Accelerating AI Applications using Analog In-Memory Computing: Challenges and Opportunities

Shravya Channamadhavuni, Sven Thijssen, Sumit Kumar Jha, and Rickard Ewetz
University of Central Florida, Orlando, USA
University of Texas at San Antonio, San Antonio, USA

ABSTRACT

Linear transformations are the dominating computation within many artificial intelligence (AI) applications. The natural multiply and accumulate feature of resistive crossbar arrays promise unprecedented processing capabilities to resistive dot-product engines (DPEs), which can accelerate approximate matrix-vector multiplication using analog in-memory computing. Unfortunately, the functional correctness of the accelerated AI applications may be compromised by various sources of errors. In this paper, we will outline the most pressing robustness challenges, the limitations of state-of-the-art solutions, and future opportunities for research.

CCS CONCEPTS

• **Hardware** → **Emerging architectures**; • **Computing methodologies** → *Artificial intelligence.*

KEYWORDS

Analog in-memory computing, memristor, analog matrix-vector multiplication, variations, reliability, robustness.

ACM Reference Format:
Shravya Channamadhavuni, Sven Thijssen, Sumit Kumar Jha, and Rickard Ewetz. 2021. Accelerating AI Applications using Analog In-Memory Computing: Challenges and Opportunities. In *Proceedings of the Great Lakes Symposium on VLSI 2021 (GLSVLSI '21), June 22–25, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3453688.3461746

1 INTRODUCTION

With the exponential growth and availability of digital data, we have entered an era dominated by data-driven artificial intelligence (AI) applications. As a result, the demand for data to be analyzed and processed has rapidly increased to exascale (10^{18} bytes/s). Unfortunately, these computing needs cannot be met through further technology scaling using traditional silicon technology and von-Neumann architecture. Mainly, due to the separation of computing units and memory units, which translates into power hungry and bandwidth limited data transfer [40]. Recently, numerous large-scale research programs and research efforts have been devoted to improving energy-efficiency and reducing data movement, by rethinking all layers of the computing stack, including hardware, software and hardware/software fundamental approaches and schemes [1, 2, 4].

Due to promises of simultaneous dense storage and energy-efficient analog processing, in-memory computing based on non-volatile resistive technology has emerged as an appealing solution to overcome the aforementioned challenges. A non-volatile resistive device is a two terminal device with programmable resistance, which may be realized using memristor [11, 35], resistive random access memory (ReRAM) [23, 38], phase change memory (PCM) [20, 39], or spin-transfer torque magnetic random access memory (STT-RAM) [18, 31]. By integrating the emerging devices into resistive crossbar arrays (RCAs), approximate matrix-vector multiplication (MVM) can be executed in the analog domain. This is promising because the computation is significantly (orders of magnitude) more energy-efficient than in the digital domain [17]. Data movement is also substantially reduced by storing the matrix in-memory and performing the computation in-situ [9, 32]. Moreover, MVM is the dominating computation in many AI applications such as deep learning [22], image processing [24], and graph analytics [34].

The main challenge of leveraging analog in-memory computing is that the computational accuracy may be degraded by various sources of errors and variations. This includes device write errors, non-zero array parasitics, limited device yield, resistance drift, temperature variations, random telegraph noise, and limited device endurance. Moreover, any error introduced in the analog domain may compromise the functional correctness of the accelerated applications. For example, the hardware classification accuracy of a neural network may be significantly lower than the software level. In contrast, robustness issues within digital computing systems only introduce timing violations, which can be alleviated using dynamic voltage frequency scaling (DVFS).

To provide guarantees on the system level performance, synergistic innovations on the device level, algorithm level, and software application level are required. While device level researchers continuously attempt to improve the characteristics of the fabricated devices, it is becoming urgent to develop the required algorithm and software level support.

In this paper, we review the challenges, solutions, and future research directions for accelerating AI applications using analog in-memory computing. The basic concept of analog matrix-vector multiplication, target AI applications, and the modeling of different errors is discussed in Section 2. The state-of-the-art solutions to improving the robustness to errors on the algorithm and software level are reviewed in Section 3. Opportunities for future research are outlined in Section 4. The paper is concluded in Section 5.

