Approximate Computing and the Efficient Machine Learning Expedition

(Invited Paper)

Jörg Henkel
henkel@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

Hai Li
hai.li@duke.edu
Duke University
Durham, USA

Anand Raghunathan
raghunathan@purdue.edu
Purdue University
West Lafayette, USA

Mehdi B. Tahoori
mehdi.tahoori@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

Swagath Venkataramani
swagath.venkataramani@ibm.com
IBM Research
Yorktown Heights, USA

Xiaoxuan Yang
xy92@duke.edu
Duke University
Durham, USA

Georgios Zervakis*
georgios.zervakis@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

ABSTRACT

Approximate computing (AxC) has been long accepted as a design alternative for efficient system implementation at the cost of relaxed accuracy requirements. Despite the AxC research activities in various application domains, AxC thrived the past decade when it was applied in Machine Learning (ML). The by definition approximate notion of ML models but also the increased computational overheads associated with ML applications–that were effectively mitigated by corresponding approximations–led to a perfect matching and a fruitful synergy. AxC for AI/ML has transcended beyond academic prototypes. In this work, we enlighten the synergistic nature of AxC and ML and elucidate the impact of AxC in designing efficient ML systems. To that end, we present an overview and taxonomy of AxC for ML and use two descriptive application scenarios to demonstrate how AxC boosts the efficiency of ML systems.

CCS CONCEPTS

• **Hardware** → **Logic circuits**; **Analog and mixed-signal circuits**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Approximate Computing, In-memory, Machine Learning, Precision Scaling, Printed Electronics, Pruning, Quantization, Transformers

ACM Reference Format:
Jörg Henkel, Hai Li, Anand Raghunathan, Mehdi B. Tahoori, Swagath Venkataramani, Xiaoxuan Yang, and Georgios Zervakis. 2022. Approxi-mate Computing and the Efficient Machine Learning Expedition: (Invited Paper). In *Proceedings of International Conference on Computer-Aided Design (ICCAD 2022).* ACM, New York, NY, USA, 9 pages.

*Also with University of Patras.

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA

ACM ISBN 978-1-4503-9217-4/22/10.
https://doi.org/10.1145/3508352.3561105

1 INTRODUCTION

The failure of Dennard scaling led to the so-called dark silicon problem [30] and computer designers were forced to explore radical new approaches to sustain and further improve the efficiency of our computing systems. Several groundbreaking novelties came from the field of computing. Among these newly established computing paradigms, over the past decade, a phenomenal boom is observed in Approximate Computing (AxC) [60] research. Approximate computing refers to techniques that exploit the inherent error resilience of several applications to achieve improvements in efficiency (e.g., energy and performance) at all layers of the computing stack [60]. For example, prior analysis on a benchmark suite of 12 recognition, mining and search applications showed that 83% of the runtime is spent in tasks that are amenable to approximation [15, 60]. The origins of approximate computing (AxC) can be traced back to various fields including computer arithmetic (floating point representation) [63], arithmetic units (adders [54] and multipliers [80]), digital signal processing (filter design) [27], algorithms (approximation algorithms) [62], and networking (best-effort packet delivery) [9].

However, the field really took off in the past decade when AxC intersected with machine learning (ML) [3, 62, 65], which arguably provides the ideal workload for AxC due to several factors. Many ML tasks at some level reduce to a problem of function approximation, where the function is incompletely specified, allowing ample scope for AxC. Further, the very process used to create models (training) can be re-purposed to recover from any adverse effects of approximations. In particular, AxC and deep neural networks (DNNs)–in which the latest improvements in the accuracy came at the cost of a great increase in computational requirements– formed a perfect match [12, 51, 57, 79]. Due to this synergy, the field of AxC witnessed significant interest and growth, and expanded into new levels of the computing stack such as approximate hardware. AxC for ML (e.g., quantization [16], pruning [70], relaxed synchronization [42]) has already been adopted in practice and is a key enabler of the AI hardware roadmap for many companies (e.g., ultra-low precision [66] and analog hardware [33]).

In this work, we first present an end-to-end view of designing approximate computing systems for ML, spanning algorithms to

circuits, and discuss the efficiency benefits accrued from AxC, highlighting also current commercial hardware and software that employ AxC. Next, we present two notable examples of approximate ML systems. Section 3 focuses on in-memory-computing-based (IMC-based) approximation for the state-of-the-art Transformer models. Transformer has become a prominent neural network model for Neural Language Processing (NLP) applications, with outstanding results in neural machine translation, entity recognition, etc. Section 4 discusses the exploitation of approximate computing as a key enabler for the realization of ultra-resource constrained ML circuits and specifically of battery powered printed ML classifiers. Printed electronics form an extreme use-case of embedded ML and pose a very promising solution to enable computing in application domains untouchable by silicon-based systems. Concluding, we discuss some key challenges that need to be addressed to enable further growth and adoption of AxC, especially in ML.

2 AXC FOR ML: A TAXONOMY

Recognizing the opportunity for significant improvements in performance and compute efficiency, developing Approximate Computing (AxC) techniques for AI/ML applications has been an active area of research in the past decade. The techniques spanned the entire computing stack from algorithms to circuits. Figure 1 shows a taxonomy of the various approximate computing techniques targeting AI/ML applications. Not surprisingly, the most successful techniques are inherently cross-layered *i.e.,* multiple layers of the compute stack are co-designed to achieve maximum benefits from the approximations, while minimizing their impact on end-application output quality [62]. The following subsections describe the AxC design techniques at each level of the compute stack in more detail.

2.1 Algorithmic Approximations for AI/ML

The goal of approximation techniques at the algorithm level is to identify the degree to which computations and data of the application can be approximated thereby driving down the compute and memory requirements and heal the impact of approximations on end-application quality. The majority of algorithmic approximations require careful co-design with the hardware layers of the compute stack to maximally exploit the benefits, while some approximations can be realized without specialized hardware support.

Some of the popular approximation techniques for AI/ML applications are listed below:

Quantization or Precision Scaling. Amongst the different approximation techniques, quantization has proven to be most successful technique widely adopted by the industry in the context of both training and inference of AI/ML models. At a high level, quantization involves scaling or reducing the number of bits used to represent the weights and activations. Reducing the numeric bit-precision super-linearly improves compute efficiency (for example, a 16-bit multiplier is roughly ~4 times lower cost than a 32-bit multiplier), and linearly reduces memory footprint and bandwidth. Further, it preserves the regularity in the compute patterns prevalent in AI/ML models, which makes it amenable to be integrated within both general-purpose and accelerator-based designs.

The key challenge in quantization is to ensure that the loss in numeric precision does not impact functional accuracy. In the context of training, research efforts have successfully explored different 16-bit floating point (FP16) variants *viz.* IEEE-FP16 (1-5-10), BFloat (1-8-7) [67] and DLFloat (1-6-9) [1]. More recently, a new Hybrid-FP8 (HFP8) data format [56], which involves combining two different FP8 formats—(1,4,3) for activations and weights, and (1,5,2) for errors—was introduced to lower training requirements even further.

