovation and hardware design. This article presents a survey about various ML accelerators.
— Partha Pratim Pande, Washington State University

Machine learning (ML) algorithms are finding their way into every single branch of scientific research lately. Deep neural networks (DNNs), particularly the convolutional neural networks (CNNs), are by far the most widely used genre of ML algorithms and are widely used in the fields of computer vision, pattern recognition, voice recognition, natural language processing [1]–[4], and cybersecurity [5]–[13]. In recent years, CNNs have achieved revolutionary performance in terms of classification and prediction, and providing intelligent solutions in terms of the smart applications that have surpassed human capability [14]–[17]. However, such a level of accuracy also comes with substantial computation workloads.

Digital Object Identifier 10.1109/MDAT.2022.3161126
Date of publication: 21 March 2022; date of current version: 22 April 2022.

Although efficient in terms of capabilities, current DNNs such as [1] and [18] require a large number of resources (in terms of computation power, memory, and bandwidth) to provide such high performance. For an enhanced ML inference, hardware implementations are preferred over software implementations [19]. Hardware implementations not only benefit in terms of inference of CNNs, but also can aid in better energy efficiency, area, and latency of the hardware platform [19]–[21]. It needs to be noted that the terms inference time and delay in this article are used interchangeably and refer to the inference time of an ML accelerator. However, the state-of-the-art processing hardware is suffering from performance bottleneck due to low-bandwidth and high-latency data communication with off-chip memory, which is popularly termed as the "memory wall" [22]. Numerous accelerator designs are proposed in the literature to address the existing challenges toward optimizing the overheads and enhancing the performance through network adaptation as well as efficient utilization of the underlying hardware resources [23].

In addition to traditional approaches, an increasing amount of research in alternative non-von Neumann computer architectures such as in-memory and near-memory computing is being carried out to overcome the existing challenges of traditional von Neumann architecture [23]. IMC or processing-in-memory (PIM) bridges the separation between memory and processor by implementing processing units inside the memory chip itself [24], [25]. Due to the fabrication limitations of the memory chips, these processing elements can achieve only limited functionalities [26]. However, they can leverage the bitline-level parallelism inside memory chip to construct a massively parallel single-instruction multidata (SIMD)-like processing architecture that also enjoys very high-bandwidth and low-latency data communication through the memory bitlines.

Similar to the efficient implementations of digital neural network (NN) architectures, biologically inspired neuromorphic computing architectures are as well introduced for efficient implementation of DNNs [27]–[32]. The neuromorphic architectures are capable of performing the multiply-and-accumulate (MAC) operations in an efficient manner through the design of crossbar memory array architectures through spiking NN designs.

Motivation and organization: ML and deep learning accelerators have become ubiquitous in a wide variety of applications. Traditional accelerators were deployed by realizing compute layers on the underlying platform ASIC or FPGA. Newer accelerators utilize approximate computing to reduce resources and power, while some other accelerators are deployed on neuromorphic hardware that is based on emerging memory technology. Thus, ML accelerators have diversified their deployment platform along with the underlying technology. Hence, to offer a comprehensive picture, we endeavor to present an overview of the research into ML-based accelerators under different platforms and constraints.

ML accelerator design trends based on hardware platform

In this work, we have categorized ML accelerators based on the platform they are deployed on and the methodology. The former is discussed in this section, while the latter in the next section. ML accelerators are designed according to the hardware platform on which it is deployed, as different hardware platforms have different constraints. The platforms considered are FPGA, ASIC, and in-memory architecture.

ML accelerators on FPGAs

FPGAs have received increased attention owing to their reconfigurability, flexibility, and speed. They are thus one of the best-suited platforms for the design of accelerators. ML applications are compute-intensive and complex and require low latency and high throughput. FPGAs satisfy such requirements, as they allow true parallelism compared to a CPU-only implementation. Nevertheless, ASIC offers some benefits over the FPGAs in terms of area, power, and speed. ASIC can be fine-tuned to optimize for the area; they consume less power comparatively and can achieve greater speeds than FPGAs. However, ASICs lack reconfigurability post-fabrication. The choice of FPGA or ASIC is application-dependent and the requirements posed by the user in terms of reconfigurability and constraints. We discuss some of the recent prominent works on FPGA-based ML accelerators below.

Qiu et al. [52] proposed an FPGA-based CNN accelerator to implement image classification on embedded platforms. This design focuses on implementing both the convolutional (CONV) and fully connected (FC) layers of the CNN, which are both computational and memory-centric operations. To implement these compute-intensive operations, the authors use singular value decomposition (SVD) to the weight matrix of the FC layer and reduce the memory footprint and the resource consumption by using a specific convolver design module. To further reduce the memory and bandwidth requirements, the hardware is designed to support dynamic precision data quantization.

Chakradhar et al. [53] proposed a dynamically configurable FPGA-based CNN accelerator. This article proposes a three-component system that consists of a coprocessor, a dynamically configurable processing core, and a 3-D bank memory system. The coprocessor in the system is designed to configure the hardware and the software automatically to exploit the parallelism. This dynamic programming is also used to perform pruning on an infeasible combination of data with low latency. Farabet et al. [54] proposed a scalable dataflow compiler called NeuFlow. This is designed as a goal to do realtime detection, recognition for large-scale computer vision systems. The compiler is implemented on the Xilinx Virtex 6 FPGA platform and is observed to produce speedup of up to 100× on real-world applications.

Unlike the previous discussed works, a framework named NEMOKD for hardware-aware evolution of knowledge-distilled student models is proposed by Stewart et al. [35]. The authors evaluate the accuracy, throughput, training time, and resource costs of two compression approaches deployed on Intel Movidius Myriad X VPU and a Xilinx Z7020 FPGA. An evolutionary algorithm to modify the structure of a 16-bit precision NN is used for the former platform, while a 1–8-bit precision quantization of fixed NNs for the latter platform. The main objective of the NEMOKD is to simultaneously minimize both latency and accuracy loss. The NEMOKD proposes a two-phase methodology: knowledge distillation and model evolution. For phase-1, NEMOKD uses a baseline architecture as a starting point to generate variants using evolutionary multiobjective optimization. For phase-2, each generation produces multiple variations of the baseline model; each model is trained using knowledge distillation. The Pareto-optimal models are retrained to form the next-generation models. The parallelizing hardware implementations of the NNs increase frames per second from 6K to 373K, nearly 62× speedup. Evolving student models increase inference accuracy up to 82%.

DLAU, a scalable deep learning accelerator unit, is presented by Wang et al. [55] to improve the performance and reduce the power consumption. The DLAU accelerator employs a three-stage pipeline processing unit that uses tile techniques to explore locality for DNNs. DLAU achieves up to 36.1× speedup, compared to the Intel Core-2 processors, at reduced power consumption.

In addition to efficient usage of existing hardware components, some other works focused on performing optimization from the network architecture perspective, that is, reducing the model size along with the hardware optimization. Model compression/pruning are techniques applied to large and complex networks to remove ineffectual neurons, compress the weights to a binary number, or reduce the weight precision. Such optimizations reduce network size and complexity, and, in many cases, improve performance and power efficiency. We discuss some of the works that applied various optimizations to address different issues with DNNs.

Gao et al. [33] proposed a novel entropy-based pruning for DNNs. The goal of this work is to reduce the network complexity under the performance and latency constraints. To address the latency challenges, reduce network complexity, and to allow DNN deployment on resource-constrained edge devices, Gao et al. [33] present an entropy-based pruning regularizer that can be integrated with existing pruning algorithms. The central theme of the work is that explicitly reducing the information entropy of DNN results in the decrease of search space for the decoder, thus reducing latency. This approach is especially useful for applications that utilize a DNN+decoder type of architecture. Compared to a network without entropy-based pruning, the decoding time is reduced by up to 1.6× in automatic speech recognition (ASR) and 10.6× in name entity recognition (NER); latency is reduced by 1.5× in ASR and 2.1× in NER.

Guan et al. [34] propose Wootz, a compiler-based framework that automatically generates TensorFlow code to materialize composability-based CNN pruning. The author based the work on the observation that two CNN networks in a subspace differ in only some layers. Wootz pretrains the building blocks of the network and then assembles them into the to-be-explored networks, thus trying to shorten the overall network evaluation and the pruning process. As against Wootz, prior CNN pruning methods such as [56] focused on training the networks from scratch. Wootz proposed composability-based CNN pruning to reuse pretrained blocks. It also proposes a novel hierarchical compression-based algorithm that efficiently identifies the set of blocks to pretrain; this maximizes the benefits of computation reuse. Wootz is evaluated for ResNet-50 and Inception-V3; it shortens the pruning process by 186.7× and 30.2×, respectively. The model generated by Wootz is up to 70% smaller compared to the default pruning schemes for the same target accuracy.

Long short-term memory (LSTM) model is another type of DNN deployed for speech recognition. LSTMs are compute intensive and require hefty resources for deployment. Han et al. [36] propose ESE to compress and make the LSTMs sparse so as to be deployed on embedded devices. ESE includes a load-balance-aware pruning method to compress LSTM model size by 20× with almost negligible loss of accuracy. The authors of ESE also propose a scheduler that encodes and partitions the compressed model to multiple processing elements (FPGA) for parallelism. Implemented on a Xilinx XCKU060 FPGA, the sparse LSTM performs at 2.52 TOPS and outperforms Intel Core i7 CPU and Pascal Titan X GPU by 43× and 3×.

To address the irregularity problems occurring due to pruning, Zhu et al. propose a sparse wise dataflow in [57] for low bandwidth and high data sharing in FPGAs. The proposed dataflow skips cycles of processing MAC operations with zero weights and minimizes energy by avoiding unnecessary computations. The design offers 1.5× to 6.7× speedup for AlexNet and VGG-16.

Wang et al. [42] proposed an FPGA-based NN design automation tool called DeepBurning. DeepBurning presents a unified RTL-level hardware accelerator generator and a coordinated compiler that generates the control flow and data layout for various NN topology implementation. RTL library holds scripts to perform logical, arithmetic, approximate lookup table (LUT) operations utilized for implementation on CNN accelerators. This approach uses model compression techniques such as spatial and temporal folding to optimize the hardware dataflow, improve performance, and reduce resource consumption.

Table 1 lists additional FPGA-based accelerator works as well as some of the ASIC-based ML accelerators works.

Table 1. Pruning/compression-based accelerator works for FPGA and ASIC.

Works	HW platform	Key features	Model/Dataset	Throughput and additional details
Gao et al. [33]	Nvidia Jetson Tx2	Reducing the information entropy to reduce search latency	LSTM model / Librispeech dataset and bi-LSTM-CRF model and CoNLL-2003 dataset	Decoding time reduced by 1.6× in ASR and 10.6× in NER
Guan et al. [34]	Test on CPU; FPGA deployable	Wootz: composability-based CNN pruning; hierarchical compression based algorithm	ResNet-50 and Inception-V3 / ImageNet ILSVRC dataset, Flowers, CUB200, Cars, Dogs	Pruning shortened by 186.7× and 30.3×; Models 70% compact
Robert Stewart et al. [35]	Xilinx Z7020 FPGA and Movidius Myriad X VPU	NEMOKD: Two-phase methodology, knowledge distillation and model evolution	MLP, MobileNetV2, Resnet/MNIST, Fashion-MNIST, CIFAR12/100	FPS increased 6 k to 373 k, a 62× speedup; Throughput: 100 - 590 FPS
Song Han et al. [36]	Xilinx XCKU060 FPGA @ 200 MHz	ESE: load-balance-aware pruning	LSTM/TIMIT dataset	282 GOPS Outperforms Core i7 CPU and Pascal Titan X GPU by 43× and 3× on speed
Chen Zhang et al. [37]	Xilinx Virtex7-485T FPGA @ 100 MHz;	Analytical design scheme with loop tiling and transformation	CNN/-	Peak performance of 61.62 GFLOPS @ 100 MHz; 17.42× speedup as against 1× and 3.64× for two CPU-based implementations
Jiantao Qiu et al. [38]	Xilinx Kintex-7 FPGA	Dynamic-precision data quantization method and a conlvolver design	VGG16-SVD / ImageNet ILSVRC	achieves 4.45 FPS; Convolution layer/network performance 187.8 GOPS and 137 GOPS @ 150 MHz, respectively
Motamedi et al. [39]	Xilinx Virtex7-485T FPGA	Design space exploration	AlexNet / -	1.9× speedup
Albericio el al. [40]	TSMC 65nm ASIC	Cnvlutin: eliminate ineffectual operations; hierarchical data-parallel units	Alex, Google, NIN, VGG-19, CNNM, CNNS and GEO/ ImageNet ILSVRC12 dataset	Improved performance: 1.24× to 1.55×; Area overhead: 4.49%; Improved EDP/ED^2P: 1.47× and 2.01×
Lili Song el al. [41]	TSMC 45nm ASIC	C-Brain Data-level parallelism: intern-kernel, intra-kernel and hybrid	Alexnet, GoogleNet, VGG and NiN / ISLVRC	Speedup:4× - 8.3× ;28.04% energy saving; 90.3% on-chip memory energy saving
Wang et al. [42]	Zynq-7045 FPGA	Compiler library, functional approximation, model compression,	MNIST and Cifar Datasets,Alexnet, ANN-0,1,3, NiN	4.7× speedup and 90% energy efficiency than CPU
Guan et al. [43]	Xilinx Virtex7-485t FPGA @150 MHz	Optimize computation performance and communication requirement	LSTm-RNN / -	7.26× GFLOPS peak performance
Rybalkin et al. [44]	Zynq Ultrascale+ XCZU7EV MPSoC	Design space exploration; binarized weights	BiLSTM / OCR plain-text dataset	8.1× GOPS compared to XCKU060 and 1.6× compared to XC7Z045 FPGA
Ma et al. [45]	Altera Arria 10 FPGA	Convolution acceleration with unform PE mapping and minimized data movement	ResNet / ImageNet 2012	ResNet-50/152: 285.1/315.5 GOPS and 27.2/71.7 ms latency, respectively
Zhao et al. [46]	Stratix-V 5SGSD8	Transfer learning with domain-specific application	VGG-16 and ResNet-50 / -	1928 GOPS for VGG-16; 973.3 GOPS for ResNet-50
Jo et al. [47]	65 nm CMOS ASIC	DNN specific instruction set processor for a master-slave configuration	AlexNet/ -	2.17× enhanced efficiency
Chen et al. [48]	65 nm ASIC	Minimize memory transfers, CNN	CNN, DNN	117.87× faster and 21.08× energy-efficient
Chen et al. [49]	28 nm technology ASIC	Multi-chip architecture CNN, DNN	CNNs, CONV, POOL, CLASS, LRN layers	outperforms the performance of a single GPU by up to 450.65× and reduce energy by up to 150.31× using 64 chip system
Daofu Liu et al. [50]	TSMC 65 nm ASIC	One chip storage, CNN	KNN, K-means, DNN, SVM, Decision Tree, Naivie Bayes, MNIST dataset	1.20× faster, and can reduce the energy by 128.41× than NVIDIA K20M GPU
Zidong Du et al. [51]	65 nm CMOS ASIC	Real time object detection, DNN, embedded application of [50], [49]	Simple Conv, LeNet-5, CNN	30× faster than high-end GPU

Table 2. In-memory accelerator design works.

