IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 11 October 2022, accepted 31 October 2022, date of publication 3 November 2022, date of current version 5 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3219066

RESEARCH ARTICLE

Neuromorphic In-Memory RRAM NAND/NOR Circuit Performance Analysis in a CNN Training Framework on the Edge for Low Power IoT

NAGARAJ LAKSHMANA PRABHU AND NAGARAJAN RAGHAVAN, (Member, IEEE)
Engineering Product Development (EPD) Pillar, Singapore University of Technology and Design, Singapore 487372

Corresponding Author: Nagarajan Raghavan (nagarajan@sutd.edu.sg)

This work was supported by the A*STAR through the Brain Efficient Nanomechanical Artificial Intelligence Computing (BRENAIC) Programmatic Research under Grant A18A5b0056. The work of Nagaraj Lakshmana Prabhu was supported by the Ministry of Education (MOE), Singapore, through the Research Student Scholarship (RSS) at SUTD (2018–2022).

ABSTRACT Training a CNN involves computationally intense optimization algorithms to fit the network using a training dataset, to update the network weight for inferencing and then pattern classification. Hence, the application of in-memory computation would enable a highly power-efficient low latency on-the-edge CNN training technique by avoiding the memory-wall created during the external memory read/write operation (for off chip instruction and data transfer). A memory write-verify, and re-program technique can control the RRAM variability. Still, memory verification and re-program is a complex process with additional resources needed for practical implementation of verification circuit. In this study, we have demonstrated a practical (First-in Max-Out) FIMO-based cache memory called Maximum Count Binary Comparator Layer (MCBC), using 1T3R, 1T5R, and 1T7R RRAM structures by using a probability-based accuracy improvement architecture, without the conventional verification process. We constructed 10 layered modified MobileNET with filter size ranging from 32 - 512 and trained with Traffic Sign Recognition Database (TSRD) using a three-tier abstraction simulation learning framework - (1) High level, 10 layered CNN implementation with Python+TensorFlow; (2) Verilog HDL based FP32MUL and FP32ADD (32-bits Floating Point adder and multiplier) circuits constructed with RRAM NAND gates using 1T2R structures; and (3) Digital Look-Up-Table (LUT) model for RRAM variability. An edge learning framework (for the forward pass) is demonstrated using digital RRAM-NAND/NOR universal gates integrated with the Maximum Count Binary Comparator Layer (MCBC) to partially circumvent the impact of RRAM variability and to quantify the RRAM variability on the CNN training prediction accuracy for 65nm CMOS OxRAM (TiN/HfO_2/Hf/TiN) with varying device current compliance of 5, 10, and 50μA for low power IoT applications. The MCBC layer was simulated using a SPICE model, for which the estimated chip layout is 1150 $\times$ 1230 nm^2 per logical gate input, which resulted in an overall prediction accuracy improvement from 10% to 60% by repeating the logical operations of the NOR gate for {1, 3, 5, and 7} cycles respectively.

INDEX TERMS Resistive RAM, convolution neural network (CNN), look-up-table (LUT), in-memory computation, image classification, CNN training, Internet of Things (IoT), complementary metal oxide semiconductor (CMOS).

I. INTRODUCTION

In today's deep learning era, advancement in artificial intelligence (AI) models are achieved with bigger training datasets. However, bigger is not always better. The problem with training high-performance AI models and deploying such models entails a tremendous amount of computation power. The model training and inferencing are performed typically by servers placed in the datacenter. The estimated temperature range for a datacenter is between 21°C $\sim$ 24°C

The associate editor coordinating the review of this manuscript and approving it for publication was Mitra Mirhassani.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

and in order to maintain the temperature range, the conventional heating, ventilation, and air conditioning systems (HVAC) such as Air Cooling, Free Cooling, Two-phase Cooled Systems, etc. are used, resulting in a substantial amount of the by product - carbon emission [1]. It has been documented that global carbon emissions due to cloud computing amount to 2.5% to 3.7% of overall global emissions, far outweighing the contributions from the aviation sector [2]. Training a deep neural network (DNNs) model takes considerable mathematical calculations, long-running time, high energy, and dedicated parallel processing units such as Intel CPU, NVIDIA GPU, AMD GPU, and Google TPU performing millions of floating-point operations per second [3], [4]. A consolidated list of popular chip makers and their chips with operational performance to power up cloud-based deep learning model training is shown in Table 1.

It has been documented that global carbon emissions due to cloud computing amount to 2.5% to 3.7% of overall global emissions, far outweighing the contributions from the aviation sector [2]. Training a deep neural network (DNNs) model takes considerable mathematical calculations, long-running time, high energy, and dedicated parallel processing units such as Intel CPU, NVIDIA GPU, AMD GPU, and Google TPU performing millions of floating-point operations per second [3], [4]. A consolidated list of popular chip makers and their chips with operational performance to power up cloud-based deep learning model training is shown in Table 1.

The parallel processing chips used for training can achieve $\sim 10 \times 10^5$ GOPS (Giga Operations Per Second) with a peak power consumption of $\sim$500W and can perform precision computation from integer 8-bits up to floating-point 64-bits of data [5]. The recent trend shows rapid progress in autonomous driving systems integrated with deep learning and AI-based navigation systems deployed for efficiency improvements in the transportation sector and enhancing safer environments. The four major modules for the autonomous navigation system are (1) perception and localization, (2) high-level path planning, (3) low-level path planning, and (4) motion controllers. Today, all four modules use deep learning with LIDAR (to sense distance) and high-speed automotive camera data to perform the necessary sensing and timely control. Training an autonomous AI model equally requires high computation power and therefore exploring the use of in-memory technology will help reduce the overall power consumption and enable moving the training process from the cloud to the edge [6], [7], [8].

With an in-memory computation system, the bottle-neck and extra power barrier to achieving high bandwidth data transfer between the external memory chip and the processor are significantly minimized using the non-von Neumann architecture [9]. The application of non-volatile memory device technologies such as resistive-switching random access memory (RRAM), phase-change memory (PCM), magnetic random-access memory (MRAM), and ferroelectric random-access memory (FeRAM) are studied for in-memory applications [10]. Here, we intend to study further the application of lower power oxygen vacancy-based RRAM for in-memory circuits used for edge-based training to build AI models for the autonomous system. The oxygen vacancy RRAM (OxRAM) is popular for its ultra-low power switching, CMOS compatible process fabrication, high endurance cycle, and multi-bit pseudo-analog memory storage [11]. Ultra-low-power in-memory computation is practical to achieve low powered and battery-operated IoT applications. However, OxRAM devices exhibit stochastic switching due to the oxygen ion / vacancy drift / diffusion and irregular stochastic conductive filament formation and rupture while switching the device between the two resistive states, namely low resistance (LRS) and high resistance states (HRS) [12]. The following section shows various methodologies and process improvements performed by different studies to deal with the imperfect switching and the overall system performance while applied on a deep learning neural network.

TABLE 1. Popular chipsets used for cloud-based AI model training [5].

Make	Chip Name	Computation Precision	Peak ~GOPS	Peak Power ~(W)
Intel®	Arria GX1150	FLOAT16/32	10×10^5	100
	Nirvana2	FLOAT16/32	10×10^4	500
	Nirvana	FLOAT32	10×10^4	500
	Phi7290F	FLOAT64	10×10^3	500
	Phi7210F	FLOAT64	10×10^3	500
Google®	TPU1	INT16	10×10^4	500
	TPU2	FLOAT16/32	10×10^4	500
	TPU3	FLOAT16/32	10×10^4	100
Baidu®	Baidu	FLOAT16/32	10×10^5	100
Cambricon	Cambricon	INT8/FLOAT16	10×10^4	100
Habana Labs	Goya	FLOAT16/32	10×10^4	100
AMD®	AMD-M160	FLOAT16	10×10^4	500
NVIDIA®	P100	FLOAT16	10×10^3	500
	V100	FLOAT16/32	10×10^4	500
	K80	FLOAT32	10×10^3	500

A. OVERVIEW OF RRAM VARIABILITY CONTROL METHODS

RRAM switching variability is an inherent property of the diffusing oxygen ions in the switching process, and two prominent methods are widely studied to achieve a more controlled and enhanced device switching. The first method uses various fabrication process improvements with different material stacks while the second method relies on using an appropriate *re-programming scheme* to identify the more defective device on the given crossbar array and re-map by re-programming these defective devices to achieve device performance improvement. However, the more prominent

TABLE 2. List of various RRAM device switching variability control methodology studies.