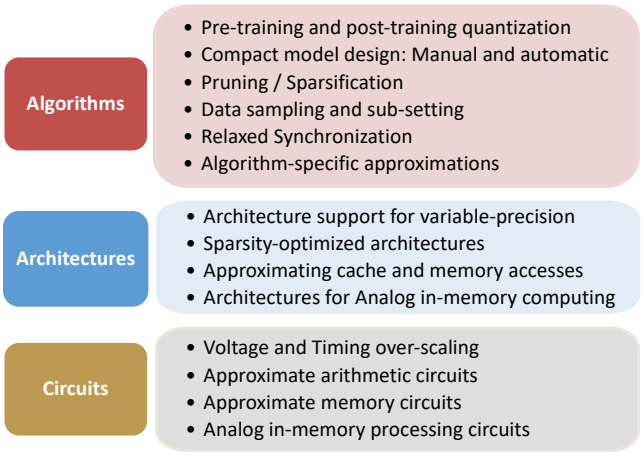

Figure 1: Taxonomy of Approximate Computing techniques for AI/ML across different levels of the compute stack

In the context of inference, the use of fixed-point (FxP) number representation is quite prevalent. There are two prevailing approaches to minimize the impact of using reduced-precision FxP for inference—post-training quantization (PTQ) and quantization-aware training (QAT). Post-training quantization involves directly turning a model trained with floating-point representation into a fixed-point model [83]. The loss in accuracy due to direct quantization can be minimized by re-calibrating batchnorm statistics. However, state-of-the-art PTQ techniques lose considerable model accuracy especially below 8-bits. Quantization-aware training (QAT) methods, first introduced in AxNN [63], retrain the model by reflecting the effect of quantization within the training process so that the weights can be suitably adapted to heal the impact of quantization. Popular QAT methods such as Parameterized Activation Clipping (PACT) and statistics-aware weight binning (SAWB) achieve state-of-the-art classification accuracy with INT4 precision, and incur only a small accuracy loss with INT2 across broad range of domains [17].

It is crucial to note that quantization is typically applied only to convolution (CONV) and matrix-multiply (GEMM) operations in an AI/ML model. While CONV/GEMM constitute the bulk of computations, other auxiliary operations for activation, pooling and normalization are typically performed in FP16 format. Also, in some cases, each layer of the AI/ML model is quantized to a different bit-precision.

Compact Model Design: Manual And Automatic. There is an urgent need to deploy AI/ML models in extremely resource-constrained mobile/IoT devices. One approach to facilitate this is to design compact AI/ML models that achieve near state-of-the-art accuracies as their large-scale counterparts. Popular neural network topologies such as inverted-bottleneck structures, depth-wise

seperable convolutions, group convolutions, significantly reduce the model size and consequently the number of computations. Recently, automatic methods to train compact models such as knowledge distillation [31] and Neural Architecture Search (NAS) [24] are gaining popularity.

Pruning / Sparsification and Compression. Another increasingly popular approach to reduce model size is *pruning*. It forces some weight values in a neural network to zero, thereby eliminating connections between associated neurons. Pruning sparsifies the weight tensors, which are then compactly stored and accessed using sparse representations (*e.g.,* index-value pairs). One of the key challenges in pruning is that it makes the compute and memory access pattern irregular and hence does not lend itself well into hardware acceleration. Recent approaches perform pruning in a structured fashion that preserves regularity [70]. One popular example is *fine-grained block sparsity*, wherein the weight tensor is split into equi-sized element blocks and each block is restricted to have a fixed percentage of non-zero elements [48]. Akin to quantization, pruning is performed at training time in order to heal its impact on accuracy.

In the context of training, which is typically performed in a distributed system, one of the key bottlenecks to overall throughput is off-chip communication. Training is parallelized by splitting the minibatch inputs across multiple learners and their respective weight gradients need to be accumulated to compute the updated weight value. To reduce off-chip communication cost, the weight gradient values are compressed using a number of interesting algorithms [11] that have achieved over >100× compression rates without loosing model accuracy.

Data Sampling and sub-setting. Data sampling is a technique where only a subset of elements of a tensor is used for computations. Sampling reduces the overall tensor footprint, which enables it to be contained within the lower levels of the memory hierarchy, alleviating performance bottlenecks due to memory latency and bandwidth limitations. Sampling techniques [38] typically exploit *value similarity* between data-elements present in a region. For example, in real-world images, adjacent pixels take similar values. In some cases, sampling can result in some of the computations to be redundant, which can be eliminated.

Relaxed Synchronization. Relaxed synchronization is a technique which is applied in the context of distributed training. Broadly, training algorithms require the learners to be synchronized at the end of each minibatch iteration so that model weights are updated using the weight gradients from each learner. Relaxed synchronization allows the learners to proceed semi-synchronously or even asynchronously *i.e.,* fast learners do not wait for slow learners to complete but proceed immediately to the next iteration [82]. The key trade-off is to ensure that this does not result in requiring additional training epochs for model convergence.

2.2 Approximate AI/ML Architectures

AI/ML applications find utility under a variety of deployment scenarios from cloud to the edge, with differing latency, throughput, and energy requirements. Hardware specialization/acceleration is regarded critical to meet the computational demands of AI/ML models, while a bulk of AI/ML applications are still executed on general-purpose CPUs owing to cost and form-factor constraints. Hence, approximations identified at the algorithm level have to judiciously incorporated within a spectrum of compute architectures from general-purpose processors to application-specific designs to maximally exploit the benefits from approximate computing.

One of the key tenets for approximate architecture design is that approximable components should dominate the overall energy consumption. Inherent control front-ends for instruction fetch/decode *etc.* needs to be executed in an accurate manner. Hardware accelerators for AI/ML contain arrays of processing elements which exploit the abundant data parallelism available in these applications with minimal control components. The execution engines dominate their overall energy consumption, and hence AxC computing techniques can be effectively employed to scale their performance and energy. Some of the design techniques at the architecture level include:

Architecture Support for Variable Precision. As mentioned earlier, the precision to which the data elements can be quantized varies widely across application domains, deployment scenarios and across layers of the model. This necessitates AI/ML architectures to design execution engines with mixed-precision support, whose accuracy levels are directly controlled by the software [61]. A key challenge in embodying mixed-precision support lies in identifying the right degree of logic sharing across the different precisions. Too much sharing would lead to poor energy efficiency at any given precision, whereas a completely decoupled execution pipeline for each precision would lead to significant area and leakage power overheads. A variable-precision AI accelerator design is the RaPiD architecture [66], fabricated at 7nm technology, that supports precisions ranging from 16-bit floating point to 2-bit fixed point, scaling peak compute capability from 8 TFLOPs to ∼200 TOPS.

Sparsity-optimized Architectures. The model pruning approximation triggers the need for architectures that can exploit sparsity in data to benefit performance and energy. Sparsity also naturally occurs in activations due to the use of ReLU activation function, particularly in the vision domain. Sparse accelerator architectures [28, 46] under a variety of scenarios—sparsity in one or both operands (weights and/or activations), support for structured *vs.* unstructured sparsity—have been explored in literature. The interplay between sparsity and quantization is an open challenge. The overheads due to sparse execution are amplified as the execution engines shrink with quantization favoring dense computation arrays. In addition to specialized architectures, sparsity support has also been explored in the context of general-purpose processors [52]. The key idea is to dynamically pre-detect and skip a set of future instructions that are rendered ineffectual due to sparsity.