Work	Platform	Memory Type	Key features	Dataset, Algorithm	Precision
DRISA [15]	PIM	DRAM	Bit-wise parallel computing	ImageNet [60], AlexNet [1], VGG-16,19 [61], ResNet-152 [18]	Binary-weighted
DrAcc [76]	PIM	DRAM	Bit-wise computing-based carry look-ahead addition	MNIST [62], CIFAR10 [63], ALexNet, VGG-16,19	Ternary-weighted
SCOPE [77]	PIM	DRAM	Bit-wise stochastic computing	Imagenet, AlexNet, VGG, ResNet	Int8, H2D (Stochastic)
LAcc [83]	PIM	DRAM	LUT-based Matrix Multiplication	MNIST, LeNet, AlexNet, VGG-16,19, ResNet-152	16/32-bit Fixed Point
pPIM [84]	PIM	DRAM	Programmable LUT-cluster-based computing	MNIST, AlexNet, ResNet 18,34,50, VGG-16	8-bit/4-bit Fixed Point
Neural Cache [79]	PIM	SRAM	Bit-serial computing	Inception v3	8-bit
ISAAC [69]	PIM	ReRAM	Analog crossbar-array computing	VGG-(1,2,3,4), MSRA-(1,2,3), Deepface	16-bit
PRIME [70]	PIM	ReRAM	Multi-purpose crossbar-array for storage/computation	MNIST, VGG-D, 3 Layer MLPs	16-bit
PipeLayer [71]	PIM	ReRAM	Crossbar-array-based spiking NN Acceleration	MNIST, ImageNet, AlexNet, VGG	16-bit
AtomLayer [72]	PIM	ReRAM	Layer-wise NN atomic processing on crossbar-array	DCGAN, VGG-19	16-bit
MRIMA [80]	PIM	STT-MRAM	Bit-wise computation on split matrices	ISCAS85	binary
IMCE [14]	PIM	SOT-MRAM	CNN acceleration with variable low-precision	ImageNet, AlexNet	variable low-precision

Accelerators on ASIC

In addition to FPGAs, ASICs offer energy- and area-efficient hardware to accelerate the ML applications. Thus, in this section, we focus on DNN accelerators implemented as ASICs.

Chen et al. [48] proposed DianNao, an ASIC-based implementation of a CNN/DNN accelerator design. The design focuses on optimizing memory transfers to perform operations with high efficiency. An efficient memory utilization achieves high performance and throughput. DianNao was implemented on a 65-nm node in 3.02 mm^2. The accelerator is 117.87× faster and 21.08× energy efficient (average) than a 128-bit SIMD clocked at 2 GHz. Lee et al. [58] proposed a unified neural processing unit (UNPU) for mobile deep learning applications. The UNPU supersedes the previous works' limitations by proposing fully variable weight bit precision, from 1 to 16 bits, and a unified DNN core architecture, supporting CONV layers and FC layers. The UNPU is suitable for mobile processing platforms where power efficiency is a priority. The LUT-based bit-serial processing element achieves energy efficiency compared to traditional fixed-point MAC operations. The UNPU is implemented on a 65-nm CMOS technology for accelerating ImageNet deep CNN (VGG-16).

In a related work to DianNao [48], Chen et al. [49] proposed DaDianNao a supercomputer with the memory capability of one chip storage of all the CNN weights. This custom multichip architecture uses 16-bit precision of state-of-the-art CNNs and DNNs to be implemented using 28-nm technology. The results show that DaDianNao outperforms the performance of a single GPU by up to 450.65× and reduce energy by up to 150.31× using a 64-chip system. This implementation of nodes is done on 28-nm ASIC, where each node has an area of 67.73 mm^2. An extension to the above work is introduced by Liu et al. [50] called PuDianNao, which supports multiple ML scenarios such as classification, regression, and clustering. PuDianNao accelerates ML techniques by extracting their key computational tasks, locality properties, and computational primitives using on-chip storage. The accelerator is designed using the TSMC 65-nm process. Compared with the NVIDIA K20M GPU (28-nm process), PuDianNao (65-nm process) is 1.20× faster and can reduce the energy by 128.41×.

Another extension to the DianNao family is ShiDianNao [51], an accelerator for realtime object detection/recognition algorithms. The main goal of ShiDianNao is to target high performance, low power consumption, and area to suit visual applications as the accelerator is used for integration with realtime applications. The accelerator is embedded inside the processing chip to eliminate off-the-chip DRAM memory accesses and minimize data movements between the SRAM holding the CNN model and the individual processing elements from the sensor. ShiDianNao has an area of 4.86 mm^2 in a 65-nm process and consumes 320.10 mW at 1 GHz; it consumes about 4,700× and 60× less energy than the GPU and DianNao, respectively.

Unlike Lee et al. [58], who exploited weight bit precision, Parashar et al. [59] have utilized sparsity in the data for an efficient accelerator. Parashar et al. [59] proposed SCNN, sparse CNN, that exploits weight and activation sparsity to improve the power and performance of the DNNs. SCNN reduces DRAM accesses by a compressed encoding for zero weights and activations, allowing efficient on-chip RAM storage and eliminating power-hungry DRAM accesses. Compression allows activations of larger networks to reside in on-die buffers, rather than DRAM accesses. The SCNN was implemented in a 16-nm technology on a 7.4-mm^2 ASIC and provides speedup by a factor of 2.7× and 2.3× energy reduction compared to a dense CNN accelerator.

Some other ASIC-based accelerators are listed in Table 1.

In-memory accelerators

In the aforementioned FPGA- and ASIC-based designs, the design of ML accelerators is primarily designed with performance in mind considering the standard design techniques and platforms using standard computation techniques and von Neumann architectures. Another form of computation is the in-memory computing, which is primarily designed to alleviate the data communication overheads between logic and memory elements. Although the concept of performing computations in-memory has been around for a while [64], its implementation began fairly recently [65]–[67]. In-memory processing *also known as* PIM significantly reduces the bandwidth bottleneck, latency, and energy waste from memory-to-processor data communications [22], which in turn enables massively parallel and efficient computations. Therefore, PIM-based hardware accelerators are being widely investigated for high-performance and ultraefficient DNN applications lately [68]. We review some of the PIM-based ML accelerators here.

Proposed by Shafiee et al. [69], ISAAC was the first PIM architecture based on the resistive memory (ReRAM) platform with the capability to perform *in situ* CNN/DNN acceleration. A crossbar-array organization of the ReRAM cells support massively parallel convolutions on the analog domain. This enables it to offer 14.8× higher throughput at 5.5× energy efficiency compared to its ASIC predecessor, DaDianNao [48]. This work was followed by several other ReRAM-based CNN/DNN accelerators such as PRIME [70], PipeLayer [71], AtomLayer [72], and so on with specialization in different areas. For example, PRIME facilitates a wider range of operations than ISAAC (i.e., sigmoid and ReLU functions, max-pooling operation) and bank-level parallelism for higher throughput. Pipelayer, on the other hand, is a spiking NN accelerator with a tiled architecture that supports batch-level intralayer parallelism and interlayer pipelining to obtain over 100× higher energy efficiency (TOPs/W) than ISAAC. An atom layer features a very compact and unique "rotating crossbar" architecture that has 15× smaller footprint than ISAAC while maintaining a 1.1× higher inference throughput.

SRAM-based PIM architectures such as IMAC [73] and XNOR-SRAM [74] also leverage analog crossbar array convolutions for CNN acceleration. IMAC is very similar in architecture to that of ReRAM-based accelerators such as ISAAC and PRIME, but has lower efficiency and performance due to the SRAM leakage issue. XNOR-SRAM, on the other hand, simplifies convolutions to XNOR operations by binarizing and ternarizing the weights [75] which results in over 500× higher energy efficiency (403 TOPS/W) compared to ReRAM-based accelerator, ISAAC. Proposed by Li et al. [15], DRISA is an *in situ* CNN accelerator built on the DRAM technology. DRISA leverages bitline-based computing where multiple logic gates are placed on each bitline to perform inherently bitwise computations with massive parallelism. However, DRISA supports complex operations (i.e., addition, multiplication) with larger precision by incorporating multiple layers of logic gates. Moreover, the DRAM memory architecture is reorganized and the chip is enlarged significantly. DRISA offers an impressive 147 frames/s AlexNet inference throughput at 8-bit binary-weighted precision.

Several other PIM architectures have been proposed [68], such as DrAcc [76], SCOPE [77], and XNOR-POP [78], which also leverage bitline-based computing for CNN/DNN acceleration applications. DrAcc, proposed by Deng et al., is capable of performing carry look-ahead addition, which enables it to perform ternarized weighted CNN inferences. Although its performance is slightly lower (1.7×) than DRISA, it is nearly 49× more energy-efficient and has a negligible (2%) chip area overhead. SCOPE is a DRAM-based PIM accelerator that leverages stochastic approximate computing to perform CNN acceleration. SCOPE has a chip area roughly 4× over a traditional DRAM chip and is capable of performing up to an incredible 2,528 frames/s of AlexNet inferences. XNOR-POP is a small-scale, binarized CNN accelerator on a Wide-IO2 DRAM platform. The convolutions are effectively reduced to bitwise xnor operations [75], in a manner similar to XNOR-SRAM [74], which enables it to offer significantly higher area and energy efficiency.

Alongside DRAM, the SRAM memory technology has also been explored for CNN/DNN acceleration. Neural cache from [79] is a PIM CNN accelerator implemented on the upper-level cache of the microprocessor. This architecture supports bitwise Boolean operations on the SRAM bitlines. It supports operations on multibit precision of operands by performing bit-serial computations with the aid of carry-propagation latches. However, its overall performance is not as impressive as the DRAM-based accelerators, mostly due to the smaller scale of design and power leakage in SRAM.

A novel platform for implementing in-memory CNN/DNN accelerators is the magnetic memory technologies spin transfer torque (STT)-Magnetoresistive random access memory (MRAM) and spin orbit torque (SOT)-MRAM. MRIMA from [80] is a bitwise processing NN accelerator on the STT-MRAM platform. MRIMA performs low-precision inferences by decomposing the convolutions into a set of bitwise operations. Pan et al. [81] proposed a binary CNN accelerator that leverages multilevel cell technology (MLC-STT) to perform XNOR-based convolutions [75]. Aside from the bitwise convolution engine, this architecture also contains a peripheral auxiliary processing unit (APU) peripheral to perform batch normalization, pooling, and so on. Proposed by Angizi et al. [14], IMCE is a CNN accelerator on the SOT-MRAM platform that can perform inferences with different low-end data precision combinations by performing splitting of convolution matrices. Another work from Angizi et al. [82] performs comparator-based DNN inferences on an SOT-MRAM-based bitwise processing PIM architecture.

Apart from the analog crossbar array and bitwise processing PIM architectures discussed so far, a new in-memory computing architecture has lately emerged that leverages in-memory LUTs for performing CNN acceleration. Implemented on DRAM, these architectures replace computationally intensive convolutions with precalculated products of matrix multiplications. Proposed by Deng et al. [83], LAcc is capable of performing high-precision fixed-point multiplications inside LUTs formed by repurposing memory subbanks and performing aggregations with the XOR-based adder block.

Programmable PIM (pPIM) by Sutradhar et al. [84], on the other hand, implements groups of tiny programmable LUTs outside memory subarrays for performing ultraefficient and high-performance CNN acceleration. It can support convolution, ReLU function, and Maxpooling operations by programming the homogeneous LUTs. Instead of operating directly on high-precision operands, this architecture decomposes the operands into smaller fragments and performs multistage partial operations. This allows the LUTs to be compact for minimizing energy waste and area overhead. Both the LUT-based architectures LAcc and pPIM offer significantly better energy efficiency compared to other DRAM-based CNN accelerators such as DRISA and SCOPE while offering a roughly similar range of performance. pPIM offers slightly better inference throughput and lower power consumption than LAcc, albeit with a bigger footprint. A subsequent work from Sutradhar et al. [85] of pPIM also explores the implementations of binarized and ternarized CNN inferences on a similar LUT-based architecture with higher energy efficiency along with optimized performance.