Ref #	First Author & Affiliation	Device Studied /Modeling Approach	Architecture/ Fabrication Approach	Variability Control Technique	Remark
[16]	Shihao Song, Drexel University, USA (2021)	Predictive technology model (PTM)	Proposes an architectural solution by formulating the read endurance of an RRAM cell as a function of the programmed synaptic weight and uses an intelligent workload mapping strategy.	The crossbar needs to be shared across multiple clusters of weights. The cluster mapping problem was formulated by exploring the cluster-to-crossbar mapping search space for the maximum inference lifetime by using Hill-Climbing-based local search	Complex Mapping algorithm
[17]	Yuhang Zhang, Shanghai Jiao Tong University, China (2020)	RRAM temporal variation Model	Shift and Duplicate Kernel (SDK) convolutional weight mapping architecture was used. Each kernel is duplicated multiple times and rearranged on different bit lines in a shifted manner, enabling higher intra-layer computational parallelism and reducing the number of input data loading,	Multiple copies of the same trained data with SDK mapping technique to select the less defective data	Increased area and device accuracy depending on SDK's complexity
[18]	Shihui Yin, Arizona State University, USA (2020)	Ideal Resistor Model	The resistance distribution of low-resistance states is tightened by an iterative write-verify scheme	Program and verify	Multiple Re-programming
[19]	Guillem Boquet, Universitat Auto`noma de Barcelona (UAB), Spain (2020)	Statistical conductance model	Neural network training technique to mitigate the impact of device-to-device variation due to conductance imperfections at weight import in offline-learning	Learning and Re-programing	Multiple Re-programming
[20]	Zhong Sun, Informazione e Bioingegneria, Politecnico di Milano, Italy (2020)	Compact Conductance model	The RRAM model and a program-verify algorithm with the PageRank of the Harvard500 data set was computed	Verification and Re-programing	Multiple Re-programming
[21]	Valerio Milo, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Italy (2021)	TiN/Ti/HfO_2/ TiN	To achieve an accurate MLC programming of the 4-kbit RRAM array, two program/verify algorithm approaches were used to modulate top electrode voltage and gate voltage respectively.	Program–Verify scheme and Re-programing	Multiple Re-programming
[22]	Eduardo Pérez, IHP-Leibniz-Institut für Innovative Mikroelektronik, Germany (2021)	TiN/Al:HfO_2/ Ti/TiN	4-kbit 1T1R RRAM array's programming parameters tuned with multi-level incremental step pulse and verify algorithm (M-ISPVA).	Program and verify	Multiple Re-programming
[23]	Yulin Feng, Peking University, China (2021)	TiN/TaO_x/ HfO_2/TiN	The excessive oxygen vacancies generated during the abrupt SET process result in resistive state instability. A triangular programming pulse improves the proposed RRAM device stack's short-term relaxation and long-term retention.	Bidirectional Write-Verify Scheme	Multiple Re-programming
[24]	Jiyong An, Kookmin University, Korea (2022)	Pt/$LaAlO_3$/Nb-doped $SrTiO_3$	The memristor crossbar's columns are divided into 3 groups. The Group-1 columns that are defined as 'severely defective' are deactivated during the training and inference. The Group-2 columns that are defined as moderately defective are re-programmed. The Group-3 columns containing fewer defects than Group-1 and Group-2 are defined as 'normal columns' and are used without re-programming.	Measure defective RRAM array and Re-Program	Multiple Re-programming

TABLE 2. ***(Continued)*. List of various RRAM device switching variability control methodology studies.**

[25]	Wen-Qian Pan, Huazhong University of Science and Technology, China (2019)	TiN/LiSiO$_x$/Pt , TiN/HfO$_2$/Ti and Ti/LiSiO$_x$/Pt	(1) Limit the weight range to improve utilization. (2) "with-read" update scheme to mitigate the write non-linearities. (3) Multiple memristors for each kernel element to reduce the impact of C2C variation.	Quantized weights, Verify-reprogram and duplicated Kernel data	Multiple Re-programming
[Our Methodology]		TiN/HfO$_2$/Hf/ TiN	FIMO-based cache memory model stores 3,5, and 7 input bits and output's a single bit matching the maximum occurring input at a given instant of time, resulting in *probability-based accuracy improvement architecture*.	Hardware-based reprogramming scheme using 1T3R, 1T5R, and 1T7R structure for RRAM-NAND and RRAM-NOR gate-level variability enhancement, applied to in-memory computation architecture.	Simple digital RRAM array, designed as cache memory and used for the reprogramming scheme without verifying to keep the RRAM validation process quick and straightforward

methodology is still to improve the fabrication process and material property. The oxygen ion movement has a stochastic nature, and therefore it is not easy to achieve a controlled switching comparatively on a lower current compliance operation when the number of oxygen vacancies comprising the filament is lower and with wider spread [13], [14], [15]. We see significant research and studies performed in the above areas, and therefore in this paper, we are focusing on the second method to improve the variability using a *re-programming* scheme and re-mapping the more defective array group.

The studies on the re-programming and verification using various programming architectures involving RRAM device model data [16], [17], [18], [19], [20] or actual fabricated device array data or intelligent workload mapping devices array data [21], [22], [23], [24], [25] are discussed in Table 2. Intelligent workload mapping strategy uses Hill-Climbing-based local search technique to map the cluster-to-crossbar array and maximize the inference accuracy [16]. Shift and Duplicate Kernel (SDK) convolutional weight mapping architecture uses multiple copies of the same weights, and the mapping algorithm would select the less defective data [17]. The above techniques used a complex mapping algorithm to address the device variability; however, more area and power were needed to implement such techniques on silicon. The switching imperfection in RRAM due to stochastic distribution of oxygen vacancies is improved by performing a two-step write-verification scheme where the device is programmed-verified-reprogrammed through an iterative process to achieve the device improvement [18], [19], [20], [21], [22], [23]. Alternatively, the defective RRAM array is grouped according to the defect severity level and the re-programming iterations are defined based on the severity levels as shown by An et al. in Ref. [24]. A multiple RRAM re-programming and verification scheme was conducted using quantized trained weights to improve the RRAM variability by Pan et al. [25]. However, all the listed techniques have an additional step to verify and re-program. Here, the verification process is much more complex than the re-programming process and it requires additional logic real estate on the silicon. In this study, we explore a simple RRAM array, designed as cache memory, and used for the reprogramming scheme without verification to keep the RRAM validation process quick and straightforward. Hence, we propose a practical and straight forward FIMO-based cache memory model that stores 3, 5, and 7 input bits and outputs a single bit matching the maximum occurring input at a given instant of time, resulting in *probability-based accuracy improvement architecture* called as *Maximum Count Binary Comparator" – MCBC Layer*. A two-stage combinational circuit demonstrates the functionality of FIMO as shown in Fig.1(a), the first stage operates as a counter logic to count the occurrence of each input (total number of 1's and 0's), and the second stage acts as a comparator to output the maximum value from the given inputs. A simple operational use case is shown in Fig.1(b). The maximum occurred bit of the input stream propagated as the output, demonstrating the probability-based write/read architecture to improve the switching defect inherited by the RRAM device. The

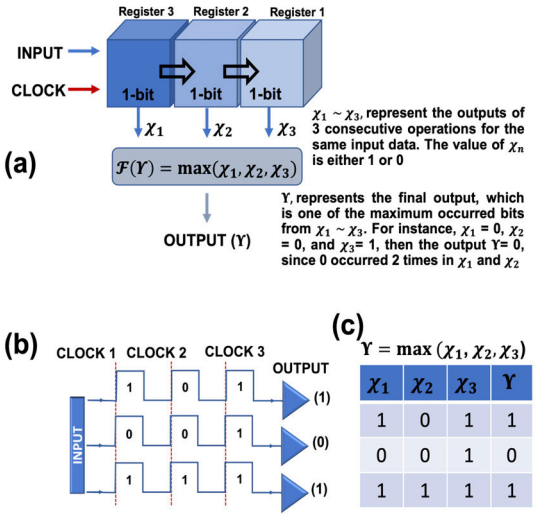


FIGURE 1. **(a) Maximum Count Binary Comparator – MCBC Layer configured with 3-bits FIMO cache memory with Register 1,2, and 3. The input data shifted 1-bit per clock from left to right, and upon reaching 3 clock cycles, the maximum occupancy probability, $F(\Upsilon)$ outputs a single bit with the most occurring input bits. (b) Shows stream of 3 inputs patterns and the most occurred bit transmitted to the output after 3 clock cycles. (c) simulation table showing the output (Υ), which is one of the most occurred bits from χ_1, χ_2 and χ_3.**

TABLE 3. Listing of other RRAM based Edge CNN learning simulation reports focusing on the variability induced performance degradation.