Approximating Cache and Memory Accesses. The memory sub-system is often a critical bottleneck to performance and energy efficiency. Techniques such as quantization and pruning naturally benefit both compute and memory, but more in favor of compute due to super-linear reduction in multiple cost and overheads of sparse index representations. This leaves the memory sub-system still as the bottleneck. Addressing this challenge, specific approximation techniques targeting the memory sub-system have been proposed. These include reducing DRAM refresh rates [41], speculating on the results of loads [43], redirecting accesses to data already available at a memory level [38], among others.

Architectures for Analog In-Memory Computing. In-memory computing is an exciting class of architectures actively explored in recent years for AI/ML applications. It attempts to overcome the classical von-neumann bottleneck inherent in any dataflow architecture that transfer data between the compute and memory elements. This is achieved by deeply embedding the compute with the memory array itself. In-memory computing is generally based on analog processing and has been explored in the context of SRAM [36] and non-voltatile memories such as ReRAM, MRAM, or PCM [8]. Section 3 presents a detailed case study outlining an in-memory accelerator architecture for Transformer models.

2.3 Approximation Circuit Design

Approximate circuits are the basic building block of any approximate computing system. Techniques such as quantization and other number representations for AI/ML rely on efficient approximate circuit design to leverage maximum benefits. From the bottom-up, approximations at the circuit-level can be classified as: (i) logic approximation, wherein the logic functionality of a circuit is modified slightly to realize a disproportionately efficient implementation [64], (ii) timing approximation, wherein the circuit is designed and operated at an over-scaled voltage-frequency point that benefits efficiency while introducing modest timing errors [62]. Prior research also focused on designing approximate memory circuits whose access accuracy can be scaled dynamically at execution time [50].

In summary, the special synergy between AxC and AI/ML has enabled unprecedented levels of compute efficiency through systematic co-design across the compute stack. AxC for AI/ML has transcended beyond academic prototypes. Commercial products—CPUs, GPUs and accelerators—targeting AI/ML applications embody AxC techniques such as low precision and fine-grained structured sparsity. AxC forms an integral part of the AI/ML roadmap for many companies.

3 TRANSFORMERS ON IN-MEMORY ACCELERATORS USE CASE

In-memory computing (IMC) designs can eliminate the cost of data transfer since the computation is performed within the memory device. And in-memory computing can be efficiently implemented using emerging technologies. For instance, resistive random-access memory (ReRAM) is one of the most promising emerging technologies and has potentials to perform vector-matrix multiplications compared with CMOS accelerators [72]. Prior ReRAM-based processing-in-memory (PIM) works have demonstrated their potentials in improving the efficiency of neural networks training and inference, such as Brain-State-in-a-Box (BSB) model, deep convolutional neural network, and generative adversarial networks (GAN) [13, 34, 55]. Note that these in-memory computing designs are essential approximate computing cases. In this section, we discuss the general approximate in-memory computing and approximate in-memory computing in Transformer-based models.

3.1 General Approximate IMC

The analog basic of in-memory computing matches the concept of approximate computing. Take the ReRAM crossbar-based computation as an example, the weights are programmed as the conductance of ReRAM cells. Meanwhile, the voltage, which encodes the input information, is supplied to the word lines. Here, digital-to-analog and analog-to-digital converters are utilized to assist the signal conversion between analog and digital domains. When converting the weight matrix and input into the conductance matrix and voltage, they use fixed-point encoding with a limited bit precision to represent computation elements [74]. According to Kirchhoff's circuit law and Ohm's law, the output currents accumulated through the bit lines represent the matrix-vector multiplication results. In this case, the analog values stored on the device or supplied to it approximate the values to be computed, and the analog results generated by the crossbar approximate the final results. Adjusting the data representation precision can tradeoff the accuracy result and computing efficiency [55]. However, due to the inherent error tolerant ability of neural networks, limited-precision data representation will not degrade the accuracy performance [35].

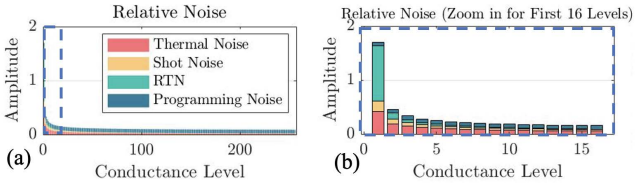

Figure 2: The distributions of stochastic noise for 8-bit ReRAM cells under $Freq = 500\,\text{MHz}$ and $T = 350\,\text{K}$: (a) in a relative scale, (b) zoomed-in version for relative noise.

Furthermore, the nonidealities in the IMC system lead the computation to be inaccurate [8]. In the programming stage, the voltage applied to the electrode layers changes the resistance status of the ReRAM cell. To be more specific, when a voltage is supplied to the ReRAM cell, a conductive filament composed of oxygen vacancies may form. Note that the randomness of generating oxygen vacancies is the primary cause of process variation. The geometry of oxygen vacancies is unexpected under the same level of voltage and equivalent circumstances, resulting in a variance in resistance levels between cycles.

There are statistical models to mimic the influence of IMC nonidealities. Variation in the resistance of the crossbar array obeys the lognormal distribution [32, 40], and the reason for this lognormal distribution is the Gaussian distribution of the average gap distances [76]. Aside from the variation, the IMC with ReRAM also suffers from thermal noise, shot noise, and random telegraph noise (RTN) [25]. For detailed noise distributions and analysis, we refer interested readers to [29, 74].

The stochastic noise for 8-bit ReRAM cell is shown in Figure 2. The relative noise is calculated with the noise amplitude divided by the absolute conductance in each cell. In Figure 2(a), the relative noise of small conductance level is greater than that of large conductance level, which indicates that the stochastic noise has a larger impact on small conductance values. As shown in Figure 2(b), RTN and thermal noise dominate the stochastic noise at small conductance levels. As the thermal noise increases with the temperature (T) and operating frequency ($Freq$), the influence of stochastic noise will be amplified when the IMC system is running at high temperature and high frequency.

The cell variation can accumulate through the accumulation via each column, and thus the variation of different computing results may overlap [14]. This overlapping variation problem will be severe if either the number of word lines or the number of low-resistance cells gets larger [37]. Therefore, the possible approaches to minimize the calculation variation are reducing the number of word lines calculated in each cycle and increasing the number of high-resistance cells within a specific rounding distance. And the IMC-based approximate computing by adjusting the number of word lines will tradeoff between performance and accuracy.

To tackle the problem of variation and noise in IMC, circuit-level designs with feedback loops can effectively reduce the variation influence. For instance, a closed-loop circuit design utilizes the inferencing results to stabilize the conductance values on the device [71]. Under this circumstance, the conductance values programmed onto the device are slightly different from the pure-software training weights because these values are determined with the integration of the real-device variation. Following the hardware-aware strategy, the noise injection adaption framework adjusts the conductance value by taking randomly sampled noise into account [29]. Similarly, the stochastic-noise-aware training method can mitigate the intrinsic noise in IMC system and improve the inferencing accuracy under high-frequency and high-temperature settings [74].