Enduring changes in the ML applications necessitate configurability and the programmability of the targeted hardware platform. Many AI, interactive graphics, and user-specific I/O devices require huge computational power. To facilitate that, lately, a new trend of research is evolving called heterogeneous shared memory computing systems. These heterogeneous multicore systems [86] are beneficial for parallel computing and provide high performance at low latency and low power consumption with improved reliability. Further research is necessary to improve the hardware flexibility and reconfigurability of the system to deploy application-specific ML algorithms.

ML accelerator design trends based on compute methodology

The aforementioned accelerator design techniques perform full-precision and precise computations. However, the DNNs and CNNs are resilient against computation errors, thereby numerous works have started utilizing the approximate computation (AC) units to perform the operations in the CNNs and DNNs. Such techniques not only lead to energy efficiency, but also lead to negligible performance loss and faster inference. We discuss some of the standard and prominent works on the design of ML accelerators using approximate computing in this section. This section discusses new abstraction for approximate computing from two perspectives: A software-level approximation that focuses on programmers' perspectives and a hardware-level approximation that explores the system perspective. Thereafter, finally discusses the innovation of an end-to-end compiler framework for exploiting approximation for hardware acceleration, which also serves as a base for experimenting with new research ideas in the field.

Approximate-computing-based ML accelerator on FPGA

Approximate computing is introduced based on the observation that exact computations demand more resources and require heavy computations. Approximating these computations by selective violation of specifications or allowing selective approximations can provide better efficiency. Mainly the approximate computing techniques can be performed to accelerate the CNN performance when implemented on a hardware platform such as FPGA or ASIC. However promising the results maybe, approximate computing is not a panacea. A naive approximation strategy such as uniform approximation is observed to be inefficient [87]. Effective use of approximate computing necessitates the careful selection of approximable data as well as approximation strategy. Based on the approximable applications, there are numerous approximation strategies such as precision scaling [88], [89], skipping memory accesses [88], performing inexact tasks [90], loop perforation [91], approximate NNs [87], pruning [89], and weight sharing. Deploying them in CNNs and DNNs requires investigation of the tradeoffs between the performance loss and design optimization. In this section, we discuss approximation computing across software and hardware on the FPGA platform.

Among various approximation approaches, neural approximate acceleration has been shown to demonstrate considerable improvement in terms of performance and efficiency when used to supplement CPUs [92]. However, there has not been much research on using the FPGA platform to achieve neural acceleration.

Moreau et al. [92] proposed the systolic NN accelerator in programmable logic (SNNAP), a flexible FPGA-based neural accelerator for approximate programs. SNNAP is designed to work with the compiler workflow that configures the CNN, the technique enables neural acceleration of approximable codes on FPGAs. This technique can be used as either a high-level mechanism in which approximable codes are offloaded to FPGA using a compiler or at a low level in which experienced programmers can exercise fine-grained control using an instruction-level interface.

In a similar interest, Zhang et al. [88] proposed a novel approximate computing framework for artificial NN (ANN) called ApproxANN. ApproxANN performs approximation for both computation and memory accesses, based on the neuron criticality analysis technique to obtain energy efficiency and improve overall performance. Memory access skipping operation on neurons is easily done if the neurons are less critical based on critical analysis. Additionally, approximation strategies such as opting for approximate arithmetic building blocks [93] and performing precision scaling by discarding a specific number of least significant bits of the data are adopted. Most of the approximate ANN designs try to optimize the approximate computational unit in neurons, while memory access may consume a significant portion of energy in ANNs. This approximate neuron design approach for ANNs is an optimization procedure wherein error-tolerance capability and energy consumption are jointly considered to achieve the optimized energy savings under quality constraint.

Similarly, to control the error that occurred due to adaptation of approximation, a novel network structure called AXNet [94] is introduced for the first time that can adopt end-to-end learning. Peng et al. [94] proposed a software-based approximate computing approach AXNet, which deploys two NNs to provide approximate results guided by multitask learning and predict whether the input data is safe to be approximated. AXNet fuses two NNs, both the approximator and predictor together to train the best approximator and consequently the best predictor separately. This work enables end-to-end trainable NNs to perform approximation with quality control is shown to have superior energy efficiency and incurs much less training time than the existing works.

So far, we have observed the traits of software-based approximation where the challenge is constraining imprecision to make it acceptable and now we discuss from the hardware perspective where the challenge is to exploit the programmer's software to improve efficiency. Most of the traditional methods for approximate circuits implement approximate multipliers that targeted approximation of the mantissa multiplier to improve the overall resource consumption of the floating or fixed-point multiplier. This is done by reducing the complexity of the multiplication by truncating the least significant bit (LSB) precision scaling.

Ullah et al. [95] introduced an efficient design methodology for approximate multiplier architecture for FPGA acceleration. The proposed method utilizes LUTs and associated carry chains for the generation of approximate partial products. This method targets to support different real-world applications where the input data is nonuniform, whereas the approximate multipliers are asymmetric, support various quality metrics, and have six error cases with fixed error magnitude for uniform data distribution. The design provides high area, latency, and energy gains, along with better output accuracy than those offered by the state-of-the-art ASIC-based approximate multipliers.

Prabakaran et al. [96] proposed a design methodology called DeMAS for building approximate adders for the FPGA platform. Using this novel methodology, the authors have designed eight unique novel 1-/2-bit approximate adder architectures using various LUTs for the Xilinx 7-series FPGAs. This library contains both hardware RTL in VHDL and behavioral models in MATLAB of the approximate adders customized for the FPGA. Compared to the 16-bit accurate adder, DeMAS design is successful in achieving area, latency, and power-delay product gains of 50%, 38%, and 53%, respectively.

The state-of-the-art implementation of approximate circuits on the ASIC systems has illustrated that the ASIC-based approximate computing offers asymmetrical gains in FPGA-based accelerators [95], [96]. Based on this, Prabakaran et al. [97] proposed ApproxFPGA based on the current state-of-the-art ASIC-based approximate circuit. ApproxFPGA uses ML models to reduce the exploration time for analyzing the state-of-the-art ASIC-based approximate circuits to determine the set of Pareto optimal FPGA-based approximate circuits. This hardware and software codesign model of approximate circuits is designed to be easily used in various arbitrary applications.

A new approximate multiplier which is an approximate log-based multiplier, minimally biased multiplier (MBM), was proposed by Saadat et al. [98] by removing the leading one detector and barrel shifting without accuracy degradation. It is observed that a log-based approximate multiplier is preferred over the approximate multiplier [98], [99] scheme since the mantissa is normalized during the floating or fixed-point multiplication and the leading one is always fixed to the most significant bit (MSB). MBM is an error configurable approximate multiplier that adapts a unique error-reduction mechanism with an approximate log-based multiplier, it offers better error-efficiency tradeoffs than traditional precision scaling.

Following the work done by Saadat et al. [98] for MBM, Mamaghani et al. [99] proposed SIM-Dive, an approximate SIMD architecture based on a novel multiplier and divider with tunable accuracy targeted for the FPGA accelerator. This approach used Mitchell's algorithms for translating the multiplication and division into addition and supports precision variability from 8 to 32 bits. In addition, this hybrid architecture uses a light-weighted error reduction scheme to tune the error to the desired range to achieve higher accuracy. The proposed SIMD multiplier–divider supercedes the accurate SIMD multiplier by achieving up to 26%, 45%, 36%, and 56% improvement in the area, throughput, power, and energy, respectively.

Approximate-computing-based ASIC accelerator

Although FPGA reduces the power consumption in performing approximate computing and optimizing the ML algorithms on the hardware design, efficiency is lower than that of the ASIC [46]. Hence, some of the recent works also started exploring the design of ASIC-based ML accelerators utilizing approximate computing [100]. Unfortunately, there are not a lot of studies exploring approximate computing for the design of ML accelerators. One of the rationales for lower focus on that direction is the lack of reconfigurability in the ASICs [46]. In this section, we discuss one of the approximate computing-based ASIC designs for ML acceleration.

Grigorian et al. [100] proposed a heterogeneous platform BRAINIAC that combines both precise and approximate accelerators to deploy the CNNs. When deploying the CNN on this reconfigurable accelerator, initially execution starts with a simpler approximation and the complexity increases gradually and ends with a precise computation. BRAINIAC used lightweight checks to dynamically control the multistage acceleration flow and output quality to achieve user-specified accuracy during the runtime. Furthermore, the CNN models allow training for different functionalities without the need to modify topology which obviates the need for design flexibility and configurable provided by FPGAs, and therefore BRAINIAC is implemented on the ASIC hardware accelerator. Evaluation of the BRAINIAC platform shows 3× speedup and energy gain on average compared to a system that only uses precise accelerators. Table 3 outlines some of the discussed FPGA and

ASIC-based ML accelerators utilizing approximated computing.

Approximate-computing-based in-memory ML accelerators

Approximate in-memory computing is another emerging field of research in which the computations are carried out inside the memory as an AC instead of accurate computation to reduce the heavy computational load and to improve the performance, speedup, and area. The main aim of approximate computing in PIM architectures is observed to improve the energy efficiency while maintaining the reliability of the system. Approximating memories also give significant reductions in the area used on the hardware.

Hassen and Khokhar [101] proposed an approximate in-memory computing on ReRAM crossbars. To perform approximation on lower computing ReRAM crossbars, approximate reduced ordered binary decision diagrams (ROBDDs) are generated and mapped to the crossbar circuits for flow-based in-memory execution. This approach also performs approximation software designs like pruning and merging the nodes, making the resultant RoBDD more compact than the exact designs. Finally, using the threshold-based approximate equality for synthesizing approximate ROBDDs maintains the desired accuracy.

In another work, SRAM is substituted to phase-changing memory like STT MRAM. The spin- transfer torque RAM (STT-RAM) technique is adopted because of its high density and power leakage, which is near zero. Pj-AxMTJ [102] proposes a joint switching mechanism of the magnetic tunnel junction to perform approximate computing.

The use of ReRAM- and SRAM-based crossbar memories minimizes the weight transfer between the processing units and the off-chip memory but still does not give the required performance for the input and output data movements.

A software-level PIM based on off-the-shelf 3-D-stacked DRAM with approximate computing techniques is proposed in ApproxPIM [103] without requiring the hardware redesign. The approximation in this article is done at the algorithmic level to perform less multiplication–division or complex operations on tunable precision data.

Accelerating approximate pattern [104] proposes the bit map algorithm to do approximate string matching. This software-level approximation takes advantage of SIMD programming to perform a high amount of parallelism, overcoming memory bottleneck by performing PIM to exploit the high internal bandwidth.

Yantır et al. [105], [106] proposed another successful approach where system-level and circuit-level approximation is performed by bit trimming and memresistance scaling. Simply skipping the least significant bits and scaling both the memresistance write voltage and write time has proved to exhibit better results in terms of energy and speedup.

Along with the two approximate approaches, Yantır et al. [106] proposed the hybrid approximate PIM approach to support dynamic approximation. In this work, the operations can be modified on the fly by the controller, and an architectural extension to perform an approximation of the bit-trimming are done dynamically on both ReRAM- and SRAM-based associative processors (APs). The AP performs the operations and processes the data inside the memory

Table 3. Approximate-computing-based accelerator work for FPGA and ASIC.

Work	HW accelerator platform	Key features	Application/ Benchmark	Performance
SNNAP [92]	FPGA (Implemention: Zynq SoC, Cortex-A9, NPU)	Neural acceleration, Compiler framework	MLP, kmeans, sobel, fft	3.8× speedup, 2.8× energy saving with less than 10% quality loss
AXNet [94]	FPGA	Neural approximation	FFT, sobel filter, K-means, MLP	50% reduction of training time
ApproxANN [88]	FPGA	Neural approximation, memory access skipping, precision scaling, approximate arithmetic's	Artificial Neural Network, MNIST, CIFAR 10, CALTECH, NASDAQ datasets	34.11% - 51.72% energy benefit with less than 5% quality loss
ApproxFPGAs [97]	FPGA (Implemention: Xilinx xc7vx485tffg1157-1)	Lookup table based, approximate circuits	Decision tree, Random forest, Regression , MLP	
Ullah et al. [95]	Virtex-7 FPGA	Approximate multiplier, LUTs, approximate partial products, asymmetric multiplier	Compiler library implemented on VHDL	30%, 53%, and 67% gains in terms of area, latency, and energy with below 1% average relative error
DeMAS [96] MBM [98]	Xilinx 7-series FPGA FPGA	Approximate adders, LUTs Approximate multiplier,error-reduction mechanism, approximate log based multiplier	Compiler library	area, latency and power-delay product gains of 50%, 38%, and 53%
SIMDive [99]	FPGA	Approximate multiplier, Mitchell's algorithm, Look up table (LUT) approach, variable precision	Dataset: MNIST, fasion MNIST, ANN and Image processing application	area, throughput, power, and energy gains of 26%, 45%, 36%, and 56%
BRAINIAC [100]	ASIC	Multi stage acceleration, Dynamic error control, neural acceleration	Neural Networks	3× speedup and energy gain on average

in SIMD fashion. This approach has proved that APs support approximate computing inherently and provide a better employment place than other architectures for approximate computing since it facilitates dynamically (runtime) tunable approximation. Table 4 outlines some other FPGAS- and ASIC-based ML accelerators utilizing approximated computing.

Compilers for design of ML accelerators

Various implementations of the CNNs on hardware discussed earlier may deliver promising and significant performance metrics in terms of area, power, delay, energy consumption, and inferences. However, frequent changes in the operating environment might cause performance degradation in terms of accuracy for real-world applications. While implementing the CNN on a hardware accelerator during runtime, the adaptability, flexibility of the network architecture, and reconfigurability of the device for acceleration becomes more evident. Therefore, to improve the real-world applicability and maintain the reliability of the system by reducing the complexity in the ML hardware acceleration compiler frameworks were introduced.