GLSVLSI '21, June 22–25, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8393-6/21/06…$15.00
https://doi.org/10.1145/3453688.3461746

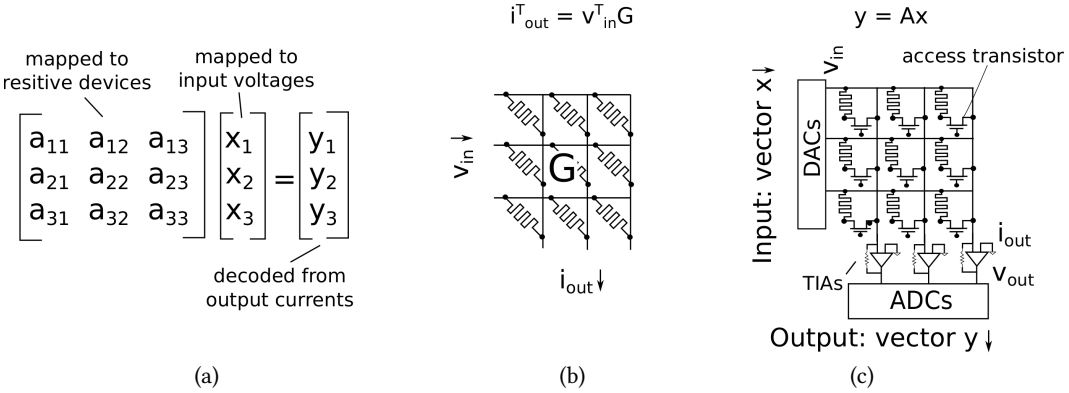

Figure 1: (a) Digital MVM (b) Analog MVM (c) RCA circuit for analog MVM.

2 BACKGROUND

In this section, we review how analog MVM can be performed using RCAs. Next, we outline target AI applications and sources of errors.

2.1 Analog MVM using RCAs

The high-level concept and circuitry needed to perform MVM operations (Ax=y) using an RCA is illustrated in Figure 1. The MVM operations are accelerated using a one-time expensive initialization phase and a fast and efficient evaluation phase. In the initialization phase, a matrix A is mapped to the state (or conductance) of the multi-level resistive devices within an RCA. This is a slow and expensive process because the conductance of each device in the RCA is required to be accurately programmed based on the input matrix. In the evaluation phase, MVM operations are performed fast and efficiently by converting input vectors (x) into input voltages and decoding the output voltages/currents into output vectors (y), which is shown in Figure 1(a). Consequently, RCAs are a promising candidate to accelerate applications where the matrix is relatively fixed and the input vectors frequently change.

The basic principal of analog MVM is shown in Figure 1(b). The figure shows a set of wordlines connected to a set of bitlines using a resistive device in each intersection. The current through each resistive device is obtained using Ohm's law by multiplying the input voltages applied to the wordlines with the conductance values of the resistive devices. Next, the currents are summed along the bitlines using Kirchhoff's current law. The relation between the input voltages (v_{in}) and the output currents (i_{out}) is equal to $v^T_{in}G = i^T_{out}$, where G is the conductance matrix realized by the RCA. The conductance matrix G has dimensions (N)x(M) for a RCA with N wordlines and M bitlines. Let the conductance values of the resistive devices (organized in a matrix form) be denoted g, where $g_{ij} \in [g_{min}, g_{max}]$ is the conductance of the resistive devices connecting wordline i with bitline j. In the ideal case (no array/input/output resistances), each entry G_{ij} in G is equal to g_{ij}. Hence, the conductance values g are obtained by linearly mapping the target matrix A into $[g_{min}, g_{max}]$.

The circuit of an RCA used for analog MVM is shown in Figure 1(c). There are digital to analog converters (DACs) attached to the wordlines that are used to convert a digital input vector x into analog input voltages v_{in}. The transimpedance amplifiers (TIAs) attached to the bitlines amplify the output currents (i_{out}) into output voltages (v_{out}), where $v_{out} = i_{out}R_s$ and R_s is the feedback resistance of the TIAs. Consequently, the output voltages are equal to $v^T_{out} = v^T_{in}GR_s$. Next, output voltages v_{out} are converted into a digital vector y using analog to digital converters (ADCs).

2.2 Target AI Applications

Applications such as DNNs, signal and image processing, and graph processing can be accelerated using RCAs. All these applications are characterized by that (i) MVM (or the associated data transfer) is the bottleneck of the application and (ii) the matrix is relatively fixed and the input vectors frequently change. Now we specifically turn our attention to the acceleration of DNNs.