3.2 Approximate IMC in Transformer-based Models

Transformer has become a prominent deep neural network model with outstanding results in neural machine translation, entity recognition, etc [59]. Transformer-based pre-trained models, such as Bert [21], have served as the backbone of NLP applications. The Transformer model consists of an encoder stack and a decoder stack, which shares similar self-attention-based structure. Figure 3 shows essential components in Transformer model. The major challenge of utilizing IMC for efficient Transformer model is that the multi-head self-attention block brings computation and programming dependency and severe latency overhead. In prior works of improving the computing efficiency with PIM inference, they utilize the weight-stationary approach and don't have to reprogram the weights during the process [53]. However, the self-attention block requires the computation of two intermediate results, such as the Query $\mathcal{Q}$ and Key $\mathcal{K}$, as shown in Figure 3(b). Therefore, performing the matrix-matrix multiplication (MatMul) of these intermediate results with PIM will involve the programming of one intermediate result matrix onto the memory device. The computation phase has to be paused until the row-by-row matrix programming is completed. Hence, the computation efficiency will be compromised due to the compute-after-write dependency [73].

Therefore, it is necessary to design IMC architecture to improve the computation efficiency of multi-head self-attention block. ReTransformer is a ReRAM-based IMC design for Transformer acceleration [73]. This design can accelerate the scaled dot-product attention of Transformer and eliminate data dependency by avoiding writing the intermediate results using the matrix decomposition technique. For instance, Key $\mathcal{K}$ is the intermediate result from the computation of $\mathcal{K} = X \cdot W_K$, where X is the input and W_K is the weight matrix related to Key. Instead of computing with Key $\mathcal{K}$, the

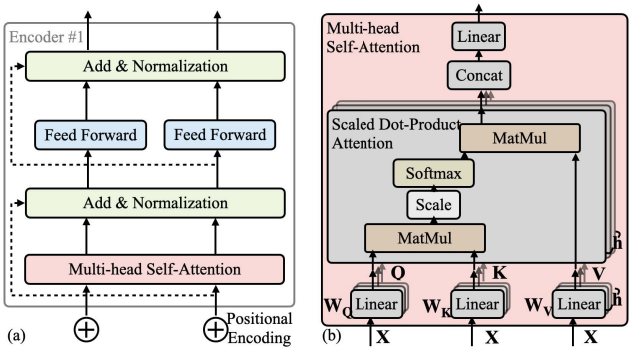

Figure 3: Transformer model structure: (a) encoder structure, (b) multi-head self-attention module.

Query $\mathcal{Q}$ computes with the W_K and X successively. Each element in weight matrix is stored as an 8-bit number in four ReRAM cells, where each cell represents two bits. And the quantization-aware training can help maintain the inference accuracy of Transformer models and encoder-based Bert models [49, 77]. Moreover, a hybrid computing unit for softmax computation is proposed, and a look-up table (LUT) is used for efficient exponential computation. ReTransformer with 8-bit data representation improves computing efficiency by 23.21× compared with GPU baseline.

It is also worth noting that the normalization and activation calculations in Transformer involve nonlinear computations. To accelerate these computations, recent work NN-LUT proposes LUT-based neural network approximation to mimic the nonlinear computation with piece-wise linear computation [75]. In addition, the CORDIC-based iterative method can also approximate the nonlinear computation and be implemented efficiently with IMC [58]. The approximate computations can boost the computation efficiency without the sacrification of computing accuracy.

4 PRINTED ML CIRCUITS, AN ULTRA RESOURCE CONSTRAINED USE CASE

In this section, we use printed electronics as a use case example and investigate approximate printed ML circuits. More specifically, we analyze how approximate computing can eventually enable, for the first time, the realization of battery-powered, high-accuracy, complex printed ML classifiers. It is noteworthy that printed electronics form an ultra resource constrained environment and constitute an extreme scenario of embedded ML application.

4.1 Printed Electronics Background

While the Moore's law has been the guiding force for the progress of lithography-based silicon Very Large Scale Integration (VLSI) technologies for higher integration density, these technologies have a lower bound on costs due to high costs of manufacturing (e.g., wafer processing, lithography, and material processing), which, in turn, increase the costs of packaging, testing, and assembly.

An alternative manufacturing scheme, the so-called printed electronics, based on low-cost additive manufacturing technologies is a promising way to target disposables and ultra-low cost margin domains, especially with conformality needs. Printing technologies

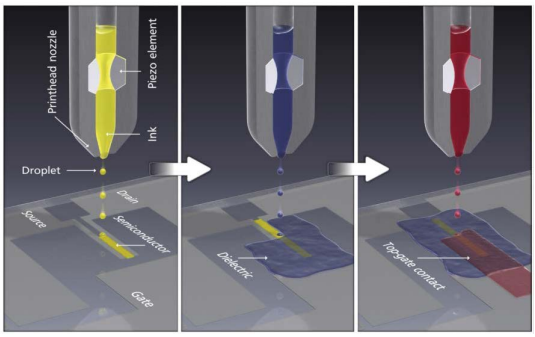

Figure 4: Inkjet printing process

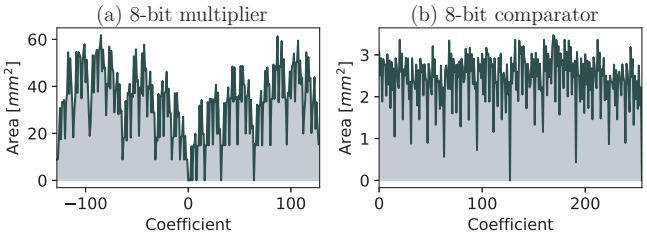

Figure 5: Area variation of a bespoke (a) multiplier and (b) comparator with respect to the coefficient value [4, 5]

.

often rely on mask-less, portable, and additive manufacturing methods which can greatly reduce costs and production timelines [10]. Printed electronics refers to the fabrication technology that relies on printing processes, such as jet printing, screen or gravure printing [19]. Figure 4 shows the inkjet printing process of an electrolyte gated field-effect transistor (EGFET).

Printed electronics technologies are broadly divided into two categories: 1) *additive manufacturing* process in which only deposition steps are involved, and 2) *subtractive process* involving a series of additive (deposition) and subtractive (etching) processes, similar to the silicon-based processing. The low equipment costs along with the simple additive manufacturing enable ultra low-cost (even subcent) electronic circuits. Nevertheless, printed electronics cannot match the area and performance characteristics of silicon systems. The large feature sizes in printed electronics result in high device latencies and low integration density (orders of magnitude lower than in silicon VLSI). However, the applications in target domains typically have extremely low performance and precision requirements (e.g., sampling rate of few Hz and few bits precision) which may be met by printing technologies under acceptable area and energy constraints. Printed systems have been successfully fabricated, such as boolean logic [18] and a 32-bit Arm microprocessor [6].