In recent times, a few research groups have taken interest in the adaption of approximation on the hardware accelerator at a software level and proposed a compiler-specific framework for hardware accelerators. Sampson et al. [107] proposed a compiler framework for approximate computing called ACCEPT. This software-specific approximation framework performs approximation strategies like auto-tuning, loop perforation, and parameter selection. This work contributes in unifying the framework to automatically choose the best strategies and a feedback mechanism that explains how annotations can be improved for better approximation opportunities.

Zhang et al. [108] proposed DNNBuilder, an automated tool for building high-performance DNN hardware accelerators for the FPGA platform. The tool bridges the gap between fast DNN construction in software and slow hardware implementation. Novel techniques, such as high-quality RTL NN components, a fine-grained layer-based pipeline architecture, and a column-based cache scheme are developed to boost throughput, reduce latency, and save FPGA on-chip memory. DNNBuilder employs an automatic design space exploration tool to generate optimized design by considering various design factors and constraints. The tool is parameterizable to fine-tune various aspects of the generated design. The tool achieves up to 5.15× faster performance and is 5.88× more efficient compared to published FPGA-based DNN accelerators for edge and cloud computing cases.

Hao et al. [109] proposed an FPGA/DNN codesign methodology to address DNN deployment challenges on the FPGA platform. The authors propose a bottom-up hardware-oriented DNN model search for high accuracy and a top-down FPGA accelerator design considering DNN-specific characteristics. An auto-DNN engine to perform hardware-oriented DNN model search and an auto-HLS engine to generate synthesizable C code of the accelerator are proposed as well. The methodology features 2.48× higher frames per second, 40% lower power consumption, and 2.5× higher energy efficiency.

Panainte et al. [110] proposed a compiler framework for reconfigurable processors, called MOLEN. MOLEN introduces two allocation algorithms for reconfigurable accelerators for maximizing the performance and reducing the reconfiguration overhead. The framework also presents an interprocedural compiler optimization that performs the anticipation of the hardware reconfiguration.

Table 4. Approximate computing on the in-memory processor.

Work	Platform	Memory type	Key features	Application/ Benchmark	Performance
Hassen et al [101]	PIM	ReRAM	Approximate Reduced Ordered Binary Decision Diagram, pruning	72% more compact than the state of the art approximate designs	
Pj-AxMTJ [102]	PIM	SOT-MRAM, STT-MRAM	Approximate adders, Precision scaling, joint switching mechanism		
ApproxPIM [103]	PIM	3D-stacked DRAM	Algorithm-level Approximate computing	K-mean, KNN, BC, BFS, SS	speedup of 1.3x, energy saving of 13%
Yantır et al [105]	PIM	SRAM	Bit trimming and Memresistance scaling	MAC, FFT, FIR, image binarization, convolution operation, sobel, mean filter	energy reduction up to 80x and speedup up to 20x when quality degradation is limited to 10%
Yantır et al [106]	PIM	SRAM, ReRAM	Bit trimming and Memresistance scaling, Dynamic approximation, Architectural extension to [105]	CNN, ANN, FFT, FIR, Conv	average energy savings of 5.17× and 59.11×, and speedups of 2.1× and 3.24× in SRAM-based and ReRAM-based architectures

MOLEN also performs a design space exploration for reconfigurable architectures under the Molen programming paradigm when dynamic reconfiguration is required. MOLEN automatically implements to compile the C language to binary code available for reconfigurable platforms such as FPGAs.

Shan et al. [111] introduce FPMR, a map-reduced framework for the FPGA platform that supports parallel programming. This model provides two primitives that the user needs to design, map, and reduce. During run time, the execution is done in parallel by issuing the map and reducing modules to the computation nodes in a parallel fashion. While task scheduling, data communication and synchronization are done by the framework automatically. Wawrzynek et al. [112] introduced the RAMP design framework to enable an FPGA-based emulator for heterogeneous architectures. The proposed work is a reconfigurable system-on-chip architecture designed for reusable and composable design modules. Additionally, the proposed architecture implements a shared memory multicore system, so all cores in the multicore processor share an off-chip DRAM memory and are deployed on Xilinx's XUP and BEE3 development board.

Given the approximate multiplier-based CNN is implemented on a hardware accelerator, the accuracy loss for this implementation is increased as the number of layers in the network increase. Therefore, to address this issue faced by approximate circuits, a reconfigurable approximate ML-based approach is obtained by applying approximation aware training.

Zervakis et al. [113] proposed an automatic and circuit-independent design framework to generate approximate circuits with dynamically reconfigurable accuracy at runtime. In this article, a power-aware heuristic approach is employed to identify the wires that will be approximated. The proposed framework replaces the wires with approximate switches, so during accurate operation, each switch outputs the wire's value. During approximate operation, the switch output is a constant "0" or "1," limiting the circuit's switching activity. Using this, the most sensitive layers can select more accurate execution while the less sensitive ones can switch to higher approximation and increase the energy/power gains. Controlling the applied level of approximation dynamically at runtime is a key to effectively optimize energy, while still containing and bounding the induced errors at runtime. This approach enables layerwise approximation while implementing the CNN, that is, dynamically set the desired accuracy level per convolution layer at runtime.

Hormigo et al. [114] introduced a self-reconfigurable multiplier suitable for LUT-based FPGA. The proposed article uses a constant multiplier to be able to reconfigure itself in runtime to change the constant value with no restriction. During run time, the architecture enables self-reconfiguration in a two-step process. First, a local mechanism to change the LUTs that contain the partial products is included to avoid long delays. Second, a portion of the multiplier is dedicated to compute the partial products to be stored in the LUTs on the fly, given a constant value. It is observed that the proposed architecture can be reprogrammed in 16 clock cycles, equivalent to less than 100 ns in current FPGAs. This value is significantly smaller than FPGA partial configuration time.

Therefore, it is observed that the dynamic reconfigurable approximate ML approach eliminates the need for retraining the network during run time by obtaining significant energy and power efficiency. So far, all the proposed architectures are for hardware acceleration of CNNs that are capable of implementing only one type of ML technique. A single architecture that is capable of implementing multiple ML techniques such as classification, regression, and clustering have obtained significant interest. Vranjković and Struharik [115] proposed a coarse-grained reconfigurable architecture for hardware acceleration for implementing an ML classifier. This architecture is composed of interconnected reconfigurable processing modules that are organized in a 1-D array to process the instance data in a pipelined manner. This corase-grained reconfigurable architecture (CGRA) accelerator is capable of implementing various kinds of ML classifiers such as decision trees (DTs), ANNs, and support vector machines. So far, most hardware accelerators have implemented a single type of classification algorithm. Additionally, the proposed architecture can implement various versions of ML classifiers such as axis-parallel DTs, oblique DTs, nonlinear DTs, functional DTs, regression DTs, linear and nonlinear SVMs with polynomial and radial kernels, multilayer perceptron (MLP), and radial-basis (RB) ANNs. In addition, we also outline some of the compilers and the achieved acceleration in ML applications in Table 5.

Table 5. Compiler design for approximate ML accelerators.

Work	HW accelerator platform	Key features	Application/ Benchmark	Performance
ACCEPT [107]	FPGA (Implementation:ARM SoC with an integrated FPGA, Zynq FPGA)	Loop perforation, neural acceleration, auto-tuning	Compiler based library	average speedups of 2.3×, 4.8×, and 1.5× on CPU, FPGA, SoC
DNNBuilder [108]	XC7Z045 and KU115 edge and cloud FPGAs	Automated tool for building high-performance DNN hardware accelerator	Alexnet, ZF, VGG16, and YOLO DNNs built and evaluated	Layer-based pipeline architecture and the column-based cache scheme contribute to 7.7x and 43x reduction of the latency and BRAM utilization; best performance (up to 5.15x faster) and efficiency (up to 5.88x more efficient); achieves 4218 GOPS for running object detection
FPGA/DNN Co-Design [109]	Pynq-Z1 FPGA	Auto-DNN engine for hardware-oriented DNN model search and auto-HLS engine to generate C code of accelerator	DNN, SSD, Tiny-Yolo and Yolo	6.2% higher IoU (Intersection-over-Union); 40% lower power, and 2.5× better energy efficiency
RAMP [112]	Xilinx XUP Virtex-II, BEE2 FPGA	Heterogeneous system, Shared memory multicore system		
Zervakis et al. [113]	FPGA	Layerwise approximation, dynamically re-configurable system, Approximate design automation	Adder, sqrt, square,multiplier, mac, divider circuits, ResNet-8, CIFAR 10 dataset	41% improvement in energy for 2% error bound
Hormigo et al. [114]	Spartan 3,6 Virtex 4,7 FPGA	LUT based, Re-configurable multiplier		91% area reduction and up to 102% speed improvement for the case-study circuits tested
Vranjkovic et al [115]	CGRA	Implementation of multiple ML classifiers	Support Vector Machines, Decision trees, ANNs	Classification spped up with MLA working at 113MHz is 28.31 x slower than GPU
MOLEN [110]	Virtex II Pro platform FPGA	Compiler framework, design space exploration, allocation mechanisim	Discrete Cosine Transform (DCT), Variable length coding(VLC), quantization on M-JPEG application.	over all speedup 2.5(about 84% efficiency over the maximal theoretical speedup of 2.96).
FPMR [111]	FPGA	Compiler framework, Supports parallel programming, automatic task scheduling	support vector machine, page rank	Speedup is 31.8x versus CPU-based implementation

Neuromorphic accelerators-based on emerging memory

Another design paradigm of DNNs is to design the ML networks inspiring from the design methodology of biological NNs. A number of ML accelerator designs aim to maximize energy efficiency by more strictly adhering to computational principles underlying biological intelligence. This paradigm, called neuromorphic computing, was brought into focus in the late 1980s by Mead [116] at Caltech. In essence, Mead [116] advocated the use of analog and mixed-signal circuits to emulate the behavior of neural tissue, which could enable orders of magnitude better energy efficiency than any digital technology. Today, the term neuromorphic has become slightly ambiguous, often referring to any ASIC ML accelerators. Here, we focus only on spike-based designs that employ emerging nonvolatile memory (NVM) technologies to emulate synaptic weighting between neurons.

Emerging memory technologies for neuromorphic accelerators

The number of weights in an NN typically grows superlinearly (quadratically in the worst case) with the number of activation units, making the design of low-area and low-energy weight circuitry a key priority for neuromorphic computing. In biology, chemical synapses perform three primary functions in the nervous system: 1) modulating communication strength between pre- and postsynaptic neurons; 2) providing a physical communication channel; and 3) facilitating learning. A number of emerging NVM devices also share these characteristics and are being intensely investigated for neuromorphic computing applications. Table 6 shows a comparison of different emerging NVM technologies across several key metrics. Each device exploits different physics to achieve nonvolatile storage of resistance states. In phase-change memory (PCRAM), a phase change material such as $Ge_2Sb_2Te_5$ or AgInSbTe is transformed between amorphous and crystalline phases to increase or decrease its resistance value, respectively. One challenge associated with PCRAM is the high current that is required to melt the phase change material, leading to large write energies. Another drawback is the relatively slow write latency that stems from the time required for the crystallization process [119]. On the other hand, both ReRAM and STTRAM have low write energies and latencies, making them better suited for neuromorphic DNN accelerators with on-chip learning. At its core, STTRAM is a magnetic tunneling junction (MTJ), where electron tunneling probability is modulated

Table 6. Comparison of NVM technologies for neuromorphic computing [117], [118].

	PCRAM	ReRAM	STTRAM
Cell Size	4-20F^2	4-6F^2	12F^2
Write Energy	6 nJ	2 nJ	< 1 nJ
Write Latency	100 ns	10 ns	2-25 ns
Read Latency	10 ns	1-10 ns	2-25 ns
MLC	4 bits	4-6 bits	2 bits
Endurance	10^6-10^9	10^6-10^9	10^{14}
Retention	10^4 s	10^4 s	-
ON/OFF Ratio	10^3	10^3	10

by modifying the relative magnetic orientation of two ferromagnetic films. Two drawbacks of STTRAM are its small number of achievable resistance states (2 bits) and a low ON/OFF conductance ratio. In contrast, ReRAM, which operates by modulating the distribution of defects like oxygen vacancies in an insulating transition metal oxide film, offers a high ON/OFF ratio and offers 4–6 bits of memory, which enables quantized NN inference with limited accuracy degradation [120].