Deep neural networks (DNNs): DNNs have surpassed human-level capabilities for a number of computer vision applications [22]. DNNs consist of multiple layers of neurons connected together by synapse weights, which is shown in Figure 2. The networks operate using a training and an inference phase. In the training phase, the synapse weights are learned to solve a classification task. In the inference phase, input images/objects/videos are classified into one of multiple output categories by passing an input to the first layer and recording the output from the last layer. The evaluation of one layer of a neural network involves multiplying the outputs from the previous layer with the synapse weights (a MVM operation) and passing the result through a non-linear activation functions. It is appealing to accelerate the inference phase of DNNs by mapping each weight matrix to an RCA because the MVM operation and the associated memory access is the bottleneck limiting the system performance. Using this high-level approach, architectural level studies have demonstrated significant improvements in power, area, latency, and throughput [9, 32, 33].

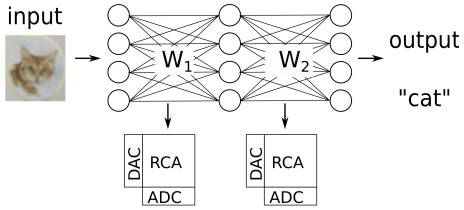

Figure 2: DNN inference deployment on RCAs.

2.3 Sources of errors

The main challenge of leveraging MCAs to accelerate MVM operations is that various sources may introduce errors that degrade system performance. A list of the most important sources of errors are provided below.

- **Write accuracy:** Resistive devices cannot be exactly programmed to a specific conductance value due to device variabilities. With respect to a desired target conductance, the obtained device conductance exhibits a log normal distribution [17]. The programming accuracy can on an abstract level be captured using bit-accuracy b, i.e., the resistive devices can be programmed to 2^b distinguishable conductance states [3, 15]. Consequently, each element in g is a discrete variable with states uniformly distributed between $[g_{min}, g_{max}]$, where g_{min} and g_{max} are the minimum and maximum conductance of programmable conductance range. Device characterizations using non-uniform state distributions have also been analyzed [30].
- **Array parasitics:** In the ideal case, the conductance matrix G is equivalent to the conductance values of the resistive devices g. However, the conductance matrix $G(g, r_{parasitic})$ (or G) is in reality a highly non-linear function of g and the parasitics $r_{parasitic}$. Let $r_{parasitic}$ capture the non-zero input (driver), output (sensing), and array (wire) parasitics that are fixed after fabrication [26]. In this paper, we interchangeably use G or $G(g, r_{parasitic})$ to balance clarity and brevity. Let the RCA have M and N wordlines and N bitlines, respectively. The conductance matrix G can be obtained as follows:

$$G(g, r_{parasitic}) = SY(g, r_{parasitic})^{-1}B, \tag{1}$$

  where B, S, and $Y(g, r_{parasitic})$ are matrices. The B and S matrices respectively have dimensions (M)x$(2NM+M+N)$ and $(2NM+M+N)$x(N) and only depend on the size of the RCA. $Y(g, r_{parasitic})$ is a matrix with dimension $(2MN+M+N)$x$(2MN+M+N)$ that depends on the fixed parasitic properties $r_{parasitic}$ and the conductance values g of the resistive devices. The details of the matrices Eq (1) are provided in [26].
- **Device defects/Stuck-at-fault defects:** A resistive device can suffer hard defect, which implies that the device conductance cannot be further programmed. The stuck-at-faults occur at fabrication or from heavy device utilization [5, 37]. Research with insights in the spatial distribution of the defects and the values have mainly focused on devices stuck to the minimum or maximum programmable conductance g_{min} and g_{max}.
- **DAC/ADC quantization errors:** The domain interfaces introduce errors in both the digital/analog and the analog/digital conversion. The DAC quantize the value range into b_{dac} bits or $2^{b_{dac}}$ states. The ADC quantize the output voltage range into b_{adc} bits or $2^{b_{adc}}$ states. The errors introduced by the DACs and ADCs are therefore proportional to the input and output value range, respectively.
- **Resistance drift:** The resistance of every memristor will be slightly changed after each MVM operation is performed due to resistance drift [7]. Consequently, the computational accuracy of the MVM operations will be degraded over time.
- **Temperature variations:** The conductance of each resistive device is dependent on the temperature of the operating environment [17]. The devices are more (less) conductive at higher (lower) temperatures.
- **Stochastic variations:** Random telegraph noise (RTN) conductance variations for every resistive device [10, 12]. Johnson and short noise introduce non-ideal currents through the resistive devices and TIAs, respectively.
- **Non-linear device characteristics:** Non-linear device characteristics result in that the resistive devices act as a non-ideal device $i(v, s)$ instead of an ideal resistor/conductor. Consequently, the current through a resistive device $i(v, s)$ is a function of a state variable s and the voltage across the device v. Quantitatively, the resistive devices have been reported to become more conductive devices under high voltage excitations [17].