Due to its unique features, printed electronics constitutes a most-promising solution to eventually enable computing and bring intelligence in applications domains, that have not witnessed yet considerable penetration of computing particularly in the 10-trillion market of fast moving consumer goods (FMCG) [39] such as smart packaging, low-end healthcare products, and disposables, to name a few. As a result, designing printed ML circuits has become the center of many research activities [22, 44, 45, 68] with remarkable results. Nevertheless, despite the high efforts and the significant achievements reported, implementing complex printed ML circuits is still questionable through conventional computing. The large hardware overheads in printed electronics mandate a higher degree of optimization that can be achieved only through alternative computing methods [4, 5, 69].

4.2 Approximate Bespoke ML Classifiers

The potential for large customization, that originates by the non-recurring engineering (NRE) cost and the low-cost in-situ and on-demand fabrication–even at low to moderate fabrication volumes–of printed electronics, enables bespoke implementations [7, 44].

In such bespoke circuits, the coefficients of the ML models are hardwired in the circuit implementation itself, leading to orders of magnitude lower hardware overheads (e.g., area, power) compared to conventional baseline implementations [7]. Exploiting the efficiency of a bespoke implementation and coupling it with approximate computing principles further reduces the hardware requirements paving the way, for the first time, towards complex, battery powered printed ML circuits [4, 5]. It is noteworthy that bespoke circuits enable new types of approximation, allowing for more customized optimization, dedicated to each model. In bespoke ML circuits the hardware overheads are highly correlated with the values of the coefficients of the model. In other words, models with the same architecture but different coefficients will feature different hardware requirements. For example, Figure 5 depicts the area overheads of 8-bit bespoke multipliers (Figure 5(a)) and 8-bit comparators (Figure 5(b)) for all coefficient values. Note that multipliers and comparators constitute basic building blocks of most ML algorithms. It is observed, in Figure 5, that different coefficient values result in significantly different area. As a result, leveraging this property, by systematically approximating the coefficient values with hardware friendly ones, high area and consequently power gains (even up to 100%) can be achieved. In [4, 5] hardware-aware coefficient approximation is combined with traditional approximation techniques such as netlist pruning [78] and precision scaling.

Support Vector Machines (SVMs) and MultiLayer Perceptrons (MLPs) are targeted in [4]. The core function of both SVMs and MLPs is to calculate a weighted sum. Hence, by replacing each coefficient (in each neuron or classifier) with a more hardware-friendly value (e.g., from Figure 5(a)) so that the positive errors (replace with smaller value) and the negative errors (replace with larger value) cancel out each other, high area savings can be achieved for a minimal accuracy loss. This approximation is performed at the software level and the newly obtained model is used to generate the bespoke circuit. Precision scaling is also leveraged to reduce the size of the required arithmetic operators. Using only 4 bits for the inputs and 8 bits for the weights proved to barely impact the accuracy of all the models examined, delivering almost identical results to floating-point inference. Then, exploiting the fact that a significant portion of a circuit's gates switch very rarely, applying gate-level pruning at the hardware level (in which such gates are replaced by a constant 0 or 1) further increases the area gains. Table 1 presents the hardware overheads of the exact and approximate classifiers when targeting a marginal accuracy loss of 1%. The datasets for training the models are obtained from the UCI ML

Table 1: Area and Power evaluation of approximate MLPs/SVMs for up to 1% accuracy loss [4].

ML Circuit	Exact Bespoke			Coef. Approx. & Gate Prune			
	Ac[1]	A[2]	P[3]	A[2]	P[3]	AG[4]	PG[5]
Cardio MLP-C	0.88	33	97	17	54	48%	44%
RedWine MLP-C	0.56	18	53	8.0	27	55%	50%
WhiteWine MLP-R	0.53	13.1	40.7	8.0	27	39%	34%
RedWine MLP-R	0.56	7.1	24	3.3	12	53%	49%
RedWine SVM-C	0.57	24	73	7.6	26	68%	65%
Cardio SVM-C	0.90	15	47	8.7	29	43%	38%

[1]Accuracy. [2]Area (cm^2). [3]Power (mW). [4]Area and [5]Power Gain compared to the bespoke baseline.

repository [23]. As shown, for all the examined models, employing approximate computing delivers large area (above 39%) and power (above 34%) gains. What is even more important is that most of the approximated SVMs and MLPs can be powered by a printed battery (e.g., a Molex 30mW). However, this is not the case for the exact bespoke baseline circuits. Among the latter, only the RedWine MLP-R features adequate power consumption to be battery-powered.

Similarly, printed Decision Trees (DTs) are examined in [5]. Again, the authors exploit the area variation of the comparators with respect to the coefficient (threshold) value (Figure 5(b)) and implement a software-based coefficient approximation. In addition, the circuit is further approximated/optimized by applying precision scaling at the input port of each bespoke comparator. However, unlike [4], the error due to coefficient approximation is not linear and its impact is hard to predict. Therefore, in order to explore the space and identify for each comparator in the DT, the respective approximation configuration (i.e., input precision and value for the coefficient approximation), a non-dominated sorting genetic algorithm (NSGA-II) is used [5]. To guide the genetic algorithm to select more area efficient solutions, the accumulated area of the required comparators is used as a proxy of the DT's area. Table 2 presents the accuracy, area, and power evaluation of the exact and approximate DTs. Again, the datasets are obtained from the UCI ML repository [23]. As shown in Table 2, the approximate DTs achieve more than 47% area and more than 48% power gains. Most importantly, all the approximate printed DTs can be powered by a small Blue Spark printed battery since they feature less than 3mW power consumption. In addition, the approximate Seeds can also be self-powered by an energy harvester as its power consumption is less than 0.1mW. Finally note that the area of the majority of the approximate DTs is well constrained and as a result they are perfect candidates for printed applications.

5 LESSONS LEARNED AND OPEN ISSUES

Approximate computing has been considered in a variety of application domains as an effective alternative in building efficient computing systems. Though, approximate computing exploitation really took off in ML. As widely demonstrated in the state of the art [3] and presented by several use cases in this paper, approximate computing and ML form a perfect synergy and their matching not only feels natural, but it seems mandatory in order to address the increased computing challenged in ML systems. ML models which try to approximate and learn the functions, naturally fit with the concept of approximate computing which, under bounded accuracy

Table 2: Accuracy, Area, and Power evaluation of approximate Decision Trees [5].