Ref #	First Author & Affiliation	Device Studied / Modeling Approach	Training Architecture Approach	Inference Dataset	Methodology	Strengths and Limitations
[26]	Jiayun Feng, Fudan University China (2021)	40nm $TiN/HfO_2/TaO_x$/ TiN and 7T SRAM arrays	Two-layer 3b-input-3b-weight network in the size of 144-40-10	MNIST	Hybrib Architecture and CIM Macros	Six RRAM CIM arrays for fixed (first) layer and one 7T SRAM CIM array for mutable (Second) layer. The Hybrid CIM architecture aims to improve overall system accuracy; however, the study was conducted on a shallow network.
[27]	Jie Li, Southwest University, China (2021)	Ag/TiO_x/F-doped SnO_2	784–400–10 MLP structure	MNIST	Emulated FPGA+ARM connected to RRAM Crossbar array	A polynomial function to fit the actual memristor pulse modulation curve was designed to accurately describe the real memristor pulse modulation characteristics. Based on a crossbar array of memristors and MLP online training algorithm, the StLU activation function is designed for low-bit algorithm. The training accuracy convergence speed is very fast, and it even exceeds the convergence speed of software. The number of weight updates, online training time, and power consumption was also reduced. However, the overhead of using ADC/DAC in a crossbar array and the device-to-device resistive variability applied to a wide and deeper CNN's training accuracy are not explored extensively.
[28]	Atreya Majumdar, Université Paris-Saclay, France (2021)	TiN/HfO_x (10 nm)/Ti. (10 nm)/TiN	Conv384, Conv384, MP, Conv768, Conv768, MP, Conv1536, Conv512, MP, FC(1024-1024-10)	MNIST / CIFAR-10	Pytorch Framework of BNN with neuronal activations and synaptic weights takes binary values (+1 and −1). A simple XNOR operation replaces the product of the activation and the weight	A weak RESET regime is attractive for learning, as it allows tuning the resistance of the devices with remarkable endurance. An RRAM model of the weak RESET process in HfO_x RRAM integrated within the PyTorch is used to train low powered deep learning framework. Here, a shallow DNN was used to study the impact of RRAM variability on the training accuracy; a practical DNN consisting of much-complicated convolution layers, which tends to explode the RRAM variability and extensively reduces the prediction accuracy.
[29]	Qiwen Wang, University of Michigan, USA (2021)	RRAM non-ideality model (conductance resolution of 4-bits, on/off ratio down to 10, device conductance variation of 1.56%, read current assumed to be 3μA and 8-bit ADC) and array size of 256×64	(1^{st}) Simple network : CNN1 3×3×1×22, MaxPool 2×2, CNN2 3×3×22×27, MaxPool 2×2, CNN3 3×3×27×64 Stride 2×2, MaxPool 4×4 and Dense 64×10. (2^{nd}) VGG-block-based and (3^{rd}) ResNet WRN-16-8	MNIST / CIFAR-10	Training topologies – corresponding to different levels of architecture-aware training. (Level-1: Quantization- aware training, Level-2: device-aware, Level-3: tile-aware and Level-4: Conductance weight mapping)	The studies show the effect of the RRAM ON/OFF ratio, ADC characteristics on a crossbar array, and RRAM programming variations on the training network accuracy. Here, comparatively small RRAM array size is used to mimic the device-to-device variability.
[30]	Kaizhong Qiu, Tsinghua University, China (2021)	SPICE RRAM Model	LeNet / VGG / AlexNet	CIFAR-10	MNSIM 2.0 based training in memory architectures	A performance modeling framework for memristor-based training-in-memory architectures (MNSIM-TIME) is proposed to help architecture designers evaluate the RRAM performance in a CNN training network. The framework was tested for varying RRAM imperfections and the prediction accuracy on various CNN networks (LeNet/VGG/AlexNet). The SPICE RRAM model was not considered for varying current compliance to simulate a real-world device with different variability switching imperfection for the given current compliance and its impact on the training performance.

TABLE 3. *(Continued)*. **Listing of other RRAM based Edge CNN learning simulation reports focusing on the variability induced performance degradation.**

[31]	Shimeng Yu, Georgia Institute of Technology, USA (2021)	AlO_x/HfO_2, Ag:a-Si, TaO_x/HfO_x	VGG-8	CIFAR-10	In-situ training DNN model on extended DNN+NeuroSim framework	The challenges associated with in-situ training are presented. A hybrid precision synapse that combines RRAM with volatile memory is designed and evaluated at the system level to support accurate and fast in-situ training and enable subsequent inference in an integrated platform. The system uses MUX-based ADC's to reduce the chip power consumption; however, exploring digital RRAM logic will improve by undoing the mixed-signal scenario.
[32]	Hongwu Jiang, Georgia Institute of Technology ,USA (2020)	TSMC's 40nm RRAM and Intel's 22nm RRAM	VGG-8 network	CIFAR-10	Mixed-precision RRAM-based in-memory Training architecture (MINT) – Modified NeuroSim	The simulation studies feedforward, back propagation, weight calculation, and update in an analog crossbar array in-memory architecture with ADC overheads addressed by using RRAM array for MSB and LSB on a regular memory array. The study does not quantify the RRAM variability and its impact on training accuracy and efforts.
[33]	Yuyi Liu, Tsinghua University, China (2021)	1-kb analog RRAM (TiN /TaO_x/HfO_x/TiN), quantum point contact (QPC) model	VGGNet	CIFAR-10	Device-to-system simulation framework with embedded compact relaxation model to benchmark the CIM system	A compact relaxation model for analog RRAM was proposed based on the statistical measurement results. Non-idealities that affect on-chip training, including weight update, were studied. CIM array is used for vector-matrix multiplication (VMM). The impact of RRAM variability and the ADC overhead is not fully considered.
[34]	Kartik Prabhu, Stanford University, USA (2022)	A 40-nm CMOS, Foundry on-chip resistive RAM (RRAM) macros	ResNet-18	ImageNet	CHIMERA DNN accelerator is fabricated in 40-nm ultra-low-power CMOS technology with a total die area of 29.16 mm^2 (2 MB of RRAM for 37% of the total die area, with SRAMs and DNN accelerator, occupying 19% and 21%, respectively and the remaining area (23%) contains the RISC-V core)	Non-volatile DNN chip for both edge AI training and inference using foundry on-chip resistive RAM (RRAM) macros and no off-chip memory, fabricated in 40-nm CMOS. Here, an incremental edge AI training algorithm, called *low-rank* training helps overcome RRAM write energy, speed, and endurance challenges. However, IoT application requires much lower current compliance RRAM with an undesired memory window overlap for training and chip performance assessment.
[Our Methodology]		TiN/HfO_2/Hf/TiN	Modified MobileNET (10 layers with filter sizes of 32,64,128 and 512)	Traffic Sign Recognition Database (TSRD)	3-Layered simulation framework; (1) High level 10-layered CNN implementation with Python+Tensorflow, (2) Verilog HDL based FP32MUL and FP32ADD (32-bits Floating Point adder and multiplier) circuits constructed with NAND gates of 1T2R structures, (3) Digital Look-Up-Table (LUT) model for RRAM variability	Our study aims to shows the impact of RRAM variability in an in-memory computation circuit such as NAND / NOR gate and further used to construct an edge learning system on a practical CNN network, by which the learning accuracy trend impacted by variability is analyzed for different RRAM current compliance in a digital domain.

hardware-based reprogramming scheme uses 1T3R, 1T5R, and 1T7R structures for RRAM-NAND and RRAM-NOR gate applied to in-memory architecture.

The MCBC Layer acts as a special shift register, where the (1-bit) input data is shifted-in serially into the memory array (3-bits or 5-bits or 7-bits), and outputs a single bit of

data matching the most occurred data from the input serial stream, at any given instant of time as shown in Fig. 1. The shift registers are configured with a 3-bits, 5-bits, and 7-bits memory array, and each of the memory elements is connected in serial. For the given 3-bits, 5-bits, and 7-bits configuration, the register takes serial input data sizes of 3, 5, and 7 respectively. Upon reaching the maximum array size, a single output is generated with the maximum occurred input data from the array. Total clock cycles of 3, 5, and 7 are required here for the above-described configurations to generate a single output. We refer to the MCBC logic scheme as *Repetitive Cycles* (RC), and it is further used in our learning simulation framework to enhance and study the RRAM variability trend while applied in an in-memory computation circuit. Henceforth in our discussion, RC represents the depth of the underlying shift registers array size used in the MCBC ranging from 3 to 7. Following the functional simulation of MCBC, we have shown a practical implementation of the MCBC circuit using the transistor and RRAM logic in Section III.B. We have also presented a practical analysis of MCBC using the 65nm SPICE Transistor model and resistor circuit.