3 STATE-OF-THE-ART SOLUTIONS TO IMPROVING ROBUSTNESS

In this section, we outline the state-of-the-art solutions to handle array parasitics, stuck-at-fault defects, and DAC/ADC quantization errors. We selected these errors as they tend to be the dominant sources for many AI applications. For each source, we outline the main techniques that can be used to improve the application level performance. We note that many of the techniques are used to compensate for multiple different types of errors.

3.1 Array parasitics

An overview of the techniques used to compensate for non-zero array parasitics are shown in Table 1.

Table 1: Techniques used to compensate for array parasitics.

Work	Conductance mapping	Post-processing	Retraining networks
[16]	Yes	-	-
[41]	Yes	-	-
[26]	Yes	Yes	-
[17]	Yes	Yes	-
[47]	Yes	-	-
[36]	Yes	-	-
[19]	-	-	Yes
[14]	-	-	Yes

Conductance mapping is the concept of mapping a matrix A into conductance values g such that the realized conductance matrix $G(g, r_{parasitic})$ is proportional to A. The challenge stems from that: (i) there is voltage drop (or IR-drop) over the array parasitics, and (ii) currents may flow through multiple paths from an input to an output in the RCA. Nevertheless, given the conductance values g and the parasitics $r_{parasitic}$, the realized conductance matrix $G(g, r_{parasitic})$ can be computed analytically using Eq (1).

Early work on conductance mapping focused on specifying the conductance values g while compensating for the non-zero output resistance of the TIAs. A linear approximation technique was used in [16]. An iterative technique was proposed in [41]. The first technique that explicitly captured the array parasitics was proposed in [26]. The method was based on defining the matrix realized by

an RCA to be $A^r = G/\alpha$, where α is scaling factor and G is the conductance matrix obtained using Eq (1). Next, the conductance values g were specified by minimizing $||A - A^r||^2$ using steepest gradient decent, where $||.||^2$ denotes the square of the Frobenious norm. The limitation of that work is that the write accuracy of the resistive devices was not considered. The run-time of that algorithm was also very long[1]. In [17], an extremely fast conductance mapping algorithm was proposed. First, an ideal target current through each resistive device was determined while treating the RCA to be ideal. Next, the conductance values of the resistive devices were tuned using Newton's algorithm to the determined target currents. This method is a heuristic because it does not explicitly minimize the difference between A and A^r. However, empirically it has shown to give good results. In [47], the method in [26] was extended to consider the write accuracy of the resistive devices. This was performed by minimizing $||A - A^r||$ while optimizing both the scaling factor α and the conductance values g. This method results in significantly higher accuracy compared with in [17] and in [26]. In [36], the method was further extended to reduce run-time and handle memristors arranged in a differential pair configuration. However, the run-time of the algorithm is still longer than desired for state-of-the-art neural networks.

An orthogonal approach to compensating for the IR-drop over the array parasitics is to modify the application to account for the IR-drop. In particular, retraining of the neural network weights has been explored [14, 19]. Our understanding is that the retraining was performed by linearly mapping the neural network weights into conductance values and computing the weight matrix that is effectively realized. In [19], the accurate model in Eq (1) was used to model the array parasitics. In [14], an approximate statistical model was used to reduce the run-time of the technique.

3.2 Device defects/Stuck-at-fault defects

An overview of the techniques used to compensate for stuck-at-fault defects is shown in Table 2.

Table 2: Stuck-at-fault mitigation techniques.

Work	Retraining	Hardware redundancy	Data to hardware assignment	Digital compensation	Post-processing
[8]	Yes	Yes	Routers	-	-
[28]	Yes	Yes	Routers	-	-
[14]	Yes	Yes	-	Yes	-
[42]	-	Yes	-	-	-
[48]	-	-	Routers	-	Yes
[45]	-	Yes	Data layout organization	-	-
[27]	Yes	Yes	Routers	-	-
[43]	Yes	-	Data layout organization	-	-
[46]	optional	Yes	Data layout organization	-	-
[44]	-	Yes	Data layout organization	-	-