ML Circuit	Exact Bespoke			Coef. Approx. & Precision Scaling				
	Ac[1]	A[2]	P[3]	Ac[1]	A[2]	P[3]	AG[4]	PG[5]
Arrhythmia	0.56	163	7.6	0.67	22	1.04	86%	86%
Balance	0.75	68	3.1	0.81	27	1.16	60%	63%
HAR	0.84	551	26.1	0.83	295	13.7	47%	48%
Mammographic	0.76	99	4.5	0.81	8.06	0.38	92%	92%
Seeds	0.89	30	1.4	0.94	2.32	0.09	92%	94%
Vertebral	0.85	58	2.7	0.86	7.84	0.38	86%	86%

[1]Accuracy. [2]Area (cm^2). [3]Power (mW). [4]Area and [5]Power Gain compared to the bespoke baseline.

impacts, try to further reduce the hardware footprint (area, delay, power) [63, 66] or even address thermal constraints [2, 81] and/or defend against advertasial attacks [26, 47]

Among a considerable amount of varying approximations for ML that have been investigated, precision scaling establishes as the most prominent one, leading to an actual success story. ML models feature such a high degree of resilience that can tolerate very aggressive precision scaling, down to only a few bits [17]. It is noteworthy that many commercial ML and DNN accelerators, such as GPUs, FPGAs, and TPUs, have already adopted precision scaling (e.g., from floating point to low precision fixed point arithmetic) as well as quantized neural networks (QNNs) have become a norm [65, 66]. This further highlights the fact that some of the core concepts of approximate computing have been well embraced by the ML community. One of the reasons that precision scaling shined in ML is that it features a straightforward design and exploitation and preserves the regularity of compute. In other words, it delivers very high (and highly correlated) gains in both the compute, the memory, and the data transfer, not affecting, thus, the application's bottleneck. Moreover, it maintains the level of design abstraction and the link between software and hardware is apparent, leading also to straightforward quantification of the induced error. On the other hand, this is not the case for other approximation techniques (e.g., logic) in which the impact of approximation is irregular and require extremely slow hardware emulation [20] or circuit simulation [57]. Finally, as discussed in Section 2, to exploit the full potential of approximation, proper hardware support is required (e.g., architecture support for variable precision and/or sparsity). On the other hand, this is not the case in bespoke implementations, as for example in printed ML circuits and fully parallel FPGA architectures. Bespoke architectures (Section 4) exploit out-of-the-box all the benefits that may originate from low precision inputs/weights and pruning. In bespoke ML classifiers, unstructured pruning will directly result in eliminating the respective multipliers and reducing the size and potentially the precision of the accumulation tree. Moreover, low-precision inputs or small-value weights will have the same impact, i.e., significantly reduced hardware overheads for multiplications and additions. Hence, such features can be additionally exploited for further approximate ML models in customized hardware realizations.

Still, despite the early emplacement of approximate computing in ML, there are still several key challenges that need to be addressed in order to enable further its growth and adoption. One of the major

open research challenges of approximate computing, particularly when combined with machine learning, is the verification, i.e., how to quantify and bound the accuracy loss due to approximation, particularly in the context of mission critical tasks and applications. Related to verification challenges, it is important to quantify (and prove) the extent of uncertainty and inaccuracy that is imposed by an approximate computing technique for a specific model since approximation adds another degree of inaccuracy on top of ML models which are inherently inexact. An open challenge is to understand what types of approximations can still be utilized, especially for critical applications, and what are the limits of approximate computing in such domains. Moreover, the impact of approximate computing on the robustness of the ML model is merely explored. Approximate techniques aim mainly in improving performance and/or reducing energy by removing, however, redundancy from the ML model and/or by inducing noise in the computations (e.g., approximate units). A reasonable assumption would be that the model becomes less robust and, for example, more vulnerable to memory errors. Nevertheless, this trade-off hasn't been comprehensively analyzed yet. Finally, a key challenge refers to the design and optimization of approximate ML accelerators. While approximate computing can further simplify design closure and reduce hardware footprint, one challenge is to ensure that this does not lead to an explosion of the design space since now the choice of approximation adds additional decision dimensions. In addition, re-training/approximation-aware training/quantization-aware training are widely and mainly used to mitigate the accuracy degradation due to approximation. Still, re-training (especially in DNNs) can be time consuming, increasing the design cycle time and exacerbating the complexity of the associated design space exploration. Overall, how to approximate the design choices for approximate computing seem to be a very promising research direction in design automation for approximate computing in the context of machine learning.

ACKNOWLEDGMENTS

This work is partially supported by the German Research Foundation (DFG) through the project "ACCROSS: Approximate Computing aCROss the System Stack" HE 2343/16-1 and by grant from the Excellence Initiative of Karlsruhe Institute of Technology under Future Field program "SoftNeuro". This work is partially supported by US NSF 1955196, 2112562, 1910299, 1937435, and ARO W911NF-19-2-0107.

REFERENCES

[1] Ankur Agrawal et al. 2019. Dlfloat: A 16-b floating point format designed for deep learning training and inference. In *26th IEEE Symposium on Computer Arithmetic, ARITH 2019, Kyoto, Japan, June 10-12, 2019*, 92–95.

[2] Hussam Amrouch, Georgios Zervakis, Sami Salamin, Hammam Kattan, Iraklis Anagnostopoulos, and Jörg Henkel. 2020. Npu thermal management. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 39, 11, 3842–3855.

[3] Giorgos Armeniakos, Georgios Zervakis, Dimitrios Soudris, and Jörg Henkel. 2022. Hardware approximate techniques for deep neural network accelerators: a survey. *ACM Comput. Surv.*, (Mar. 2022).

[4] Giorgos Armeniakos, Georgios Zervakis, Dimitrios Soudris, Mehdi B. Tahoori, and Jörg Henkel. 2022. Cross-layer approximation for printed machine learning circuits. In *Design, Automation Test in Europe Conference Exhibition*, 190–195.

[5] Konstantinos Balaskas, Georgios Zervakis, Kostas Siozios, Mehdi B. Tahoori, and Jörg Henkel. 2022. Approximate decision trees for machine learning classification on tiny printed circuits. In *Int. Symp. Quality Electronic Design (ISQED)*.

[6] John Biggs et al. 2021. A natively flexible 32-bit arm microprocessor. *Nature*, 595, (July 2021), 532–536.

[7] Nathaniel Bleier, Muhammad Husnain Mubarik, Farhan Rasheed, Jasmin Aghassi-Hagmann, Mehdi B Tahoori, and Rakesh Kumar. 2020. Printed microprocessors. In *Annu. Int. Symp. Computer Architecture (ISCA)*. (June 2020), 213–226.

[8] Indranil Chakraborty et al. 2020. Resistive crossbars as approximate hardware building blocks for machine learning: opportunities and challenges. *Proceedings of the IEEE*, 108, 12, 2276–2310.

[9] Srimat T. Chakradhar and Anand Raghunathan. 2010. Best-effort computing: re-thinking parallel software and hardware. In *Design Automation Conference*.

[10] Joseph S Chang, Antonio F Facchetti, and Robert Reuss. 2017. A circuits and systems perspective of organic/printed electronics: review, challenges, and contemporary and emerging design approaches. *IEEE Journal on emerging and selected topics in circuits and systems*, 7, 1, 7–26.

[11] Chia-Yu Chen, Jungwook Choi, Daniel Brand, Ankur Agrawal, Wei Zhang, and Kailash Gopalakrishnan. 2017. Adacomp : adaptive residual gradient compression for data-parallel distributed training. (2017). arXiv: 1712.02679 [cs.LG].

[12] Chia-Yu Chen, Jungwook Choi, Kailash Gopalakrishnan, Viji Srinivasan, and Swagath Venkataramani. 2018. Exploiting approximate computing for deep learning acceleration. In *Design, Automation Test in Europe Conference Exhibition*, 821–826.