B. VARIABILITY STUDY IN NEUROMORPHIC LEARNING CIRCUITS ON THE EDGE

Training a CNN is an inevitable process, and significant studies have been conducted to move the training process from cloud to the edge system, using the low-powered in-memory computation architecture by overriding the excessive energy required to handle the memory wall during the memory read-write operations. Table 3 summarizes various edge-based in-memory studies conducted for low-power IoT applications. A study conducted by Feng et al. in Ref. [26] shows the OxRAM device stack used to construct a neural network of size 144-40-10 to train the MNIST dataset using a hybrid compute-in-memory (CIM) architecture illustrating the CNN model accuracy impact for the given RRAM device. In another study, a similar oxide device exploration was conducted by Li et al. [27] on a 784–400–10 MLP structure to emulate an FPGA+ARM based RRAM crossbar array to train the MNIST dataset and benchmark the prediction accuracy drop versus RRAM device variability. Adding to this list, a (10nm) HfO_x/Ti/TiN resistance distribution data set was used to build a shallow DNN network using a Pytorch learning framework with neuron activation that takes a binary value ($+1$ and -1) with a simple XOR operation scheme by Majumdar et al. in Ref. [28]. These studies are conducted on a simple neural network, whereas a practical network would comprise of deep hidden layers with tightly packed convolution operations to achieve a required prediction accuracy. Therefore, exploring a simulation framework with practical and deep NN layers will be useful in understanding the trend of RRAM variability for a real-world application.

The RRAM non-ideal model performance was studied by Wang et al. in Ref. [29] using a deep practical CNN such as VGG, and a ResNET architecture was used to train the MNIST and CIFAR-10 datasets. Here, (1) quantization aware, (2) device-aware, (3) tile aware, and (4) conductance weight mapping architecture are used to study the impact of the RRAM programming variability on the training network. A second study by Qiu et al. [30] was conducted on a deep practical CNN (LeNET/ VGG /AlexNET) using a SPICE RRAM model. A CIFAR-10 dataset was used with MNSIM2.0 based Training-In-Memory simulation framework to perform training at the edge and analyze the CNN prediction accuracy. The two studies aim to show the device variability trend by using RRAM models that use a small memory array. Hence, such a training framework can be tested for different RRAM current compliance with varying memory window overlap. The study with varying RRAM current compliance is necessary to quantify the RRAM variability and its impact on the training accuracy on a different current compliance scale when it is considered for an IoT application at very low power.

The RRAM crossbar array shows interesting multiplication and addition operations implemented using Kirchhoff's law to apply convolution operations of CNN. The crossbar arrangement uses comparatively fewer resources. However, the overhead of using external peripherals such as the ADC/ DAC to read and write the data on the analog crossbar array is significant in terms of power and area. The CNN is a parallel architecture in nature, where the total number of parallel processing computational elements define the throughput latency. Hence the primary constraints for designing an IoT system are defined by the total available processing power and chip area occupied by the parallel processing elements, that in turn define the latency of the system. The study conducted by Yu et al. in Ref. [31] using TSMC® 40nm RRAM technology and Intel® 22nm RRAM technology was used to build a VGG-8 NN and trained on a CIFAR-10 dataset using a modified Neurosim, with the intention of optimizing the use of ADC by using a MUX based ADC. The same group utilized a mixed RRAM design with RRAM memory designed for MSB bits, and a regular memory used for LSB bits as shown in Ref. [32]. Another work by Liu et al. in Ref. [33], uses a 1Kb quantum point contact oxide RRAM model to build the VGGNet on a CIFAR-10 dataset and applied on the device-to-system simulation framework to validate the on-chip training performance. A similar edge-based training analysis was conducted by Giordano et al. in [34], on a 40nm-CMOS foundry on-chip RAM used on ResNet-18 with an ImageNet dataset effectively demonstrating the training accuracy performance. Considering the above studies, it is evident that exploring the use of RRAM-based in-memory computation system on a digital domain will be useful to get rid of the ADC/ DAC overheads with a synchronized global digital clock driving the system for a more organized and controlled operation to achieve high speed digital in-memory computation logic. Hence, we focus on designing and operating the RRAM as digital memory to harness the

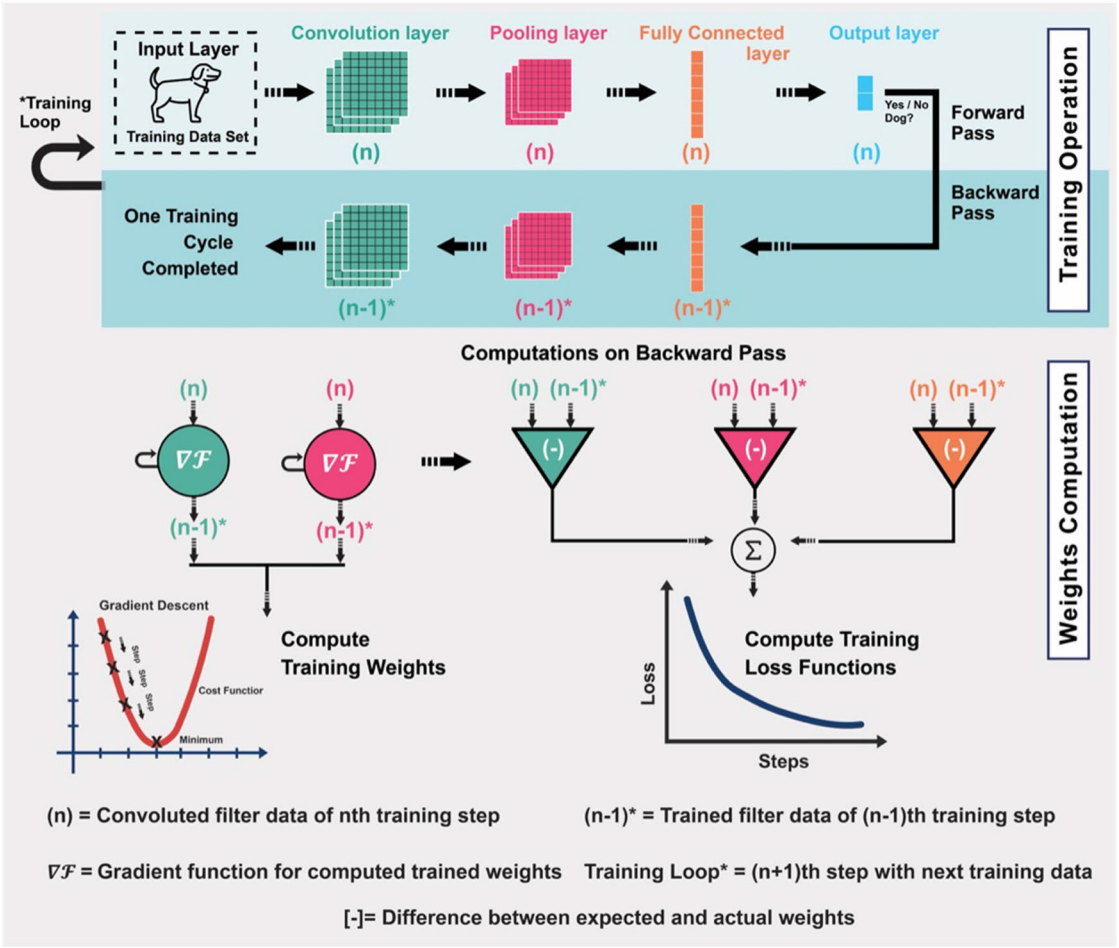


FIGURE 2. The various steps of a CNN training operation are shown as forward pass and backward pass, both executed in a loop to compute the training weight and loss function. The forward pass consists of transformation and mapping operations such as convolution, pooling, fully connected, and softmax, whereas the backward pass computes the weight difference between the 'n' and 'n-1' loop to derive the optimized weights and training loss or error.

advantage of avoiding the peripheral circuit overheads and achieve high external noise immunity when designed as a digital system.