Hardware-aware training aims to train the weights of a neural network in software to mimic the defects in the RCA hardware [8, 28]. The main limitation is that each neural network application must be retrained based on the unique defect pattern of each RCA based platform. Digital co-processing is the concept of compensating for the defects using a digital co-processor [14]. This technique can be used to compensate for any number of defective devices but is expected to introduce significant performance and hardware overhead. Stuck-at-fault defects can also be compensated for using redundant hardware, which involves representing each matrix element using multiple parallel resistive devices [42]. In [48], post-processing techniques were used to minimize the errors using a first order polynomial. In [45], it was observed that the constant term of the polynomial can be captured by using the weights without any overhead. In [27], it was observed that the negative impact of stuck-at-fault defects can be reduced by optimizing the data to hardware assignment. In [27], small (large) matrix elements were assigned to devices stuck-off (stuck-on). The assignment was realized by permuting rows using routers. The matrix row to RCA row assignment was guided by a greedy algorithm. In [8], the mapping was formulated as an assignment problem, which can be solved optimally using the Hungarian algorithm. In [43], it was observed that the data to hardware assignment could be performed without hardware overhead by reordering the neurons in each layer of neural network applications, which can be referred to as data layout organization. In [42], data layout organization was performed by solving an assignment problem for the neurons in each layer of a neural network. The technique also seamlessly incorporated the use of hardware redundancy. The main drawback is that there are few opportunities to perform data layout organization when weights are shared in modern neural networks. In [44], it was shown that additional opportunities for data layout organization are created when weights are replicated to improve throughput. The main drawback of that approach is that the run-time becomes long for larger DNNs.

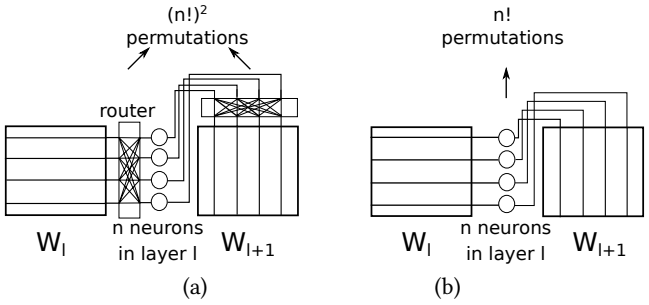

Figure 3: Data to hardware assignment (a) using routers and (b) using data layout organization. The figure is adapted from [46].

3.3 DAC/ADC quantization errors

An overview of the techniques used to reduce the impact of DAC/ADC quantization errors is shown in Table 3.

Table 3: Techniques for DAC/ADC quantization errors.

Work	Bit-slicing	Computational restructuring	Fully analog
[6]	Yes	-	-
[32]	Yes	-	-
[13]	Yes	-	-
[25]	-	Yes	-
[29]	-	-	Yes

The precision of the DAC and ADCs is limited to 8-bits. However, many scientific computing applications require 16-bit fixed-point

[1]The long run-time was observed by the authors when reimplementing the algorithm.

or floating point. To overcome this limitation, many architectural level studies have adopted bit-slicing techniques to emulate high precision [6, 13, 32]. The concept is based on decomposing both the input vector and the matrix into bit-slices. Next, each of the bit-slices are multiplied and added together using a shift-and-add reduction network. The concept of bit-slicing of a matrix is shown below:

$$\begin{bmatrix} 2 & 1 \\ 7 & 3 \end{bmatrix} = 2^2 \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} + 2^1 \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} + 2^0 \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

The limitation of bit-slicing is that the paradigm is highly vulnerable to errors. Specifically, small analog errors may be amplified into large digital errors by the shift operations.

Two alternative methods to minimizing DAC/ADC quantization errors is based on computational reconstruction and fully-analog. In [25], the neural network was trained to minimize the impact of quantization errors, i.e., the functional computation was reconstructed. In [29], multiple crossbars were connected together without intermediate digital interfaces. Consequently, a neural network can be implemented only using DACs and ADCs before the first and after last layer, respectively.

4 FUTURE RESEARCH OPPORTUNITIES

In this section, we outline three future research directions. The first two are related to the challenges described in this paper and the last has a broader perspective.

Solutions to simultaneously handling multiple sources of errors: Many research studies are focused on handling a single source of errors, which is a natural starting point to addressing robustness and reliability issues for an emerging technology. However, when multiple sources of errors are simultaneously considered, it is crucial that different error mitigation techniques are compatible. For example, it is easy to understand that bit-slicing techniques (the concept of shifting and adding results) can reduce the precision requirements on the digital/analog and analog/digital domain interfaces. However, the shift operations will amplify any small analog error into large digital error, which will certainly impact the application level functional correctness. Consequently, there is an urgent need to rethink how to mitigate the negative impact of errors while considering the interplay between different sources of errors. In particular, it may be important to understand how the techniques used to compensate for non-zero array parasitics interplay with the techniques used to handle the other robustness issues.