NVM crossbars for ML acceleration

The most common way to integrate NVMs into neuromorphic architectures is with a crossbar structure, as shown in Figure 1, which offers a $4F^2$ cell size, where F is the wire half-pitch. The NVM conductances implement the weight matrix W between two densely connected layers of neurons

$$s^l(t) = W^l(t)\, x^{l-1}(t) \qquad (1)$$

where $x(t)$ and $s(t)$ represent the vectors of crossbar inputs (rows/word lines) and outputs (columns/bit lines) at time t, respectively. More specifically, $x(t)$ is a collection of spike trains for layer $l-1$ and $s(t)$ are the currents driving the spiking neuron membrane capacitances for layer l. Note that, in practice, the crossbar inputs $x(t)$ can be represented as either voltages or currents with tradeoffs discussed in [121]. In biology, synapses typically have either an exclusively excitatory or an inhibitory effect on postsynaptic neurons, although corelease of both neurotransmitter types has been observed in some chemical synapses [122]. In contrast, deep learning algorithms depend on the flexibility of all weights to be positive or negative and to smoothly transition between positive and negative values (or *vice versa*) during training. A typical strategy for achieving bipolar weights is to use two NVM devices to

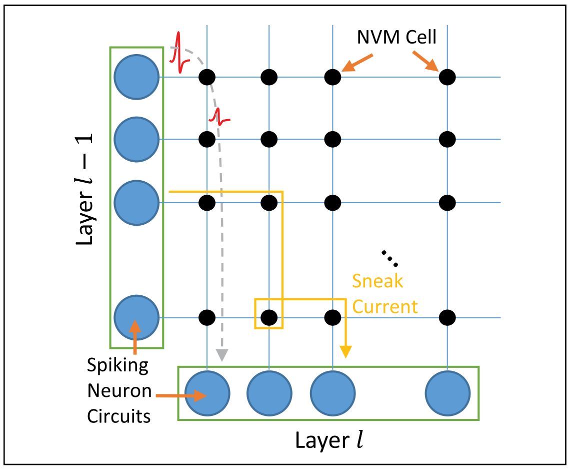


Figure 1. NVM crossbar circuit for implementing weight matrix between two FC neuron layers in a neuromorphic DNN accelerator.

represent a single weight value, where one represents the positive component and the other represents the negative component. Then, the current entering the postsynaptic neuron is computed as the difference between the currents flowing through the NVMs representing each component (see [123] and [124]).

Crossbar sizes up to 128 × 128 have been demonstrated [125], and multiple crossbars can be combined to implement larger layers. However, a key challenge, especially in large crossbars, is sneak path current, which is current that flows in one of several unintended parallel current pathways, and is affected by several factors such as the NVM off conductance, the potential at the inputs of postsynaptic neurons, parasitic wire resistance, and more [126]. Sneak currents not only cause memory read and write errors, but also increase the crossbar power consumption. Several mitigation strategies such as row/column grounding [127], algorithmic approaches (e.g., [128]), and use of selector devices [129] exist. The use of MOSFET selectors is the most common approach to mitigating sneak paths and allows for easy and controllable isolation of NVM cells to avoid interference between cells on adjacent bit lines. Two-terminal NVMs like ReRAM integrated with MOSFET selectors are called 1T1R cells. Other NVMs, such as ferroelectric FETs (FeFETs) [130] and other gated synaptic devices [131], employ a 3-terminal structure, so their selector is "built-in."

Spiking neuron circuits

A key component of neuromorphic systems is the circuitry that emulates neuronal behavior. Neuromorphic neuron circuits are usually implemented in CMOS (although other implementations have been proposed such as those based on memristors [27] and biristors [28]), and they have varying levels of biological realism. Threshold logic gates based on high-gain CMOS logic [29], [30] provide the simplest level of modeling for the so-called first-generation NNs. These circuits facilitate the implementation of binary activation functions, which were the key building blocks of perceptrons [132], and more recently have become popular in deep binarized NNs [31], [32]. The 0/1 output of these neuron circuits represents whether a neuron is firing or not firing. Second-generation NNs, with continuous activation functions, employ neurons that represent the average spike code over a given time window. Circuits for common deep learning activations such as sigmoid [123], ReLU [133], and softmax [134] have been known for quite some time. However, a common problem in most of the second-generation circuits is that they have large-area and -power overheads stemming from dc biasing, transistor matching requirements, and so on. To reduce these overheads while maintaining quasicontinuous outputs, third-generation neuron circuits produce "all-or-nothing" voltage (or current) spikes that can encode information in a variety of ways such as spike rate [131], [135]. Mead's original axon hillock spiking neuron circuit [135] uses only a handful of transistors and capacitors to achieve spike rate encoding of a stimulus current. However, by increasing the circuit complexity, a number of phenomena observed in real neurons can be replicated in silicon, such as refractory period, tonic and bursting behaviors, spike rate, and spike threshold adaptation [136]. In addition, other neuron circuits employ temporal encoding [137] and stochastic spike behavior [138]. Common figures of merit for spiking neuron circuits include energy-per-spike and maximum spike frequency. The design proposed in [131] operates at approximately 1 pJ/spike with a maximum operating frequency close to 10 MHz. Other designs may offer lower energy at lower operating frequencies and vice versa. For example, Sourikopoulos et al. [139] propose a spiking neuron circuit operating at just 4 fJ/spike, but with a maximum operating frequency of only 26 kHz due to its deep subthreshold operation. Because of this fundamental tradeoff in speed and energy efficiency, it is important to understand application requirements such as whether or not realtime or faster than realtime processing is needed and battery capacity.

Training NVM-based neuromorphic accelerators

Neuromorphic DNN accelerators can be designed for pure inference functionality or training+inference, depending on the application. In cases where on-chip training is not needed, neuromorphic accelerators are programed by first training a model of the neuromorphic hardware offline using standard CPU/GPU resources and any of the number of NN software frameworks such as Tensorflow, PyTorch, and so on. Then the parameters of the offline-trained model are transferred to the neuromorphic hardware by programming the conductance states of the individual NVM elements. It is important that the offline model contains sufficient detail about the behavior of the neuromorphic hardware behavior/constraints (e.g., number of neurons/synapses, maximum neuron fan-in and fan-out, and number of conductance states per NVM element). Some hardware behavior is difficult to model exactly or may change over time, such as probabilistic behavior caused by CMOS and NVM process variation and NVM cycle-to-cycle programming variations. In this case, incorporating the physical hardware into the training process is required via on-chip training or hardware-in-the-loop training to avoid accuracy degradation [140]. For on-chip training, significant and open challenges include NVM asymmetric nonlinearity [141] and low-precision training [142]. For neuromorphic systems that employ spiking neurons (i.e., not just a continuous representation of spike rate), there are some additional challenges and opportunities. In the case of supervised learning, a key challenge is the implementation of gradient descent optimization with nondifferentiable spiking activation functions, as well as determining proper temporal credit assignment [118], [143]. However, there are also opportunities to implement more biologically plausible unsupervised learning algorithms such as spike timing-dependent plasticity [144] leveraging incremental conductance updates in NVM devices.

Benchmarking and ML in hardware design and automation

The business need to take semiconductor and hardware products faster to market is more than ever before. This constantly pushes the quest for further efficiency improvements in the design and automation of hardware designs. One exciting means toward this end in the last few years has been the accelerated adoption and infusion of ML and AI techniques. This has spanned across the hardware design and the related automation ecosystem. ML enables extraction of high-level features and their reuse across related scenarios. The adoption of ML in EDA tools and techniques now spans across almost all stages of the hardware design flow, with noteworthy results. Starting right from hardware design exploration, logic design, logic debug, synthesis, timing, power, noise, placement, routing, verification, manufacturing, and test. The rapid advent of ML techniques in EDA was also enabled by a low barrier for the adoption of a mature set of ML libraries and high-level programming application programming interface (APIs).

Over the last few decades, the EDA [145] tools and flows have evolved into the EDA 3.0 [146] era. In the EDA 1.0 era discrete design flow (shown in Figure 2), higher level languages were more logically written, before logic synthesis became mainstream and industry class. O(n log n) or O(n log 2n) consisted of spectral methods, network flows, and different kinds of approximations. Data structures mostly spanned quad trees [147], [148], KD trees [149], and BDD [150]. Over time, the EDA 1.0 flow came to an end, with increasing design complexity and reducing margins for hardware design guard bands. In the EDA 2.0 era (Figure 3), or what is now known as the "integration era," the focus was on increased integration across the data, tools, and flows. This slowly paved the way for the EDA 3.0 era, with exploding size of hardware design data and increased IO overheads.

In the present EDA 3.0 era, the desire has been to move toward more modular microservices-based, predictive design flows, which can enable a new wave of experiences for hardware design teams. The end use-cases include better decision making in core techniques, where the "trained" model is leveraged to determine EDA techniques, and better leverage cloud computing. Increasing the number of production class EDA flows utilize "guidance" from the trained models [151]. Reinforcement learning is increasingly automating the management of such systems and flows. Cloud burst [152] and related techniques allow flexibility and scalability to high performance computing (HPC)-like EDA microservice workloads. These workloads also aim to leverage the different benchmarked hardware accelerators for AI and ML in the cloud ecosystem [153], [154].

As outlined in [151], ML is now used across the design automation flow. Zhuo et al. [155] discuss the application of ML techniques in physical design automation. Linear regression, random forest [156], and ANNs [157] are classical regression models, while the support vector machine [158] is a classification algorithm for tasks with a smaller training set. Classification models like *K*-nearest-neighbor (KNN) [159] and RF can be combined with other techniques. ML techniques have been applied to improve high-level synthesis from the following: quicker result estimation [160], refining design space explorer (DSE) algorithms [161], and transforming DSE as an active learning problem [162]. Logic synthesis is an optimization problem with relatively complex constraints. Here, ML techniques are leveraged to better schedule optimization strategies. There are several logic transformations in the current set of EDA synthesis tools, such as ABC [163]. To select an appropriate

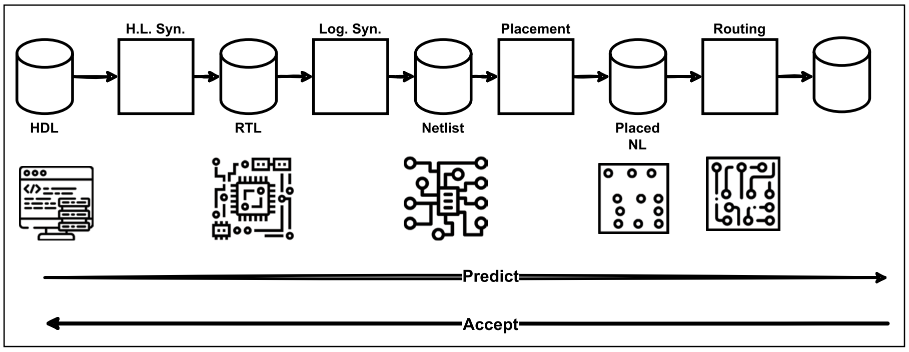


Figure 2. EDA 1.0 Era—discrete design flow.

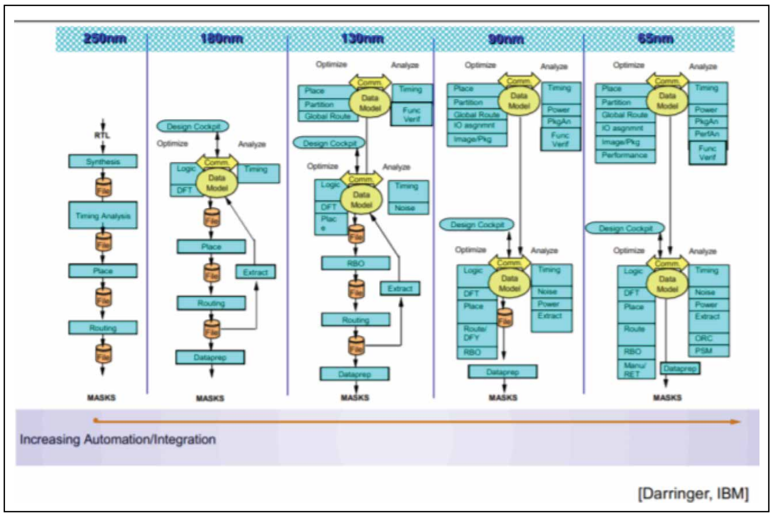


Figure 3. EDA 2.0 Era—integrated design tools on distributed servers.

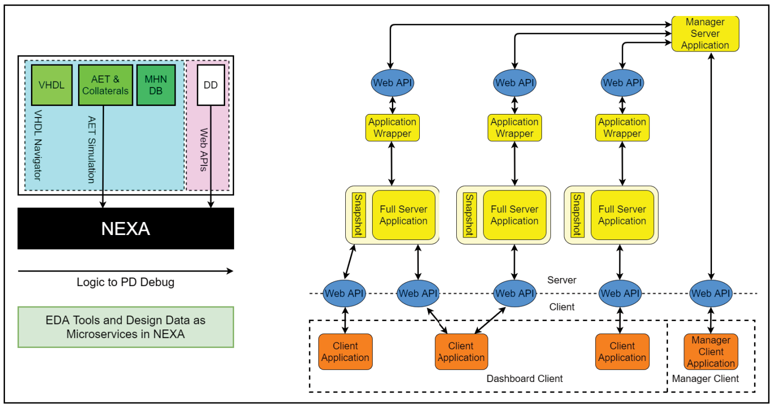
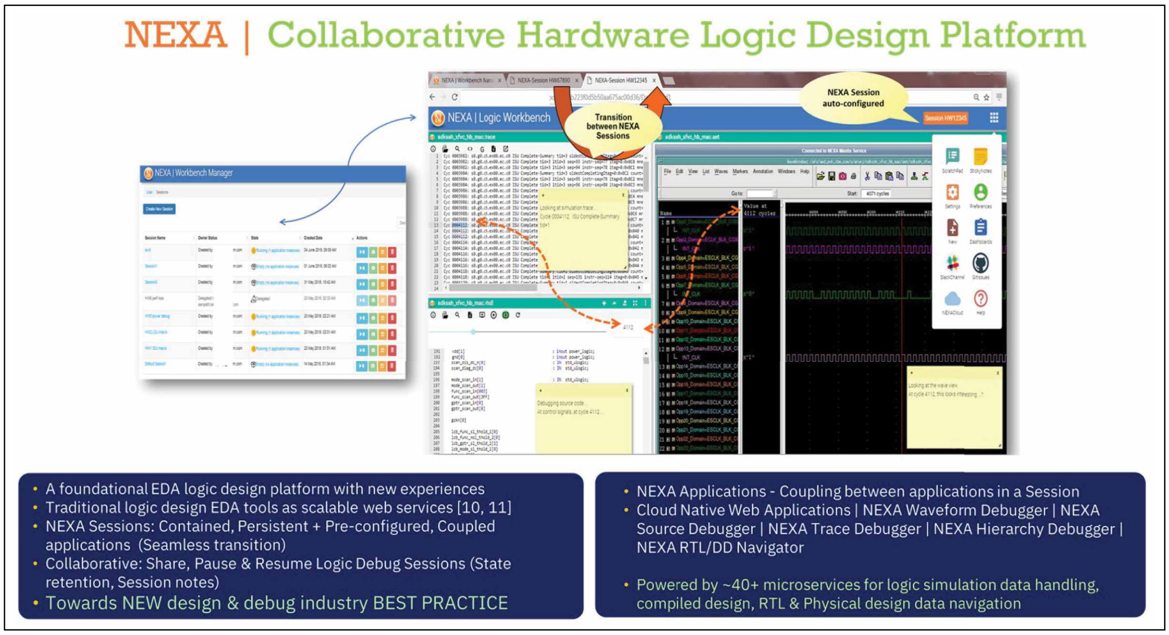


Figure 4. EDA 3.0 Era—data and tools as cloud-native microservices.

synthesis flow, Yu et al. [164] presented a classification problem and designed a CNN to map the synthesis flow to improved quality of results. In PADE [165], Ward et al. present a placement flow for automatic data path extraction. Here, the placement of data path logic is conducted independent of the random logic.