Our current work aims to build a full-sized practical CNN with 10 layers (filter sizes ranging from 32 to 512) of a modified MobileNET trained with traffic road sign data set (TSRD) with a three-tier abstraction of the simulation learning framework - (1) High level 10-layered CNN implementation with Python+TensorFlow; (2) Verilog HDL based FP32MUL and FP32ADD (32-bits Floating Point adder and multiplier) circuits constructed with NAND gates of 1T2R structures for the logic computations; and (3) a Digital Look-Up-Table (LUT) model for encoding RRAM variability.

The novelty of our work is the methodology of edge learning framework (forward pass) using digital RRAM-NAND/NOR universal gates integrated with Maximum Count Binary Comparator Layer (MCBC) to control the impact of RRAM variability and to quantify the RRAM variability on the CNN training prediction accuracy for varying low device current compliances ranging from 5 to 50μA for ultra-low power IoT applications. We have also demonstrated a practical implementation of the RRAM-NAND-based standard cell and simulated it using the SPICE model. Today we see models such as TinyML etc., targeted for embedded processors performing well for a low footprint application. While looking into such embedded specific CNN architecture, the resource reduction is achieved by pruning and shrinking the deep neural network size to fit the small memory computing device. Such systems are popular for binary result conditions, such as ''visual wake words'' saying YES or NO, to predict the trained label from the input image. However, in a vision-guided autonomous system, a CNN navigation guidance system requires a considerably higher feature mapping than the simple YES or NO prediction conditions, which are very well achievable using a deep CNN such as MobileNET, which uses a wide range of filter sizes and intense convolution operations. The large-scale datasets such as CIFAR100 and COCO are structures with a few hundred generalized categories, whereas the autonomous application trained with the traffic road sign dataset, will aid in developing a more optimized navigation system with a higher recognition rate by training the CNN with a specific ''traffic road sign'' dataset.

Section II presents the CNN simulation framework used to train a model to identify the road signs, wherein the convolution operations are implemented with RRAM variability encoded floating-point multipliers and adders and for which, the prediction error loss is computed between the hardware and software pipelines. Section III discusses the results and trends obtained using the learning framework for different current compliance RRAM datasets and the prediction trend variation by applying the Maximum Count Binary Comparator Layer to each RRAM NAND gate. We conclude our work in Section IV after a summary and inference based on all the analyses carried out.

II. SIMULATION METHODOLOGY FOR DESIGN AND IMPLEMENTATION OF RRAM BASED IN-MEMORY COMPUTATION MODELS APPLIED TO TRAINING CIRCUIT

A. FUNDAMENTAL ELEMENTS OF CNN TRAINING OPERATION AND WEIGHT COMPUTATION

1) TRAINING

The Training is a constructive process that involves parallel computation to be performed on the input images by applying convolution and linear transformation operation for feature map extraction and finally calculating the loss function during the Backward Pass to update the learning weights as shown in Fig. 2. The training process is classified as Forward Pass and Backward Pass operations; the former involves having a batch of input images to pass through the given network in a forward direction, and the predicted output is then compared with the actual labels. By knowing the closest predicted value from the actual label, the weights of the networks are adjusted using the Backward Pass operation, during which the network predicts close to the actual labels when the images are seen the next time. A similar process is repeated for as many different batches of available images, and this is regarded as an *epoch*. The training process is performed for as many epochs as necessary to achieve the desired accuracy level, and the five steps involved are (1) Prepare the images in batch for training, (2) Pass the batch to the network – Forward Pass, (3) Calculate the loss between predicted and actual data, (4) Compute the gradient for the loss function and update the weights to reduce the loss, (5) Loop – repeat 1-4 for 'n' iterations to reach the required minimum error.

2) FORWARD PASS

The Forward Pass is a pipeline with stages of chained matrix transformation functions performed to extract features from the input image to classify it against the trained label class. The various transformation functions are convolution, ReLU activation function, Maxpool operation, fully connected layer and Cross-Entropy coding (Softmax). Each layer in the network has its own transformation, and all individual layers constitute the total transformation of the given network. The objective of this operation is to transform and map the inputs to the right output class with minimum possible error [35].

3) CONVOLUTION

The convolution is the sum of the products (SOP) matrix operation of the input image to its local neighbor or filters. By sliding the filter matrix of size ($n \times n$) over the image matrix ($m \times m$), and the SOP between these two matrices generates the feature map, with the consideration that the size of 'm' is greater than 'n'. The *stride* is the size of the step the convolution filter moves each time over the image matrix [36].

4) ReLU-MAXPOOL

The rectified linear unit (ReLU) activation function is a linear functioning operation whereby the output is equal to the given input if it is positive, while for all other cases, the output to be maintained at zero [37]. The ReLU is a popular function used among many neural networks as they perform better for a broader class of data sets. Max Pooling is a down sampling approach to reduce the computation power and avoid network overfit [38].

5) FULLY CONNECTED LAYER

The feed-forward neural network, known as the Fully Connected layer, forms the last few layers in the CNN and receives flattened or 1-dimensional data array from final pooling or convolution layers [39].

6) SOFTMAX

The Softmax is a Cross-Entropy activation function that predicts a multinomial probability distribution where a class member requires no more than two class labels at the last or output layer of the neural network [40].

7) BACKWARD PASS

The Backward Pass is an intense computation operation where the weights are updated based on the learning rate and by computing the past and present weight gradients [41]. Adam optimization and Stochastic Gradient Descent (SGD) are the commonly used gradient functions and the SGD is used as the gradient calculation algorithm in our current training framework.

8) LOSS FUNCTION

The Loss Function, which is one of the critical operations in the training process, is the difference between the expected value and the predicted value. The *loss* is used to calculate the gradient, and they are further used to update the weights of the network layers [42].

9) ACCURACY METRICS

The confusion matrix is one of the more useful accuracy matrices used to show where the trained model became confused in predicting correct labels and help us re-train the model with more related data sets to improve prediction accuracy.

FIGURE 3. Illustrates 10 different road sign categories labeled and used to train the CNN model.

B. OVERVIEW OF THE TRAINING DATASET AND CNN USED FOR SIMULATION

Autonomous vehicles are developed for safe roads by reducing human error. However, safety here is a function of how precisely the navigation system identifies and how fast a decision is being made with high uncompromisable accuracy. The navigation system must process multi-dimensional parameters such as vehicle speed, road lane identification, and position of other neighbors (vehicles and civil structures), road and traffic signs, weather conditions, etc., to make the right decision to navigate. The identification of traffic signboards is one of the primary essences of autonomous systems, and we have used the Traffic Sign Recognition Database (TSRD) for our training simulation and analysis. The TSRD is extracted from Beijing Jiaotong University, China's Traffic sign database repository (repo), and it consists of 6000 images, of which 4000 images are used for training and 2000 are test/validation images. The images are classified into 10 categories, such as *No_Entry, No_Left, No_Right, No_Parking, Stop_sign, Entry, Left_Turn, Right_Turn, Parking, and Pedestrian Crossing* as shown in Fig. 3. There is an eleventh category called *Others*, whereby any sign which does not fall under the previous 10 categories will be grouped under this, and all our simulation results are shown for the main 10 categories only. The first 5 traffic sign categories are the mandatory signs that signal the navigation system to actions that represent "not to do" or "can't do" and are designated with a red color border *(No_Entry to Stop_Sign)*; whereas the next 5 sign categories include "can do" symbols with blue and white colored signs. The 10 categories of labelled images were trained with 50K steps of 32 epochs using Stochastic Gradient Descent weights optimized for MobileNET CNN. In real-time, the front-facing camera and traffic sign recognition system used to complement the navigation system must cater to filter the background scenes and should be able to read signs from an angle, faded text and broken signboards, ensuring consistent readability during high vehicle speed, etc. All of these conditional parameters and considerations are out of scope of this study and considered to be problems with already available solutions. Here, we assume the camera system has inbuilt pre-processing modules for image enhancement. The backend CNN system receives a good quality image processed data set that is to be classified into any one of the trained 10 categories with high accuracy.