Standardisation of modeling, simulation, and evaluation frameworks and tools: While the device models for resistive devices are still emerging, it is still possible to standardize evaluation frameworks and modeling/simulation tools based on well thought-out assumptions. This may be particularly important for RCA based computing systems with a noisy analog component, which behaves fundamentally different from a deterministic digital systems. Without clear assumptions, it is very difficult to analyse the real capabilities and limitations of analog in-memory computing. Consequently, there is a risk that academic researchers are not focusing on the critical issues that are preventing the technology from reaching commercial deployment. Moreover, due to the lack open-source tools, it is very difficult (or impossible) to benchmark alternative solutions and reproduce previously published results. We believe that the establishment of open-source evaluation frameworks and modeling/simulation tools has the potential to improve the overall quality of the research in the academic community and reduce the infrastructure building efforts for each independent research group.

Mapping of new applications to RCAs and characterization of hardware requirements: The mapping of new applications to resistive technology is of high interest to both the industry and the academic community. In particular, the mapping of new mathematical kernels that enable new families of applications to be solved. For example, the technique of solving linear equations through iterative refinement is a brilliant innovation that enables linear systems of equations to solved using resistive computing system [21]. A different piece of the puzzle that is missing is the characterization of the hardware requirements of various applications. Currently, experiments in resistive hardware have mainly demonstrated promising results for image processing and small neural networks on the MNIST data set. It would be useful to truly understand the device level properties that limit larger applications (or more error sensitive applications) deployed on resistive hardware. This would provide insight and feedback to device level researchers on what properties are the most urgent to improve.

5 CONCLUSIONS

A departure from silicon technology and the von-Neumann architecture is required to accelerate modern AI applications that are driven by big data. While analog in-memory computing using emerging non-volatile resistive technology promises reductions in data movement and significant (orders of magnitude) improvements in power-efficiency, the functional correctness of the accelerated applications may be compromised by errors. In this paper, we have reviewed the state-of-the-art techniques to improve the robustness and reliability of resistive computing systems to errors. We concluded the paper with our vision of opportunities for future research within analog in-memory computing.

ACKNOWLEDGMENTS

This work was partly supported by NSF awards 1755825, 1908471, 2113307, and Cyber-Florida grant 3910-1011-00-E.

REFERENCES

[1] [n.d.]. Exascale Proxy Applications. https://proxyapps.exascaleproject.org.
[2] [n.d.]. Joint University Microelectronics Program (JUMP). https://www.darpa.mil/program/joint-university-microelectronics-program.
[3] Fabien Alibart, Ligang Gao, Brian D Hoskins, and Dmitri B Strukov. 2012. High Precision Tuning of State for Memristive Devices by Adaptable Variation-tolerant Algorithm. *Nanotechnology* 23, 7 (2012), 075201.
[4] Cornelia I Bargmann and William T Newsome. 2014. The Brain Research Through Advancing Innovative Neurotechnologies (BRAIN) Initiative and Neurology. *JAMA neurology* 71, 6 (2014), 675–676.
[5] Karsten Beckmann, Josh Holt, Harika Manem, Joseph Van Nostrand, and Nathaniel C Cady. 2016. Nanoscale Hafnium Oxide RRAM Devices Exhibit Pulse Dependent Behavior and Multi-level Resistance Capability. *Mrs Advances* 1, 49 (2016), 3355–3360.
[6] Mahdi Nazm Bojnordi and Engin Ipek. 2016. Memristive Boltzmann Machine: A Hardware Accelerator for Combinatorial Optimization and Deep Learning. In *HPCA'16*. IEEE, 1–13.
[7] Ting Chang, Sung-Hyun Jo, and Wei Lu. 2011. Short-Term Memory to Long-Term Memory Transition in a Nanoscale Memristor. *ACS nano* 5, 9 (2011), 7669–7676.
[8] Lerong Chen, Jiawen Li, Yiran Chen, Qiuping Deng, Jiyuan Shen, Xiaoyao Liang, and Li Jiang. 2017. Accelerator-friendly Neural-network Training: Learning Variations and Defects in RRAM Crossbar. In *DATE'17*. IEEE, 19–24.

[9] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 27–39.

[10] Shinhyun Choi, Yuchao Yang, and Wei Lu. 2014. Random telegraph noise and resistance switching analysis of oxide based resistive memory. *Nanoscale* 6, 1 (2014), 400–404.

[11] Leon Chua. 1971. Memristor-The Missing Circuit Element. *IEEE Transactions on circuit theory* 18, 5 (1971), 507–519.