It is not easy to determine routing during the placement stage. In [166], CNN is leveraged for design rule checking hotspot detection. The input features of the fully convolutional network (FCN) are outputs of rectangular uniform wire density and a prerouting congestion estimate. Kahng et al. [167] presented a framework to reduce the number of mismatches between sign-off timing analysis tools. Recent techniques aim to reuse ML models that take specifications of an application domain and predict results in the new domain, based on results acquired in the different original domains. RL models pretrained on one task can perform a new task in a new domain [113], [147], [168], [169], which reduces overall training time and overheads. Having said this, there is an opportunity to create better standards across the industry EDA tool and data sets. This will allow for more interoperability of techniques across the industry ecosystem, as also outlined in a survey [170] published by Si2.

In the EDA3.0 era, as shown in Figure 4, EDA tools and design data become scalable and secure microservices that can be used in cloud native platforms and applications [171], [172]. This paves the foundation for a cost-efficient solution for enabling


NEXA | Logic Workbench
NEXA | Workbench Manager
Transition between NEXA Sessions
NEXA Session auto-configured

NEXA | Collaborative Hardware Logic Design Platform

- A foundational EDA logic design platform with new experiences
- Traditional logic design EDA tools as scalable web services [10, 11]
- NEXA Sessions: Contained, Persistent + Pre-configured, Coupled applications (Seamless transition)
- Collaborative: Share, Pause & Resume Logic Debug Sessions (State retention, Session notes)
- Towards NEW design & debug industry BEST PRACTICE

- NEXA Applications - Coupling between applications in a Session
- Cloud Native Web Applications | NEXA Waveform Debugger | NEXA Source Debugger | NEXA Trace Debugger | NEXA Hierarchy Debugger | NEXA RTL/DD Navigator
- Powered by ~40+ microservices for logic simulation data handling, compiled design, RTL & Physical design data navigation

Figure 5. Cloud-native platform for collaborative hardware logic debug.

new designer experiences and improved designer efficiency. The ability to closely operate with analytics and AI techniques services provided most leading cloud providers and web technology ecosystem reuse and building of new solutions faster. The EDA3.0 direction is equally (or more) applicable for highly interactive use cases and workloads, in addition to the traditional HPC-like EDA workloads. In such interactive EDA workloads, the design automation platform microservices can be augmented to learn from the different transactions on the platform. Additionally, to meet interactive application responsiveness, such workloads can also leverage ML and AI techniques and related accelerators on edge devices [173], [174], in addition to the techniques on the cloud.

One example of an interactive EDA3.0 workload is hardware logic debug. Large cycles of the overall chip design cycle are spent in debugging the root cause of fails relating to mainline functional aspects. Figure 5 shows NEXA [172], an industry-first cloud-native platform for enabling collaborative design and debug of industry class hardware designs. Here, the focus is put to transform the logic design and debug EDA tools and data sources into scalable microservices. These microservices are foundational for the logic design platform, with web applications in the platform providing new experiences for hardware design. One such experience is that of a NEXA session, for enabling collaborative hardware logic design. As shown in Figure 5, a hardware logic designer can create NEXA sessions, say for each hardware logic debug context, and then “share, pause, and resume” these logic debug sessions. Another related experience is that these sessions are contained and persistent, enabling state retention and seamless transition across debug contexts. These experiences of creating and sharing sessions reduce the overall time taken for debugging next-generation hardware designs. In addition to mainline hardware function design and debug, the platform also supports nonmainline hardware aspects [175]–[178] like the test, clocking, timing, power, and codebug of physical design-driven logic changes.

In the EDA 3.0 setting, it is important, than ever before, to benchmark ML EDA tool and data microservices and techniques. This drives trust, better decision-making, and broader deployments, while maintaining cost-efficiency. To further this direction, benchmarking and providing design data as microservices provides new opportunities for rapid innovation across the hardware design ecosystem.

We presented a quick overview on accelerators, underlying hardware platforms, computation paradigms, compiler frameworks, and the evolution of EDA in the context of ML/AL infused hardware and accelerator design. Different designs use different frameworks, evaluation platforms, and design paradigms for the evaluation of viable techniques. Controlled design environments and specific design techniques make benchmarking and broader comparisons an interesting problem statement, which drives several new technologies, research, and innovation. The amount of data to be analyzed grows, constantly driven by aggressive business requirements, digital transformation, and desire from end-users, at each touch point, for better and new design experiences. The design of future ML accelerators continues to rapidly evolve as a multiobjective optimization, with ML accelerator designs and frameworks "self-assisted" by AI/ML techniques. In the future, new and more complex networks will evolve for various application domains. Therefore, hardware architectures are required to improve, based on new approximate computing, reconfigurable computing, to efficiently run and to adapt to the latest ML models. Furthermore, the deployment of the network architecture cannot be separated from system architecture. Flexibility and programmability of the system to configure to the new transformations in the deep learning algorithms determine the lifetime, adaptability, and efficiency of the system. Therefore, further research for new design environments, such as reconfigurable hardware and heterogeneous architectures, need to be explored. ■

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[2] C. Szegedy et al., "Going deeper with convolutions," 2014, *arXiv:1409.4842*.

[3] M. Wess, S. M. P. Dinakarrao, and A. Jantsch, "Weighted quantization-regularization in DNNs for weight memory minimization toward HW implementation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2929–2939, Nov. 2018.

[4] S. D. Barve et al., "Adversarial attack mitigation approaches using RRAM-neuromorphic architectures," in *Proc. ACM Great Lakes Symp. VLSI (GLSVLSI)*, 2021, pp. 201–206.

[5] S. M. P. Dinakarrao, "Self-aware power management for multi-core microprocessors," in *Proc. IEEE Int. Green Sustain. Conf. (IGSC)*, vol. 29, Mar. 2021, p. 100480.

[6] S. Shukla et al., "RNN-based classifier to detect stealthy malware using localized features and complex symbolic sequence," in *Proc. IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2019, pp. 406–409.

[7] S. Shukla et al., "On-device malware detection using performance-aware and robust collaborative learning," in *Proc. IEEE Design Autom. Conf.*, Dec. 2021, pp. 967–972.

[8] S. Shukla et al., "Stealthy malware detection using RNN-based automated localized feature extraction and classifier," in *Proc. IEEE Int. Conf. Tools With Artif. Intell. (ICTAI)*, Nov. 2019, pp. 590–597.

[9] M. M. Ahmed et al., "AWARe-Wi: A jamming-aware reconfigurable wireless interconnection using adversarial learning for multichip systems," in *Proc. IEEE Int. Green Sustain. Conf. (IGSC)*, vol. 29, Mar. 2021, p. 100470.

[10] S. M. P. Dinakarrao et al., "Cognitive and scalable technique for securing IoT networks against malware epidemics," *IEEE Access*, vol. 8, pp. 138508–138528, 2020.

[11] S. M. P. Dinakarrao et al., "Adversarial attack on microarchitectural events based malware detectors," in *Proc. Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[12] X. Guo et al., "Deep multi-attributed graph translation with node-edge co-evolution," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2019, pp. 250–259.

[13] Z. Chen et al., "Estimating the circuit de-obfuscation runtime based on graph deep learning," in *Proc. ACM/EDAA/IEEE Design Autom. Test Eur. (DATE)*, 2020, pp. 358–363.

[14] S. Angizi et al., "IMCE: Energy-efficient bit-wise in-memory convolution engine for deep neural network," in *Proc. Asia South Pacific Design Autom. Conf. (ASP-DAC)*, 2018, pp. 111–116.

[15] S. Li et al., "DRISA: A DRAM-based reconfigurable in-situ accelerator," in *Proc. IEEE/ACM Int. Symp. Microarchit.*, Oct. 2017, pp. 288–301.

[16] M. Lechner et al., "ResCoNN: Resource-efficient FPGA-accelerated CNN for traffic sign classification," in *Proc. Int. Green Sustainable Comput. Conf.*, 2019, pp. 1–6.

[17] A. Dhavlle and S. M. P. Dinakarrao, "A comprehensive review of ML-based time-series and signal processing techniques and their hardware implementations," in *Proc. IEEE Int. Green Sustain. Conf. (IGSC)*, Oct. 2020, pp. 1–8.

[18] K. He et al., "Deep residual learning for image recognition," 2015, *arXiv:1512.03385*.

[19] A. A. Awan, H. Subramoni, and D. K. Panda, "An in-depth performance characterization of CPU- and GPU-based DNN training on modern architectures," in *Proc. Mach. Learn. HPC Environ.*, 2017, pp. 1–8.

[20] S. Bavikadi et al., "uPIM: Performance-aware online learning capable processing-in-memory," in *Proc.*

IEEE 3rd Int. Conf. Artif. Intell. Circuits Syst. (AICAS), Jun. 2021, pp. 1–4.

[21] S. Pagani et al., "Machine learning for power, energy, and thermal management on multicore processors: A survey," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 1, pp. 101–116, Jan. 2020.

[22] S. Lu et al., "Scaling the 'memory wall': Designer track," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2012, pp. 271–272.

[23] A. Ganguly, R. Muralidhar, and V. Singh, "Towards energy efficient non-von Neumann architectures for deep learning," in *Proc. Int. Symp. Qual. Electron. Design (ISQED)*, 2019, pp. 335–342.

[24] D. Patterson et al., "A case for intelligent ram," *IEEE Micro*, vol. 17, no. 2, pp. 34–44, Mar./Apr. 1997.

[25] P. D. S. Manoj et al, "A scalable network-on-chip microprocessor with 2.5 D integrated memory and accelerator," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 6, pp. 1432–1443, Jun. 2017.

[26] H. S. Stone, "A logic-in-memory computer," *IEEE Trans. Comput.*, vol. C-19, no. 1, pp. 73–78, Jan. 1970.

[27] R. Midya et al., "Artificial neural network (ANN) to spiking neural network (SNN) converters based on diffusive memristors," *Adv. Electron. Mater.*, vol. 5, no. 9, 2019, Art. no. 1900060.

[28] J.-W. Han and M. Meyyappan, "Leaky integrate-and-fire biristor neuron," *IEEE Electron Device Lett.*, vol. 39, no. 9, pp. 1457–1460, Sep. 2018.

[29] M. Soltiz et al., "Memristor-based neural logic blocks for nonlinearly separable functions," *IEEE Trans. Comput.*, vol. 62, no. 8, pp. 1597–1606, Aug. 2013.

[30] D. Chabi et al., "Robust neural logic block (NLB) based on memristor crossbar array," in *Proc. IEEE/ACM Int. Symp. Nanoscale Archit.*, Jun. 2011, pp. 137–143.

[31] I. Hubara et al., "Binarized neural networks," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4114–4122.

[32] C. Merkel and A. Nikam, "A low-power domino logic architecture for memristor-based neuromorphic computing," in *Proc. Int. Conf. Neuromorphic Syst.*, 2019, pp. 1–4.

[33] D. Gao et al., "Rethinking pruning for accelerating deep inference at the edge," in *Proc. SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2020, pp. 155–164.

[34] H. Guan, X. Shen, and S.-H. Lim, "Wootz: A compiler-based framework for fast CNN pruning via composability," in *Proc. 40th ACM SIGPLAN Conf. Program. Lang. Design Implement. (PLDI)*, 2019, pp. 717–730, doi: 10.1145/3314221.3314652.

[35] R. Stewart et al., "Optimising hardware accelerated neural networks with quantisation and a knowledge distillation evolutionary algorithm," *Electronics*, vol. 10, no. 4, p. 396, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/4/396

[36] S. Han et al., "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2017, pp. 75–84.

[37] C. Zhang et al., "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2015, pp. 161–170.

[38] J. Qiu et al., "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2016, pp. 26–35.

[39] M. Motamedi et al., "Design space exploration of FPGA-based deep convolutional neural networks," in *Proc. 21st Asia South Pacific Design Autom. Conf. (ASP-DAC)*, 2016, pp. 575–580.

[40] J. Albericio et al., "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 1–13.

[41] L. Song et al., "C-brain: A deep learning accelerator that tames the diversity of CNNs through adaptive data-level parallelization," in *Proc. 53nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2016, pp. 1–6.