The MobileNetV1 [43] CNN is designed to operate on small-footprint embedded devices with reduced model size and complexity. The CNN operates with two different convolution modules, which comprise of the Depth Separable Convolution (DS) followed by the Pointwise Convolution (PW). The DS performs a single convolution on every channel rather than combining all three and flattening it. The DS is a 3×3 convolution operation, and all the outputs are combined with a single 1×1 PW convolution in a single step. The DS is constructed with two layers, one for filtering and the other for combining all the outputs; hence this factorization has a more significant reduction of computation and is suitable for IoT applications. The MobileNet is a 30-layered architecture with a combination of stride-2 convolution block, depthwise – pointwise convolution block, fully connected layers, and the softmax classifier. The convolution layers are intense computational layers and consume more than 80% of the overall device power. Hence, we confine our simulation and study on the convolution layers and Fig. 4(a) shows the slightly modified first 10 convolution layers of MobileNet, that we have used for our device variability simulation framework. Fig. 4(b) shows the core modules of every convolution layer, such as Conv module configured with a 3×3 convolution, followed by the Batch Normalization (BN) and the ReLU layers. In contrast, the second core, Conv_dw module, configures a 3×3 convolution, BN, and is further applied with a 1×1 convolution, BN and ReLU. The Conv layers operate with a stride of 2 while Conv_dw operates with a stride of 1. We have considered using the 10 convolution layers in 5 different groups as shown in Fig. 4(a) with group 5 comprising of layer 1 and 2 with a filter size of 32; group 4 including layers 1 to 4 with a filter size of 32 and 64, and further groupings are performed similarly all the way up to group 1, with 10 layers and filter sizes from 32 up to 512. The trained weights in Groups 1-5 are represented in a 32-bit floating-point format, which consists of the mantissa (23-bit), exponent (8-bits), and signed bit (1-bit).

C. LUT-BASED RRAM VARIABILITY ENCODING SCHEME AND NEUROMORPHIC SIMULATION FRAMEWORK TO TRAIN ROAD SIGN DATASETS

The TiN/HfO_2/Hf/TiN RRAM element's electrical characterization data set showing device-to-device and cycle-to-cycle variability is considered for our in-memory 1T2R gate logic simulation. The RRAM resistance data is extracted from Fantini et al. [15] with a varying I_{comp} switching data for 5, 10, and 50 μA. The resistive element was fabricated on a 65nm CMOS process and with an oxide thickness of 5 nm. Since we intend to examine the low power IoT regime, we confine our simulation to an OxRAM device operating in a low power regime and do not consider a CBRAM device which has comparatively higher power due to the metallic nature of the switching conducting filament

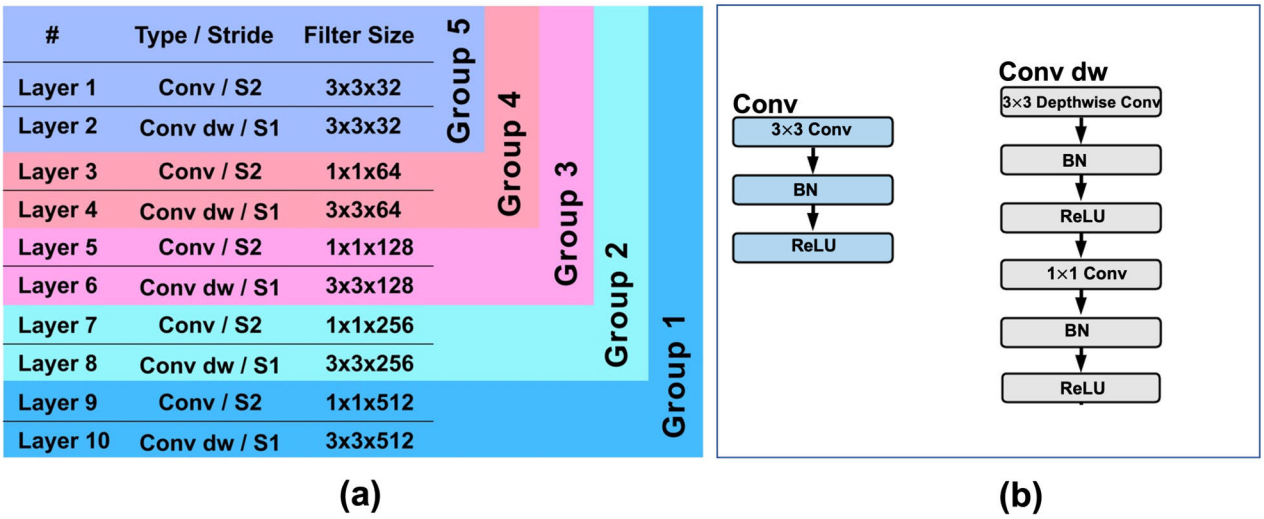


FIGURE 4. (a) The 10 convolution layers with filter size ranging from 32 to 512, grouped into 5 groups as shown and used in the simulation framework; (b) Every group further comprises of core modules such as Conv module (3 × 3 convolution, Batch Normalization (BN) and ReLU) and Conv dw module (3 × 3 convolution, BN, ReLU, 1 × 1 convolution, BN, and ReLU).

[44], [45]. The objective of our simulation framework is to demonstrate the configuration of an RRAM-based device in a CNN, where the device is operating in a digital domain. The device's Low Resistance State (LRS) is considered as "Logic-0" state, and the High Resistance State (HRS) is considered as the "Logic-1" in our simulation framework. We have proposed a practical circuit in Section III.B, demonstrating a digital RRAM-based gate used to build the convolution operations, as digital circuits with high noise immunity and also comprising a digital synchronizing clock for precise and high-speed operations compared to analog circuits. We perceive RRAM devices to be configured on a crossbar array, demonstrating a multi resistive state that has greater power-saving and is less dense with smaller chip area. In spite of all its advantages, the given crossbar design exhibits a sneak path effect. Moreover, additional peripheral circuits such as ADC and DAC are employed to read and write from the RRAM device in a crossbar array. Today's image acquisition and preprocessing modules operate in a digital pipeline with a Digital Signal Processing (DSP) backbone, where the images captured in a CMOS sensor are transmitted to Image Signal Processing module for preprocessing, followed by CNN recognition and finally to a post-processing module for final presentation. As such, the entire pipeline is digitized.

An analog crossbar CIM (Compute-In-Memory) RRAM system becomes challenging to fabricate as a mixed-signal processing SoC (System On Chip) for a vision-based neuromorphic application. The I_{comp} corresponding to LRS and HRS logarithmic resistance distribution extracted from Fantini et al. [15] is shown in Fig. 5(a). Plotting the resistance as a normal distribution shows an overlap region between the LRS and HRS state, as shown in Fig. 5(b), and by considering the LRS as Logic-0 and HRS as Logic-1, these overlap regions show the possibility for false-0 and false-1 regions, which result in the device level variability getting translated into the final circuit. The false-0 and false-1 regions represent the memory window overlap region. For higher current compliance, the overlap is minimal or negligible. For the lower current compliance, the overlap is significant and critical. Therefore, quantifying the impact of the device variability for different current compliances is critical to understand the final usability of the device.

We propose a Look-Up-Table (LUT) model to encode the RRAM variability into an RRAM-based NAND and NOR gate logic. The universal gates are the basic building blocks for any digital circuit. With the RRAM NAND and NOR logics, we aim to construct a convolution framework to demonstrate the 10-layered MobileNET. The primary consideration of the LUT model works by encoding the resistance region of 0 to $\text{Log}_{10}(0.5\text{R})$ as Logic-0 and the next half of the resistance distribution region, which is $\text{Log}_{10}(0.5\text{R})$ to $\text{Log}_{10}(\text{R})$ as Logic-1 from Fantini et al. [15] data set for 5, 10, and 50μA respectively. A 1T2R structure is further proposed as shown in Fig. 5(d), where two parallel RRAM devices are connected to the gate terminal of the MOS transistor. Here, the RRAM resistance threshold controls the transistor gate operation in a sequence to demonstrate the NAND and NOR functionality.