[12] R Degraeve, A Fantini, N Raghavan, L Goux, S Clima, B Govoreanu, A Belmonte, D Linten, and M Jurczak. 2015. Causes and consequences of the stochastic aspect of filamentary RRAM. *Microelectronic Engineering* 147 (2015), 171–175.

[13] Ben Feinberg, Uday Kumar Reddy Vengalam, Nathan Whitehair, Shibo Wang, and Engin Ipek. 2018. Enabling scientific computing on memristive accelerators. In *ISCA'18*. IEEE, 367–382.

[14] 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 *DAC'19*. 1–6.

[15] Miao Hu, Catherine E Graves, Can Li, Yunning Li, Ning Ge, Eric Montgomery, Noraica Davila, Hao Jiang, R Stanley Williams, J Joshua Yang, et al. 2018. Memristor-Based Analog Computation and Neural Network Classification with a Dot Product Engine. *Advanced Materials* 30, 9 (2018), 1705914.

[16] 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 TNNLS* 25, 10 (2014), 1864–1878.

[17] Miao Hu, John Paul Strachan, Zhiyong Li, Emmanuelle M Grafals, Noraica Davila, Catherine Graves, Sity Lam, Ning Ge, Jianhua Joshua Yang, and R Stanley Williams. 2016. Dot-Product Engine for Neuromorphic Computing: Programming 1T1M Crossbar to Accelerate Matrix-Vector Multiplication. In *ACM/EDAC/IEEE DAC'16*. IEEE, 1–6.

[18] Yiming Huai. 2008. Spin-Transfer Torque MRAM (STT-MRAM): Challenges and Prospects. *AAPPS bulletin* 18, 6 (2008), 33–40.

[19] Shubham Jain, Abhronil Sengupta, Kaushik Roy, and Anand Raghunathan. 2020. RxNN: A Framework for Evaluating Deep Neural Networks on Resistive Crossbars. *IEEE TCAD* (2020).

[20] Brian G Johnson and Charles H Dennison. 2004. Phase change memory. US Patent 6,791,102.

[21] Manuel Le Gallo, Abu Sebastian, Roland Mathis, Matteo Manica, Heiner Giefers, Tomas Tuma, Costas Bekas, Alessandro Curioni, and Evangelos Eleftheriou. 2018. Mixed-Precision In-Memory Computing. *Nature Electronics* 1, 4 (2018), 246–253.

[22] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[23] HY Lee, PS Chen, TY Wu, YS Chen, CC Wang, PJ Tzeng, CH Lin, F Chen, CH Lien, and M-J Tsai. 2008. Low Power and High Speed Bipolar Switching with A Thin Reactive Ti Buffer Layer in Robust HfO_2 Based RRAM. In *IEEE IEDM'08*. IEEE, 1–4.

[24] Can Li, Yunning Li, Hao Jiang, Wenhao Song, Peng Lin, Zhongrui Wang, J Joshua Yang, Qiangfei Xia, Miao Hu, Eric Montgomery, et al. 2018. Large Memristor Crossbars for Analog Computing. In *ISCAS'18*. IEEE, 1–4.

[25] Beiye Liu, Miao Hu, Hai Li, Zhi-Hong Mao, Yiran Chen, Tingwen Huang, and Wei Zhang. 2013. Digital-Assisted Noise-Eliminating Training for Memristor Crossbar-based Analog Neuromorphic Computing Engine. In *ACM/EDAC/IEEE DAC'13*. IEEE, 1–6.

[26] Beiye Liu, Hai Li, Yiran Chen, Xin Li, Tingwen Huang, Qing Wu, and Mark Barnell. 2014. Reduction and IR-drop Compensations Techniques for Reliable Neuromorphic Computing Systems. In *ICCAD'14*. IEEE, 63–70.

[27] Beiye Liu, Hai Li, Yiran Chen, Xin Li, Qing Wu, and Tingwen Huang. 2015. Vortex: Variation-aware Training for Memristor X-bar. In *ACM/EDAC/IEEE DAC'15*. 1–6.

[28] Chenchen Liu, Miao Hu, John Paul Strachan, and Hai Li. 2017. Rescuing Memristor-based Neuromorphic Design with High Defects. In *ACM/EDAC/IEEE DAC'17*. IEEE, 1–6.

[29] Xiaoxiao Liu, Mengjie Mao, Beiye Liu, Hai Li, Yiran Chen, Boxun Li, Yu Wang, Hao Jiang, Mark Barnell, Qing Wu, et al. 2015. RENO: A High-efficient Reconfigurable Neuromorphic Computing Accelerator Design. In *DAC'15*. 1–6.