[13] Fan Chen, Linghao Song, and Yiran Chen. 2018. Regan: A pipelined reram-based accelerator for generative adversarial networks. In *Asia and South Pacific Design Automation Conference, ASP-DAC*. IEEE, 178–183.

[14] Wei-Hao Chen et al. 2018. A 65nm 1mb nonvolatile computing-in-memory reram macro with sub-16ns multiply-and-accumulate for binary dnn ai edge processors. In *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 494–496.

[15] Vinay K. Chippa, Srimat T. Chakradhar, Kaushik Roy, and Anand Raghunathan. 2013. Analysis and characterization of inherent application resilience for approximate computing. In *Design Automation Conference*, 9 pages.

[16] Jungwook Choi, Swagath Venkataramani, Vijayalakshmi (Viji) Srinivasan, Kailash Gopalakrishnan, Zhuo Wang, and Pierce Chuang. 2019. Accurate and efficient 2-bit quantized neural networks. In *Proceedings of Machine Learning and Systems*. A. Talwalkar, V. Smith, and M. Zaharia, (Eds.) Vol. 1, 348–359.

[17] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. Pact: parameterized clipping activation for quantized neural networks. (2018). arXiv: 1805.06085.

[18] Silvia Conti et al. 2020. Low-voltage 2d materials-based printed field-effect transistors for integrated digital and analog electronics on paper. *Nature communications*, 11, 1, 1–9.

[19] Zheng Cui. 2016. *Printed electronics: materials, technologies and applications*. John Wiley & Sons.

[20] Dimitrios Danopoulos, Georgios Zervakis, Kostas Siozios, Dimitrios Soudris, and Jörg Henkel. 2022. Adapt: fast emulation of approximate dnn accelerators in pytorch. *arXiv preprint arXiv:2203.04071*. https://arxiv.org/abs/2203.04071.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[22] Matthew Douthwaite, Fernando García-Redondo, Pantelis Georgiou, and Shidhartha Das. 2019. A time-domain current-mode mac engine for analogue neural networks in flexible electronics. In *IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 1–4.

[23] Dheeru Dua and Casey Graff. 2017. UCI machine learning repository. (2017).

[24] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: a survey. *J. Mach. Learn. Res.*, 20, 1, (Jan. 2019), 1997–2017.

[25] Ben Feinberg, Shibo Wang, and Engin Ipek. 2018. Making memristive neural network accelerators reliable. In *Int. Symp. High Performance Computer Architecture, HPCA*. IEEE, 52–65.

[26] Amira Guesmi et al. 2021. Defensive approximation: securing cnns using approximate computing. In *Int. Conf. Architectural Support for Programming Languages and Operating Systems*, 990–1003.

[27] Vaibhav Gupta, Debabrata Mohapatra, Anand Raghunathan, and Kaushik Roy. 2013. Low-power digital signal processing using approximate adders. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 32, 1, 124–137.

[28] Song Han et al. 2016. Eie: efficient inference engine on compressed deep neural network. *SIGARCH Comput. Archit. News*, 44, 3, (June 2016), 243–254.

[29] Zhezhi He, Jie Lin, Rickard Ewetz, Jiann-Shiun Yuan, and Deliang Fan. 2019. Noise injection adaption: end-to-end reram crossbar non-ideal effect adaption for neural network mapping. In *Design Automation Conference*, 1–6.

[30] Jörg Henkel, Heba Khdr, Santiago Pagani, and Muhammad Shafique. 2015. New trends in dark silicon. In *Design Automation Conference (DAC)*, 1–6.

[31] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

[32] KC Hsu et al. 2015. A study of array resistance distribution and a novel operation algorithm for wox reram memory. In *SSDM*.

[33] Miao Hu, Hai Li, Yiran Chen, Qing Wu, Garrett S. Rose, and Richard W. Linderman. 2014. Memristor crossbar-based neuromorphic computing system: a case study. *IEEE Trans. on Neural Networks and Learning Systems*, 25, 10, 1864–1878.

[34] Miao Hu, Hai Li, Yiran Chen, Qing Wu, Garrett S. Rose, and Richard W. Linderman. 2014. Memristor crossbar-based neuromorphic computing system: A case study. *IEEE Trans. Neural Networks Learn. Syst.*, 25, 10, 1864–1878.

[35] Miao Hu et al. 2016. Dot-product engine for neuromorphic computing: programming 1t1m crossbar to accelerate matrix-vector multiplication. In *design automation conference (dac)*. IEEE, 1–6.

[36] Mingu Kang, Sujan K. Gonugondla, Ameya Patil, and Naresh R. Shanbhag. 2018. A multi-functional in-memory inference processor using a standard 6t sram array. *IEEE Journal of Solid-State Circuits*, 53, 2, 642–655.

[37] Yao-Wen Kang, Chun-Feng Wu, Yuan-Hao Chang, Tei-Wei Kuo, and Shu-Yin Ho. 2020. On minimizing analog variation errors to resolve the scalability issue of reram-based crossbar accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39, 11, 3856–3867.

[38] Y. Kim, S. Venkataramani, N. Chandrachoodan, and A. Raghunathan. 2019. Data subsetting: a data-centric approach to approximate computing. In *Design, Automation Test in Europe Conference Exhibition*. (Mar. 2019), 576–581.

[39] Peter Lacy, Jessica Long, and Wesley Spindler. 2020. Fast-moving consumer goods (fmcg) industry profile. In *The Circular Economy Handbook*. Springer.

[40] Seung Ryul Lee et al. 2012. Multi-level switching of triple-layered taox rram with excellent reliability for storage class memory. In *Symposium on VLSI Technology (VLSIT)*. IEEE, 71–72.

[41] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G. Zorn. 2011. Flikker: saving dram refresh-power through critical data partitioning. *SIGPLAN Not.*, 46, 3, (Mar. 2011), 213–224.

[42] Jiayuan Mengte, Anand Raghunathan, Srimat Chakradhar, and Surendra Byna. 2010. Exploiting the forgiving nature of applications for scalable parallel execution. In *Int. Symp. Parallel & Distributed Processing (IPDPS)*, 1–12.

[43] Joshua San Miguel, Mario Badr, and Natalie Enright Jerger. 2014. Load value approximation. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture* (MICRO-47). IEEE Computer Society, Cambridge, United Kingdom, 127–139.

[44] Muhammad Husnain Mubarik et al. 2020. Printed machine learning classifiers. In *Annu. Int. Symp. Microarchitecture (MICRO)*, 73–87.

[45] Emre Özer et al. 2020. A hardwired machine learning processing engine fabricated with submicron metal-oxide thin-film transistors on a flexible substrate. *Nature Electronics*, 3, (July 2020), 1–7.

[46] Subhankar Pal et al. 2018. Outerspace: an outer product based sparse matrix multiplication accelerator. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 724–736.

[47] Priyadarshini Panda, Indranil Chakraborty, and Kaushik Roy. 2019. Discretization based solutions for secure machine learning against adversarial attacks. *IEEE Access*, 7, 70157–70168.