1) NOR GATE

When input A and B are at logic-0 (Zero voltage), the transistor is not gated and stays at OFF state, as a result of which, the output is at Logic-1 or close to the VCC (Supply voltage). Considering the scenario where input A or B or both A and B are supplied with a Logic-1 an equivalent current resulting from the parallel resistors flows into the gate of the transistor, which supplies the required gating voltage to turn ON the gate. This will result in a low resistance path from the

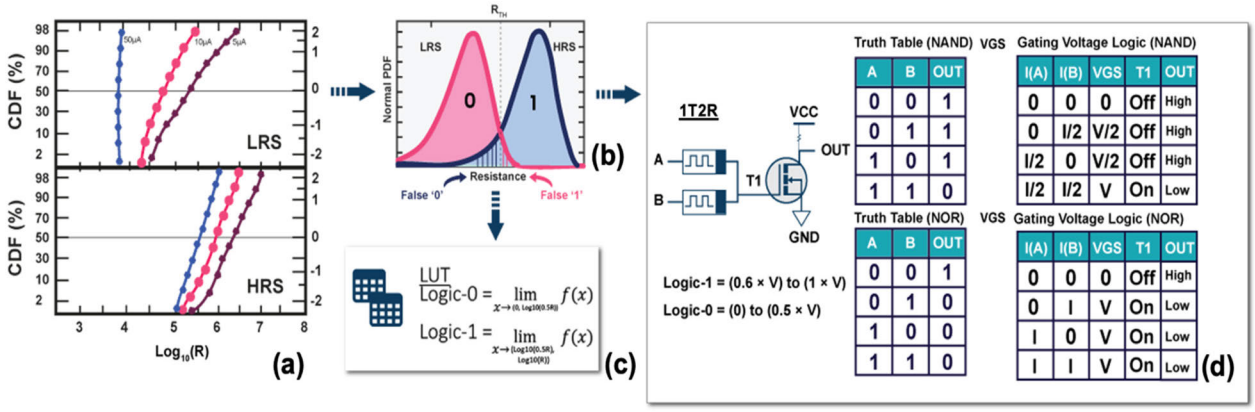


FIGURE 5. (a) Cumulative distribution function (CDF) of a 65nm OxRAM shown for 5,10, and 50μA I_{comp} extracted from Fantini. et al. [42], where the distribution is shown for LRS and HRS, (b) Representation of CDF converted to a normal probability distribution function (PDF), showing the LRS and HRS memory window overlap region (False-1 and False-0). (c) A Look-Up-Table (LUT) encoded to a digital domain as Logic-0 and Logic-1 using the RRAM LRS and HRS resistance distribution with the range of {0,Log_{10}(0.5R)} for Logic-0 and {Log_{10}(0.5R),Log_{10}(R)} for Logic-1. (d) Illustration of 1T2R NAND and NOR structures, which uses the encoded RRAM variability to reproduce the Boolean truth table.

VCC to ground (GND) and cause the output to be maintained at Logic-0 for the given input conditions demonstrating a NOR operation.

2) NAND GATE

The operation is vice versa for the NAND gate, where the resulting voltage from the two input resistors reaches the gating threshold only when both the inputs are maintained at Logic-1. The gating threshold is not met for the other input sequences to turn ON the transistor in order to demonstrate the NAND operation.

The RRAM-based NAND and NOR circuits demonstrated here may work theoretically, but in order to make these circuits a practical working system, we need to add a gate biasing resistance and carefully choose the input resistance range to achieve the gating sequence logic of the NAND and NOR gates. We have simulated and shown the various resistance values for the inputs and biasing resistance in Section III.A.

The learning process used here is constructed as a three-stage simulation framework using TensorFlow and Python programming, as shown in Fig. 6. Stage 1 is the Forward Pass of the MobileNET CNN with 10 layers of varying and increasing filter size ranging from 32 to 512 and with Depth-Wise and Point-Wise convolution modules, as shown in the layer group of Fig. 4. The Stage-1 Forward Pass consists of two parallel CNN computation pipelines called the Software (SW) and Hardware (HW) pipelines. The SW pipeline was purely implemented with the TensorFlow framework while the HW pipeline was implemented with a combination of Python, and Verilog HDL with the RRAM variability encoded NAND gates used to perform the convolution operations in all the 10 layers of the CNN. Every NAND gate in the HW pipeline is implemented using 1T2R structure as shown in Fig. 5(d) and furthermore, the two RRAM devices in the NAND gate are encoded with the varying resistance data using a LUT model, where the 0 to Log_{10}(0.5R) range defines Logic-0 and Log_{10}(0.5R) to Log_{10}(R) gives the Logic-1.

The Hardware pipeline is programmed with Python, Verilog-HDL, and the data exchange between the two programming languages is handled through the CSV (Comma Separated Version) file system as shown in the block diagram in Fig. 7. Here, the Python program implements the 10 layers of the MobileNET architecture with varying filter sizes, the sub-routine of the convolution operations inside the 10 layers calls the Verilog program, where a combinational circuit of 32 bits floating point multipliers and adders are used to perform the convolution operation. As discussed in our previous work [46], the general Verilog Convolution implemented includes 7-layers of abstraction that exposes all the NAND gates used for the convolution operation. Hence, by looking at the NAND structure, we substitute our proposed 1T2R structure to obtain the truth table for the NAND as in Fig. 7. The low-level subroutine in the Verilog code defines the 1T2R functionality for the NAND gate and the simulation framework uses a LUT file with 5000 varying RRAM resistance data set for LRS and HRS, where the framework randomly reads and assigns the resistance of the 1T2R value based on the LUT file. Thus, the false-0 and false-1 states from the RRAM data set are encoded into the FP32MUL and FP32ADD modules through the 1T2R NAND structure, which in turn affect the convolution operation performed in the HW pipeline.

Stage 2 is the backpropagation process where the error value is calculated as the weights difference between the expected and computed image and a Stochastic Gradient Descent (SGD) is implemented to adjust and update the training weights. Here, in Stage 2, the entire process was implemented using TensorFlow and Python, and no hardware logic (RRAM NAND) was used. Finally, Stage 3 will be the simulation framework to predict the difference in calculation between the SW and HW pipelines from Stages 1 and 2. The difference between the prediction percentage

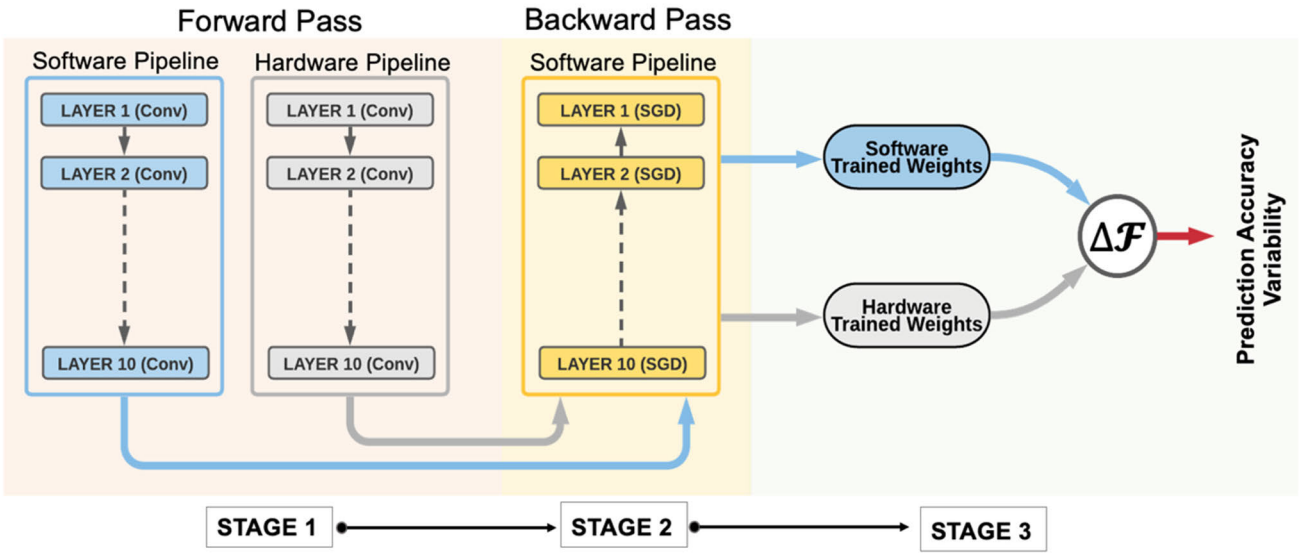


FIGURE 6. A TensorFlow Python and a Verilog HDL-based three-stage simulation framework to validate the RRAM variability encoded CNN training network. STAGE-1: constructed with forward pass software pipeline – SW (actual logic) and hardware pipeline – HW (RRAM encoded logic). STAGE-2: Backpropagation using stochastic gradient descent (SGD) to update the trained weights. STAGE-3: Prediction error computation between the accuracy obtained between SW and HW pipelines.