[42] Y. Wang et al., "DeepBurning: Automatic generation of FPGA-based learning accelerators for the neural network family," in *Proc. 53nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, 2016, pp. 1–6.

[43] Y. Guan et al., "FPGA-based accelerator for long short-term memory recurrent neural networks," in *Proc. 22nd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, 2017, pp. 629–634.

[44] V. Rybalkin et al., "FINN-L: Library extensions and design trade-off analysis for variable precision LSTM networks on FPGAs," in *Proc. 28th Int. Conf. Field Program. Log. Appl. (FPL)*, 2018, pp. 89–897.

[45] Y. Ma et al., "End-to-end scalable FPGA accelerator for deep residual networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

[46] R. Zhao et al., "Towards efficient convolutional neural network for domain-specific applications on FPGA," in *Proc. 28th Int. Conf. Field Program. Log. Appl. (FPL)*, 2018, pp. 1–8.

[47] J. Jo et al., "Dsip: A scalable inference accelerator for convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 53, no. 2, pp. 605-618, Nov. 2018.

[48] T. Chen et al., “DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning,” in *Proc. Archit. Support Program. Lang. Oper. Syst.*, 2014, pp. 269–284.

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

[50] D. Liu et al., “PuDianNao: A polyvalent machine learning accelerator,” in *Proc. 20th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*. New York, NY, USA: Association for Computing Machinery, 2015, pp. 369–381, doi: 10.1145/2694344.2694358.

[51] Z. Du et al., “ShiDianNao: Shifting vision processing closer to the sensor,” in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2015, pp. 92–104.

[52] J. Qiu et al., “Going deeper with embedded FPGA platform for convolutional neural network,” in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2016, pp. 26–35.

[53] S. Chakradhar et al., “A dynamically configurable coprocessor for convolutional neural networks,” *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 247–257, Jun. 2010, doi: 10.1145/1816038.1815993.

[54] C. Farabet et al., “NeuFlow: A runtime reconfigurable dataflow processor for vision,” in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Workshops*, Jun. 2011, pp. 109–116.

[55] C. Wang et al., “DLAU: A scalable deep learning accelerator unit on FPGA,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 3, pp. 513–517, Mar. 2017.

[56] M. M. Pasandi et al., “Modeling of pruning techniques for deep neural networks simplification,” 2020, *arXiv:2001.04062*.

[57] C. Zhu et al., “An efficient hardware accelerator for structured sparse convolutional neural networks on FPGAs,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 9, pp. 1953–1965, Sep. 2020.

[58] J. Lee et al., “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, Jan. 2019.

[59] A. Parashar et al., “SCNN: An accelerator for compressed-sparse convolutional neural networks,” in *Proc. Int. Symp. Comput. Archit. (ISCA)*, 2017, pp. 27–40.

[60] J. Deng et al., “ImageNet: A large-scale hierarchical image database,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.

[61] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, *arXiv:1409.1556*.

[62] L. Deng, “The MNIST database of handwritten digit images for machine learning research [best of the web],” *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, Nov. 2012.

[63] A. Krizhevsky, V. Nair, and G. Hinton, “Cifar-10 (Canadian Institute for Advanced Research),” Univ. Toronto, Toronto, ON, Canada, Tech. Rep. [Online]. Available: https://academictorrents.com/details/463ba7ec7f37ed414c12fbb71ebf6431eada2d7a

[64] H. S. Stone, “A logic-in-memory computer,” *IEEE Trans. Comput.*, vol. C-19, no. 1, pp. 73–78, Jan. 1970.

[65] M. Gokhale, B. Holmes, and K. Iobst, “Processing in memory: The terasys massively parallel PIM array,” *Computer*, vol. 28, no. 4, pp. 23–31, Apr. 1995.

[66] D. Patterson et al., “A case for intelligent ram,” *IEEE Micro*, vol. 17, no. 2, pp. 34–44, Mar. 1997.

[67] D. G Elliott et al., “Computational ram: Implementing processors in memory,” *IEEE Design Test*, vol. 16, no. 1, pp. 32–41, Jan./Mar. 1999, doi: 10.1109/54.748803.

[68] S. Bavikadi et al., “A review of in-memory computing architectures for machine learning applications,” in *Proc. Great Lakes Symp. VLSI*, Sep. 2020, pp. 89–94.

[69] A. Shafiee et al., “ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars,” in *Proc. Int. Symp. Comput. Archit. (ISCA)*, Sep. 2016, pp. 89–94.

[70] P. Chi et al., “PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory,” in *Proc. ACM/IEEE Int. Symp. Comput. Archit. (ISCA)*, 2016, vol. 44, no. 3, pp. 27–39.

[71] L. Song et al., “PipeLayer: A pipelined ReRAM-based accelerator for deep learning,” in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 541–552.

[72] X. Qiao et al., “AtomLayer: A universal ReRAM-based CNN accelerator with atomic layer computation,” in *Proc. ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jan. 2018, pp. 1–6.

[73] A. Mustafa et al., “IMAC: In-memory multi-bit multiplication and accumulation in 6T SRAM array,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 8, pp. 2521–2531, Mar. 2020.

[74] S. Yin et al., “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, Jan. 2020.

[75] M. Rastegari et al., “XNOR-Net: ImageNet classification using binary convolutional neural networks,” in *Proc. Eur. Conf. Comput. Vis.*, Oct. 2016, pp. 525–542.

[76] Q. Deng et al., “DrAcc: A DRAM based accelerator for accurate CNN inference,” in *Proc. ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[77] S. Li et al., “SCOPE: A stochastic computing engine for DRAM-based in-situ accelerator,” in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Oct. 2018, pp. 696–709.

[78] L. Jiang et al., “XNOR-POP: A processing-in-memory architecture for binary convolutional neural networks in wide-IO2 DRAMs,” in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Jul. 2017, pp. 1–6.

[79] C. Eckert et al., “Neural cache: Bit-serial in-cache acceleration of deep neural networks,” in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 383–396.

[80] S. Angizi et al., “MRIMA: An MRAM-based in-memory accelerator,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 5, pp. 1123–1136, May 2020.

[81] Y. Pan et al., “A multilevel cell STT-MRAM-based computing in-memory accelerator for binary convolutional neural network,” *IEEE Trans. Magn.*, vol. 54, no. 11, pp. 1–5, Nov. 2018.

[82] S. Angizi et al., “CMP-PIM: An energy-efficient comparator-based processing-in-memory neural network accelerator,” in *Proc. 55th ACM/ESDA/IEEE Design Automat. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[83] Q. Deng et al., “LAcc: Exploiting lookup table-based fast and accurate vector multiplication in DRAM-based CNN accelerator,” in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, Jun. 2019, pp. 1–6.

[84] P. R. Sutradhar et al., “pPIM: A programmable processor-in-memory architecture with precision-scaling for deep learning,” *IEEE Comput. Archit. Lett.*, vol. 19, no. 2, Jun./Dec. 2020, Art. no. 118121.

[85] P. R. Sutradhar et al., “Look-up-table based processing-in-memory architecture with programmable precision-scalingfor deep learning applications,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 2, pp. 263–275, Feb. 2021.

[86] O. Terzo et al., *Heterogeneous Computing Architectures: Challenges and Vision*. Boca Raton, FL, USA: CRC Press, 2019.

[87] Q. Zhang et al., “ApproxANN: An approximate computing framework for artificial neural network,” in *Proc. Design Autom. Test Eur. Conf. Exhib. (DATE)*, 2015, pp. 701–706.

[88] Q. Zhang et al., “ApproxANN: An approximate computing framework for artificial neural network,” in *Proc. Design Autom. Test Eur. Conf. Exhib. (DATE)*, 2015, pp. 701–706.

[89] M. Samadi et al., “SAGE: Self-tuning approximation for graphics engines,” in *Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2013, pp. 13–24, doi: 10.1145/2540708.2540711.

[90] S. T. Chakradhar and A. Raghunathan, “Best-effort computing: Re-thinking parallel software and hardware,” in *Proc. 47th Design Automat. Conf. (DAC)*, 2010, pp. 865–870, doi: 10.1145/1837274.1837492.

[91] Q. Shi, H. Hoffmann, and O. Khan, “A cross-layer multicore architecture to tradeoff program accuracy and resilience overheads,” *IEEE Comput. Archit. Lett.*, vol. 14, no. 2, pp. 85–89, Jul./Dec. 2015.

[92] T. Moreau et al., “SNNAP: Approximate computing on programmable SoCS via neural acceleration,” in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2015, pp. 603–614.

[93] Z. Du et al., “Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators,” in *Proc. 19th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2014, pp. 201–206.

[94] Z. Peng et al., “AXNET: Approximate computing using an end-to-end trainable neural network,” 2018, *arXiv:1807.10458*.

[95] S. Ullah et al., “Area-optimized low-latency approximate multipliers for FPGA-based hardware accelerators,” in *Proc. 55th Annu. Design Autom. Conf.*, Jun. 2018, p. 159.

[96] B. S. Prabakaran et al., “DeMAS: An efficient design methodology for building approximate adders for FPGA-based systems,” in *Proc. Design Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 917–920.

[97] B. S. Prabakaran et al., “ApproxFPGAs: Embracing ASIC-based approximate arithmetic components for FPGA-based systems,” in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–6.

[98] H. Saadat, H. Bokhari, and S. Parameswaran, “Minimally biased multipliers for approximate integer and floating-point multiplication,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2623–2635, Nov. 2018.

[99] Z. E. Mamaghani, S. Ullah, and A. Kumar, “SIMDive: Approximate SIMD Soft multiplier-divider for FPGAs with tunable accuracy,” in *Proc. 30th ACM Great Lakes Symp. VLSI (GLSVLSI)*, Sep. 2020, pp. 151–156.

[100] B. Grigorian, N. Farahpour, and G. Reinman, “BRAINIAC: Bringing reliable accuracy into neurally-implemented approximate computing,” in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2015, pp. 615–626.

[101] A. U. Hassen and S. A. Khokhar, “Approximate in-memory computing on reram crossbars,” in *Proc. IEEE 62nd Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2019, pp. 1183–1186.

[102] H. Cai et al., “Pj-AxMTJ: Process-in-memory with joint magnetization switching for approximate computing in magnetic tunnel junction,” in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2019, pp. 111–115.

[103] Y. Tang et al., “ApproxPIM: Exploiting realistic 3D-stacked DRAM for energy-efficient processing in-memory,” in *Proc. 22nd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, 2017, pp. 396–401.

[104] D. S. Cali et al., “Accelerating approximate pattern matching with processing-in-memory (PIM) and single-instruction multiple-data (SIMD) programming,” Tech. Rep., 2018.

[105] H. E. Yantir, A. M. Eltawil, and F. J. Kurdahi, “Approximate memristive in-memory computing,” *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5s, Sep. pp. 1–8, 2017, doi: 10.1145/3126526.

[106] H. E. Yantir, A. M. Eltawil, and F. J. Kurdahi, “A hybrid approximate computing approach for associative in-memory processors,” *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 4, pp. 758–769, Jul. 2018.

[107] A. Sampson et al., “ACCEPT: A programmer-guided compiler framework for practical approximate computing,” Univ. Washington, Seattle, WA, USA, Tech. Rep. UW- CSE-15-01, Jan. 2015.

[108] X. Zhang et al., “DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2018, pp. 1–8.

[109] C. Hao et al., “FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge,” in *Proc. Design Autom. Conf.*, 2019, pp. 1–6.

[110] E. M. Panainte, K. Bertels, and S. Vassiliadis, “The Molen compiler for reconfigurable processors,” *ACM Trans. Embedded Comput. Syst.*, vol. 6, no. 1, p. 6, Feb. 2007, doi: 10.1145/1210268.1210274.

[111] Y. Shan et al., “FPMR: Mapreduce framework on FPGA,” in *Proc. FPGA*, 2010, pp. 93–102, doi: 10.1145/1723112.1723129.

[112] J. Wawrzynek et al., “RAMP: Research accelerator for multiple processors,” *IEEE Micro*, vol. 27, no. 2, pp. 46–57, Mar. 2007, doi: 10.1109/MM.2007.39.

[113] G. Zervakis, H. Amrouch, and J. Henkel, “Design automation of approximate circuits with runtime reconfigurable accuracy,” *IEEE Access*, vol. 8, pp. 53522–53538, Mar. 2020.

[114] J. Hormigo et al., “Self-reconfigurable constant multiplier for FPGA,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 6, no. 3, pp. 1–17, Oct. 2013.

[115] V. VranjkoviC and R. Struharik, “Coarse-grained reconfigurable hardware accelerator of machine learning classifiers,” in *Proc. Int. Conf. Syst., Signals Image Process. (IWSSIP)*, May 2016, pp. 1–5.

[116] C. Mead, “Neuromorphic electronic systems,” *Proc. IEEE*, vol. 78, no. 10, pp. 1629–1636, Oct. 1990.

[117] K. Roy et al., “In-memory computing in emerging memory technologies for machine learning: An overview,” in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.

[118] V. Saxena, “Mixed-signal neuromorphic computing circuits using hybrid CMOS-RRAM integration,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 2, pp. 581–586, Feb. 2020.

[119] B. S. Lee et al., “Nanoscale nuclei in phase change materials: Origin of different crystallization mechanisms of Ge2Sb2Te5 and AgInSbTe,” *J. Appl. Phys.*, vol. 115, no. 6, Feb. 2014, Art. no. 063506.

[120] A. Gholami et al., “A survey of quantization methods for efficient neural network inference,” 2021, *arXiv:2103.13630*.

[121] C. Merkel, “Current-mode memristor crossbars for neuromorphic computing,” in *Proc. 7th Annu. Neuro-Inspired Computat. Elements Workshop*, Mar. 2019, pp. 1–6.