[30] Yun Long, Xueyuan She, and Saibal Mukhopadhyay. 2019. Design of Reliable DNN Accelerator with Un-reliable ReRAM. In *DATE'19*. IEEE, 1769–1774.

[31] Stuart Parkin, Xin Jiang, Christian Kaiser, Alex Panchula, Kevin Roche, and Mahesh Samant. 2003. Magnetically Engineered Spintronic Sensors and Memory. *Proc. IEEE* 91, 5 (2003), 661–680.

[32] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 14–26.

[33] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. 2017. PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning. In *HPCA'17*. IEEE, 541–552.

[34] Linghao Song, Youwei Zhuo, Xuehai Qian, Hai Li, and Yiran Chen. 2018. GraphR: Accelerating Graph Processing Using ReRAM. In *HPCA'18*. IEEE, 531–543.

[35] Dmitri B Strukov, Gregory S.Snider, Duncan R.Stewart, and R.Stanley Williams. 2009. The missing memristor found. *Nature* 453, 12 (2009), 80–83.

[36] Necati Uysal, Baogang Zhang, Sumit Kumar Jha, and Rickard Ewetz. 2020. DP-MAP: Towards Resistive Dot-Product Engines with Improved Precision. In *IEEE/ACM ICCAD'20*. IEEE, 1–9.

[37] A van de Goor and Y Zorian. 1993. Effective March Algorithms for Testing Single-Order Addressed Memories. In *1993 European Conference on Design Automation with the European Event in ASIC Design*. IEEE, 499–505.

[38] H-S Philip Wong, Heng-Yuan Lee, Shimeng Yu, Yu-Sheng Chen, Yi Wu, Pang-Shiu Chen, Byoungil Lee, Frederick T Chen, and Ming-Jinn Tsai. 2012. Metal–oxide RRAM. *Proc. IEEE* 100, 6 (2012), 1951–1970.

[39] H-S Philip Wong, Simone Raoux, SangBum Kim, Jiale Liang, John P Reifenberg, Bipin Rajendran, Mehdi Asheghi, and Kenneth E Goodson. 2010. Phase Change Memory. *Proc. IEEE* 98, 12 (2010), 2201–2227.

[40] Wm A Wulf and Sally A McKee. 1995. Hitting the Memory Wall: Implications of the Obvious. *ACM SIGARCH computer architecture news* 23, 1 (1995), 20–24.

[41] Lixue Xia, Peng Gu, Boxun Li, Tianqi Tang, Xiling Yin, Wenqin Huangfu, Shimeng Yu, Yu Cao, Yu Wang, and Huazhong Yang. 2016. Technological Exploration of RRAM Crossbar Array for Matrix-Vector Multiplication. *JCST* 31, 1 (2016), 3–19.

[42] Lixue Xia, Wenqin Huangfu, Tianqi Tang, Xiling Yin, Krishnendu Chakrabarty, Yuan Xie, Yu Wang, and Huazhong Yang. 2018. Stuck-at Fault Tolerance in RRAM Computing Systems. *IEEE JETCAS* 8, 1 (2018), 102–115.

[43] L. Xia, Mengyun Liu, Xuefei Ning, K. Chakrabarty, and Yu Wang. 2017. Fault-Tolerant Training with On-Line Fault Detection for RRAM-Based Neural Computing Systems. In *Proc. Design Automation Conference*. 1–6.

[44] Baogang Zhang and Rickard Ewetz. 2020. Towards Resilient Deployment of In-Memory Neural Networks with High Throughput. In *DAC'21*. 1–9.

[45] Baogang Zhang, Necati Uysal, and Rickard Ewetz. 2019. STAT: Mean and Variance Characterization for Robust Inference of DNNs on Memristor-based Platforms. In *GLSVLSI*. 339–342.

[46] Baogang Zhang, Necati Uysal, Deliang Fan, and Rickard Ewetz. 2019. Handling Stuck-at-Fault Defects Using Matrix Transformation for Robust Inference of DNNs. *IEEE TCAD* 39, 10 (2019), 2448–2460.

[47] Baogang Zhang, Necati Uysal, Deliang Fan, and Rickard Ewetz. 2020. Representable Matrices: Enabling High Accuracy Analog Computation for Inference of DNNs using Memristors. In *ASP-DAC'20*. IEEE, 538–543.

[48] Fan Zhang and Miao Hu. 2020. Defects Mitigation in Resistive Crossbars for Analog Vector Matrix Multiplication. In *ASP-DAC'20*. IEEE, 187–192.