[48] Jeff Pool, Abhishek Sawarkar, and Jay Rodge. 2021. Accelerating inference with sparsity using the nvidia ampere architecture and nvidia tensorrt. Nvidia blog. Retrieved Aug. 8, 2022 from https://developer.nvidia.com/blog/accelerating-inference-with-sparsity-using-ampere-and-tensorrt.

[49] Gabriele Prato, Ella Charlaix, and Mehdi Rezagholizadeh. 2019. Fully quantized transformer for machine translation. *arXiv preprint arXiv:1910.10485*.

[50] Ashish Ranjan, Swagath Venkataramani, Xuanyao Fong, Kaushik Roy, and Anand Raghunathan. 2015. Approximate storage for energy efficient spintronic memories. In *Design Automation Conference (DAC)*, 1–6.

[51] Syed Shakib Sarwar, Swagath Venkataramani, Anand Raghunathan, and Kaushik Roy. 2016. Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 145–150.

[52] Sanchari Sen, Shubham Jain, Swagath Venkataramani, and Anand Raghunathan. 2019. Sparce: sparsity aware general-purpose core extensions to accelerate deep neural networks. *IEEE Trans. Comput.*, 68, 6, (June 2019), 912–925.

[53] Ali Shafiee et al. 2016. Isaac: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News*, 44, 3, 14–26.

[54] Muhammad Shafique, Waqas Ahmad, Rehan Hafiz, and Jörg Henkel. 2015. A low latency generic accuracy configurable adder. In *Design Automation Conference (DAC)*, 1–6.

[55] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. 2017. PipeLayer: A pipelined ReRAM-based accelerator for deep learning. In *Int. Symp. High Performance Computer Architecture, HPCA*, 541–552.

[56] Xiao Sun et al. 2019. Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks. In *Advances in Neural Information Processing Systems*, 4901–4910.

[57] Zois-Gerasimos Tasoulas, Georgios Zervakis, Iraklis Anagnostopoulos, Hussam Amrouch, and Jörg Henkel. 2020. Weight-oriented approximation for energy-efficient neural network inference accelerators. *IEEE Trans. Circuits Syst.*, 67-I, 12, 4670–4683.

[58] Minh SQ Truong et al. 2021. Racer: bit-pipelined processing using resistive memory. In *Annu. Int. Symp. on Microarchitecture*, 100–116.

[59] Ashish Vaswani et al. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

[60] Swagath Venkataramani, Srimat T. Chakradhar, Kaushik Roy, and Anand Raghunathan. 2015. Approximate computing and the quest for computing efficiency. In *Design Automation Conference (DAC)*, 1–6.

[61] Swagath Venkataramani, Vinay K. Chippa, Srimat T. Chakradhar, Kaushik Roy, and Anand Raghunathan. 2013. Quality programmable vector processors for approximate computing. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 1–12.

[62] Swagath Venkataramani, Anand Raghunathan, Jie Liu, and Mohammed Shoaib. 2015. Scalable-effort classifiers for energy-efficient machine learning. In *Design Automation Conference*, 6 pages.

[63] Swagath Venkataramani, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. 2014. Axnn: energy-efficient neuromorphic systems using approximate computing. In *Int. Symp. Low Power Electronics and Design (ISLPED)*, 27–32.

[64] Swagath Venkataramani, Amit Sabne, Vivek Kozhikkottu, Kaushik Roy, and Anand Raghunathan. 2012. Salsa: systematic logic synthesis of approximate circuits. In *Design Automation Conference (DAC)*, 796–801.

[65] Swagath Venkataramani et al. 2020. Efficient ai system design with cross-layer approximate computing. *Proceedings of the IEEE*, 108, 12, 2232–2250.

[66] Swagath Venkataramani et al. 2021. Rapid: ai accelerator for ultra-low precision training and inference. In *Int. Symp. Computer Architecture (ISCA)*, 153–166.

[67] Shibo Wang and Pankaj Kanwar. 2019. Bfloat16: the secret to high performance on cloud tpus. Google blog. Retrieved Aug. 8, 2022 from https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus.

[68] Dennis D. Weller, Michael Hefenbrock, Mehdi B. Tahoori, Jasmin Aghassi-Hagmann, and Michael Beigl. 2020. Programmable neuromorphic circuit based on printed electrolyte-gated transistors. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 446–451.

[69] Dennis D. Weller et al. 2021. Printed stochastic computing neural networks. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, 914–919.

[70] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*. D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, (Eds.) Vol. 29. Curran Associates, Inc.

[71] Bonan Yan, Jianhua Yang, Qing Wu, Yiran Chen, and Hai Li. 2017. A closed-loop design to enhance weight stability of memristor based neural network chips. In *Int. Conf. Computer-Aided Design (ICCAD)*. IEEE, 541–548.

[72] Xiaoxuan Yang, Brady Taylor, Ailong Wu, Yiran Chen, and Leon O. Chua. 2022. Research progress on memristor: from synapses to computing systems. *IEEE Trans. Circuits Syst. I, Reg. Papers*, 69, 5, 1845–1857.

[73] Xiaoxuan Yang, Bonan Yan, Hai Li, and Yiran Chen. 2020. Retransformer: reram-based processing-in-memory architecture for transformer acceleration. In *Int. Conf. Computer-Aided Design (ICCAD)*, 1–9.

[74] Xiaoxuan Yang et al. 2021. Multi-objective optimization of reram crossbars for robust dnn inferencing under stochastic noise. In *Int. Conf. Computer-Aided Design*, 1–9.

[75] Joonsang Yu et al. 2021. Nn-lut: neural approximation of non-linear operations for efficient transformer inference. *arXiv preprint arXiv:2112.02191*.

[76] Shimeng Yu, Ximeng Guan, and H-S Philip Wong. 2012. On the switching parameter variation of metal oxide rram—part ii: model corroboration and device design strategy. *IEEE Transactions on Electron Devices*, 59, 4, 1183–1188.

[77] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*. IEEE, 36–39.

[78] G. Zervakis, K. Koliogeorgi, D. Anagnostos, N. Zompakis, and K. Siozios. 2019. Vader: voltage-driven netlist pruning for cross-layer approximate arithmetic circuits. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 27, 6, 1460–1464.

[79] Georgios Zervakis, Ourania Spantidi, Iraklis Anagnostopoulos, Hussam Amrouch, and Jörg Henkel. 2021. Control variate approximation for dnn accelerators. In *Design Automation Conference (DAC)*, 481–486.

[80] Georgios Zervakis, Kostas Tsoumanis, Sotirios Xydis, Dimitrios Soudris, and Kiamal Pekmestzi. 2016. Design-efficient approximate multiplication circuits through partial product perforation. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 24, 10, 3105–3117.

[81] Georgios Zervakis et al. 2022. Thermal-aware design for approximate dnn accelerators. *IEEE Trans. Comput.*, 1–1.

[82] Wei Zhang, Suyog Gupta, Xiangru Lian, and Ji Liu. 2016. Staleness-aware async-sgd for distributed deep learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence* (IJCAI'16). AAAI Press, New York, New York, USA, 2350–2356.

[83] Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. 2019. Improving neural network quantization without retraining using outlier channel splitting. In *International Conference on Machine Learning*, 7543–7552.