[122] E. Fransen, “A synapse which can switch from inhibitory to excitatory and back,” *Neurocomputing*, vol. 65, pp. 39–45, Jun. 2005.

[123] C. Merkel and D. Kudithipudi, “A current-mode CMOS/memristor hybrid implementation of an extreme learning machine,” in *Proc. 24th Ed. Great Lakes Symp. VLSI*, May 2014, pp. 241–242.

[124] R. Hasan, T. M. Taha, and C. Yakopcic, “On-chip training of memristor crossbar based multi-layer neural networks,” *Microelectron. J.*, vol. 66, pp. 31–40, Aug. 2017.

[125] C. Li et al., “Analogue signal and image processing with large memristor crossbars,” *Nature Electron.*, vol. 1, no. 1, pp. 52–59, Jan. 2018.

[126] M. A. Zidan et al., “Memristor-based memory: The sneak paths problem and solutions,” *Microelectron. J.*, vol. 44, no. 2, pp. 176–183, 2013.

[127] A. Dozortsev, I. Goldshtein, and S. Kvatinsky, “Analysis of the row grounding technique in a memristor-based crossbar array,” *Int. J. Circuit Theory Appl.*, vol. 46, no. 1, pp. 122–137, 2018.

[128] M. A. Zidan et al., “Memristor multiport readout: A closed-form solution for sneak paths,” *IEEE Trans. Nanotechnol.*, vol. 13, no. 2, pp. 274–282, Jan. 2014.

[129] L. Shi et al., “Research progress on solutions to the sneak path issue in memristor crossbar arrays,” *Nanosc. Adv.*, vol. 2, no. 5, pp. 1811–1827, 2020.

[130] H. Mulaosmanovic et al., “Novel ferroelectric FET based synapse for neuromorphic systems,” in *Proc. Symp. VLSI Technol.*, 2017, pp. T176–T177.

[131] A. Jones et al., “A neuromorphic SLAM architecture using gated-memristive synapses,” *Neurocomputing*, vol. 381, pp. 89–104, Mar. 2020.

[132] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain.” *Psychol. Rev.*, vol. 65, no. 6, p. 386, 1958.

[133] P. Priyanka, G. Nisarga, and S. Raghuram, “CMOS implementations of rectified linear activation function,” in *Proc. Int. Symp. VLSI Design Test*, 2018, pp. 121–129.

[134] R. Zunino and P. Gastaldo, “Analog implementation of the SoftMax function,” in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, May 2002, p. 2.

[135] C. Mead, *Analog VLSI and Neural Systems*, 1st ed. Reading, MA, USA: Addison-Wesley, 1989.

[136] G. Indiveri et al., “Neuromorphic silicon neuron circuits,” *Frontiers Neurosci.*, vol. 5, p. 73, May 2011.

[137] C. Zhao et al., “Energy efficient spiking temporal encoder design for neuromorphic computing systems,” *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 2, no. 4, pp. 265–276, Oct./Dec. 2016.

[138] M. Al-Shedivat et al., “Inherently stochastic spiking neurons for probabilistic neural computation,” in *Proc. 7th Int. IEEE/EMBS Conf. Neural Eng. (NER)*, Apr. 2015, pp. 356–359.

[139] I. Sourikopoulos et al., “A 4-fJ/spike artificial neuron in 65 nm CMOS technology,” *Frontiers Neurosci.*, vol. 11, p. 123, Mar. 2017.

[140] C. Merkel and D. Kudithipudi, “Comparison of off-chip training methods for neuromemristive systems,” in *Proc. 28th Int. Conf. VLSI Design*, 2015, pp. 99–104.

[141] M. E. Fouda et al., “Effect of asymmetric nonlinearity dynamics in RRAMs on spiking neural network performance,” in *Proc. 53rd Asilomar Conf. Signals, Syst., Comput.*, 2019, pp. 495–499.

[142] I. Hubara et al., “Quantized neural networks: Training neural networks with low precision weights and activations,” *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, 2017.

[143] E. O. Neftci, H. Mostafa, and F. Zenke, “Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks,” *IEEE Signal Process. Mag.*, vol. 36, no. 6, pp. 51–63, Nov. 2019.

[144] V. Milo et al., “Demonstration of hybrid CMOS/RRAM neural networks with spike time/rate-dependent plasticity,” in *IEDM Tech. Dig.*, Dec. 2016, pp. 16–18.

[145] J. Darringer et al., “EDA in IBM: Past, present, and future,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 12, pp. 1476–1497, Jan. 2001.

[146] L. Stok, “The next 25 years in EDA: A cloudy future?” *IEEE Des. Test*, vol. 31, no. 2, pp. 40–46, Mar. 2014.

[147] R. A. Finkel and J. L. Bentley, “Quad trees a data structure for retrieval on composite keys.” *Acta Inform.*, vol. 4, pp. 1–9, Mar. 1974.

[148] C. L. Jackins and S. L. Tanimoto, “Quad-trees, Oct-trees, and K-trees: A generalized approach to recursive decomposition of Euclidean space,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-5, no. 5, pp. 533–539, Sep. 1983.

[149] J. L. Bentley, “K-d trees for semidynamic point sets,” in *Proc. Symp. Comput. Geometry (SCG)*, 1990, pp. 187–197.

[150] S. B. Akers, “Binary decision diagrams,” *IEEE Trans. Comput.*, vol. C-27, no. 6, pp. 509–516, Jun. 1978.

[151] G. Huang et al., “Machine learning for electronic design automation: A survey,” *ACM Trans. Des. Automat. Electron. Syst.*, vol. 26, no. 5, pp. 1–46, 2021.

[152] I. Lee, “An optimization approach to capacity evaluation and investment decision of hybrid cloud: A corporate customer’s perspective,” *J. Cloud Comput.*, vol. 8, Nov. 2019, Art. no. 15.

[153] S. Ambrogio et al., “Equivalent-accuracy accelerated neural-network training using analogue memory,” *Nature*, vol. 558, no. 7708, pp. 60–67, Jun. 2018, doi: 10.1038/s41586-018-0180-5.

[154] M. Talib et al., “A systematic literature review on hardware implementation of artificial intelligence algorithms,” *J. Supercomput.*, vol. 77, pp. 1897–1938, Feb. 2021.

[155] C. Zhuo, B. Yu, and D. Gao, "Accelerating chip design with machine learning: From pre-silicon to post-silicon," in *Proc. 30th IEEE Int. Syst.-Chip Conf. (SOCC)*, Sep. 2017, pp. 227–232.

[156] A. Liaw and M. Wiener, "Classification and regression by randomforest," *Forest*, vol. 23, pp. 18–22, Nov. 2001.

[157] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, Jul. 1989.

[158] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proc. 5th Annu. Workshop Comput. Learn. Theory (COLT)*, 1992, pp. 144–152, doi: 10.1145/130385.130401.

[159] E. Fix and J. L. Hodges, "Discriminatory analysis. Nonparametric discrimination: Consistency properties," *Int. Stat. Rev./Revue Internationale Statistique*, vol. 57, no. 3, pp. 238–247, 1989. [Online]. Available: http://www.jstor.org/stable/1403797

[160] J. Zhao et al., "Machine learning based routing congestion prediction in FPGA high-level synthesis," in *Proc. Design Autom. Test Eur. Conf. Exhib. (DATE)*, 2019, pp. 1130–1135.

[161] Z. Wang and B. C. Schafer, "Machine learning to set meta-heuristic specific parameters for high-level synthesis design space exploration," in *Proc. DAC*, Jul. 2020, pp. 1–6.

[162] D. Liu and B. Schafer, "Efficient and reliable high-level synthesis design space explorer for FPGAS," in *Proc. FPL*, Aug. 2016, pp. 1–8.

[163] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Computer Aided Verification*, T. Touili, B. Cook, and P. Jackson, Eds. Berlin, Germany: Springer, 2010, pp. 24–40.

[164] C. Yu, H. Xiao, and G. De Micheli, "Developing synthesis flows without human knowledge," in *Proc. 55th Annu. Design Automat. Conf.*, 2018, pp. 1–6, doi: 10.1145/3195970.3196026.

[165] S. I. Ward, D. Ding, and D. Pan, "Pade: A highperformance placer with automatic datapath extraction and evaluation through high-dimensional data learning," in *Proc. DAC Design Autom. Conf.*, 2012, pp. 756–761.

[166] Z. Xie et al., "RouteNet: Routability prediction for mixed-size designs using convolutional neural network," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Dec. 2018, pp. 1–8.

[167] A. Kahng, M. Luo, and S. Nath, "SI for free: Machine learning of interconnect coupling delay and transition effects," in *Proc. ACM/IEEE Int. Workshop Syst. Level Interconnect Predict. (SLIP)*, Jun. 2015, pp. 1–8.

[168] A. Mirhoseini et al., "Chip placement with deep reinforcement learning," 2020, *arXiv:2004.10746*.

[169] H. Wang et al., "GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *Proc. DAC*, Jul. 2020, pp. 1–6.

[170] J. D. et al., "Gaps and opportunities for AI/ML techniques in the EDA domain," in *Proc. Si*, Nov. 2020.

[171] L. Stok, "EDA 3.0: EDAaaS, EDA as a service," in *Proc. TAU*, 2015.

[172] A. Joseph, "NEXA: Cloud native platform for collaborative hardware logic design in step-wise refinement implementation flows," in *Proc. Design Automat. Conf.*, 2021.

[173] W. Li and M. Liewig, "A survey of AI accelerators for edge environment," in *Proc. World Conf. Inf. Syst. Technol.*, Jun. 2020, pp. 35–44.

[174] X. Sun et al., "Ultra-low precision 4-bit training of deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 1796–1807.

[175] W. Roesner, "Aspect-oriented design—Optimizing SOC verification via separation of concerns," in *Proc. Design Autom. Conf.*, 2014.

[176] W. Roesner, "Software methods meet large-scale system-on-a-chip design," in *Proc. TCE*, 2015.

[177] M. H. Safieddine et al., "Methodology for separation of design concerns using conservative RTL flipflop inference," in *Proc. DVCon*, 2015, pp. 1–8.

[178] M. Safieddine et al., "Verification at RTL using separation of design concerns," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 8, pp. 1529–1542, Aug. 2019.

Sathwika Bavikadi is pursuing a PhD in electrical and computer engineering with George Mason University (GMU), Fairfax, VA, USA. Bavikadi has an MSc in electrical engineering with emphasis on signal processing from the Blekinge Institute of Technology (BTH), Karlskrona, Sweden. She is a Student Member of IEEE.

Abhijitt Dhavlle is a Graduate Research Assistant, pursuing a PhD with George Mason University (GMU), Fairfax, VA, USA. His research interests are microarchitectural attacks, physical side-channel attacks, hardware-based malware detectors, and deep learning accelerators. Dhavlle has a master's in computer engineering from GMU. He is a Student Member of IEEE.

Amlan Ganguly is an Associate Professor and the Department Head of the Department of Computer Engineering at Rochester Institute of Technology, Rochester, NY, USA. His research interests are in robust and scalable intrachip and interchip interconnection architectures and novel datacenter networks with wireless interconnects as well as non-von Neumann architectures for data-intensive applications such as deep learning. Ganguly has an MS and a PhD from Washington State University, Pullman, WA, USA. He is a Senior Member of IEEE.

Anand Haridass is a Senior Principal Engineer with the Intel's Data Center and AI Group, Bengaluru, India. Haridass has an MSEE from Georgia Institute of Technology, Atlanta, GA, USA. He is a Senior Member of IEEE.

Hagar Hendy is pursuing a PhD in electrical and computer engineering with the Rochester Institute of Technology, Rochester, NY, USA. Her research focuses on neuromorphic circuit design based on memristor devices. Hendy has an MSc from the Faculty of Engineering, Ain Shams University, Cairo, Egypt, where her research focused on designing read/write circuits for multibit memristor memories.

Cory Merkel is an Assistant Professor with the Department of Computer Engineering, Rochester Institute of Technology (RIT), Rochester, NY, USA. His current research focuses on mapping of AI algorithms, primarily artificial neural networks, to mixed-signal hardware, design of brain-inspired computing systems using emerging technologies, and trustworthy neuromorphic systems. Merkel has an MS in computer engineering and a PhD in microsystems engineering from RIT. He is a Member of IEEE.

Vijay Janapa Reddi is an Associate Professor at Harvard University, Cambridge, MA, USA, and the Vice President and a Founding Member of MLCommons (mlcommons.org), a nonprofit organization aiming to accelerate machine learning (ML) innovation for everyone. His research sits at the intersection of ML, computer architecture, and runtime software. He is a Member of IEEE.

Purab Ranjan Sutradhar is pursuing a PhD in computer engineering with the Rochester Institute of Technology, Rochester, NY, USA. His research interests include data/memory-centric computing architectures and deep learning applications. Sutradhar has a Bachelor of Science in electrical and electronic engineering from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh. He is a Student Member of IEEE.

Arun Joseph is Senior Engineering Leader of the IBM India Hardware Development Laboratory, Hardware Electronic Design Automation Group, Bengaluru, India.

Sai Manoj Pudukotai Dinakarrao is an Assistant Professor with George Mason University, Fairfax, VA, USA. His research interests include on-chip hardware security, neuromorphic computing, adversarial machine learning, self-aware SoC design, image processing and time-series analysis, emerging memory devices, and heterogeneous integration techniques. Pudukotai Dinakarrao received a PhD in electrical and electronics engineering from Nanyang Technological University, Singapore. He is a Member of IEEE.

■ Direct questions and comments about this article to Sai Manoj Pudukotai Dinakarrao, Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA 22030 USA; spudukot@gmu.edu.