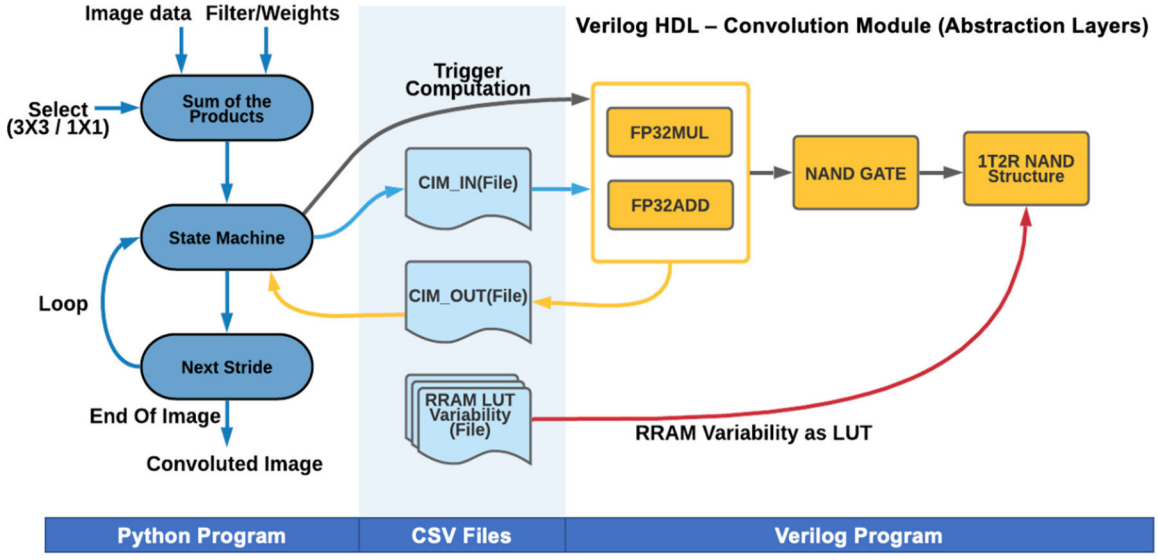


FIGURE 7. Block diagram illustrating the training data and convolution operation flow from left to right (Python program to CSV files to Verilog HDL Program and finally back to Python program). The CNN is implemented with Python, the convolution operations are implemented with Verilog HDL (FP32MUL and FP32ADD) and the data exchange (weights and RRAM variability encoded LUT) is performed using file operations (CSV file format).

(confidence) using SW and HW trained weights quantifies the variability inherited from the underlying RRAM NAND Logic (1T2R). Every NAND logic is coupled to an Arbitrary Logic (MCBC) layer to improve the RRAM variability, as in a computational logic pipeline, even a few gate failures can result in a massive system calculation failure. The MCBC layer works like an inter cache memory which takes 3 to 7 inputs and outputs the best or most occurred data among the input at any given instance. The MCBC layer decreases the probability of device variability/failure to enhance the system performance. The prediction error reduction is quantified for three different configurations of MCBC (RC1, 3, and 7) in the following results and discussion in detail. A practical MCBC layer is demonstrated and discussed in Section III.B. The Python pseudo-code for the SW and HW pipeline is shown in Fig. 8, where Fig. 8(a) shows the 10 layers of convolution implemented with the specific filter sizes, and Fig. 8(b) shows the two different subroutines of SW and HW. The SW is

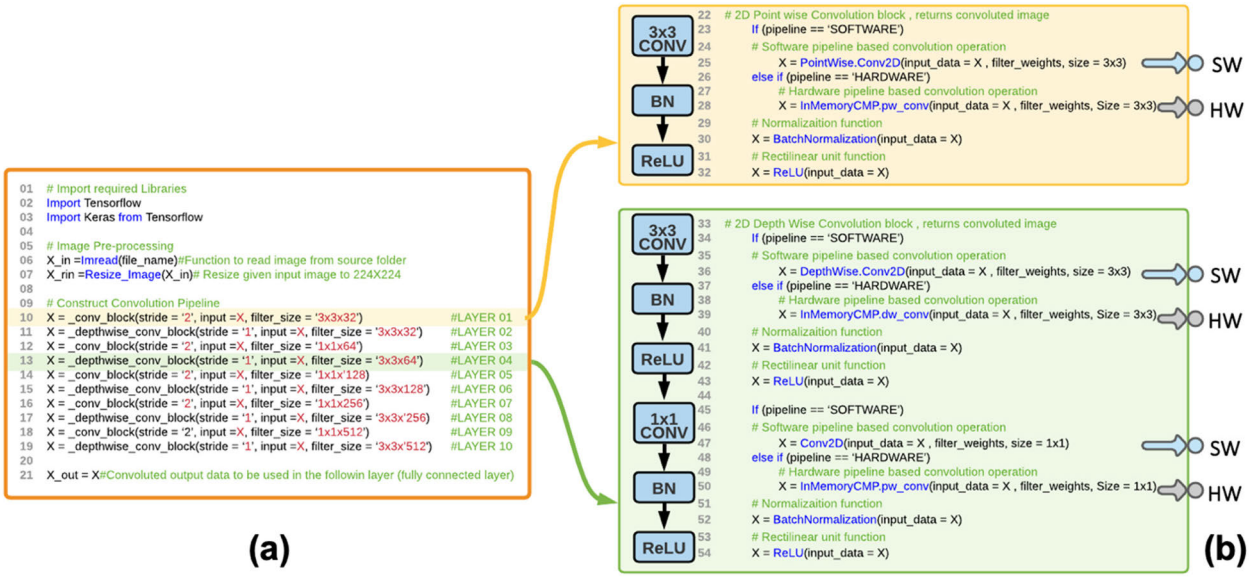


FIGURE 8. (a) Python pseudo-code showing the implementation of CNN by interconnected hidden layers as Python subroutine with different filter sizes ranging from 32 to 512. (b) Snippet of the SW and HW pipeline implementation, where the HW pipeline calls the RRAM variability encoded Verilog HDL to compute the convolution operation.

implemented with a direct TensorFlow library, and the HW subroutine calls the Verilog model to perform the convolution operations.

III. RESULTS AND DISCUSSION–PERFORMANCE OF RRAM BASED IN-MEMORY COMPUTATION CIRCUIT IN A CNN TRAINING SYSTEM

6000 different traffic sign images were used for the training framework, out of which 4000 images were used for training and the rest were test images. Stochastic Gradient Descent (SGD) weights optimizer and an error function were used for the purpose of configuring the framework to compute the training accuracy given by the loss function. Final trained weights were obtained from the 50,000 training steps resulting from different stimulation tests using the above parameter setup. Under various scenarios, the training test results obtained are plotted on the Y-axis while the corresponding training steps (epochs) are plotted on the X-axis, and the resulting trend will be examined in depth in the different cases as follows.

A. IMPACT OF DIFFERENT MAXIMUM COUNT BINARY COMPARATOR LAYERS, HIDDEN LAYERS AND TRAINING IMAGE LABEL CATEGORY ON PREDICTION ERROR FOR VARYING RRAM COMPLIANCE

The training accuracy trend is examined for the following parameters which include 10 convolution layers (Group 1), 10 different image categories, 10μA RRAM current compliance resistive encoded data set for the HW pipeline and four different Repetitive Cycles (RC) viz. RC = 1, 3, 5, and 7 as shown in Fig. 9. The objective of this analysis is to plot the variability in training trends for the different RC, as it is a known fact that by increasing the RC, the RRAM variability reduces as a result of probability. Here, we aim to quantify the effect of variability to the corresponding trend improvement in the final prediction accuracy for the increase in training steps. The given figure is generated by using a five-simulation results plotter, such as the SW pipeline computed software trained data (SD) and four different RRAM variability encoded HW pipeline logic with RC = 1, 3, 5, and 7, with the obtained training accuracy on the Y-axis and the number of training steps on the X-axis. As the training steps increase to 9000, the SD accuracy gradually increases to 12%, but thereafter, the SD accuracy took a steep increment to 88% when at 27,000 steps, and at 50,000 steps, the obtained training results are 92% accurate. We use the SD data set training accuracy as a benchmark to compare the four different hardware trained logics. As explained, the RC3 logic is performed by repeating the basic NAND operation three times with RRAM encoded data to choose the maximum occurred outcome from the three results, hence the entire convolution operations in the given 10 layers are repeated three times in total for RC-3. The same concept applies to RC-5 and RC-7, meaning the operations are repeated five and seven times respectively, while the RC-1 is simulated with no repetitive cycle. The 10μA RRAM data set has an approximate 10% overlap in the LRS and HRS resistive distribution and the impact on the training accuracy is plotted using the given overlap and different repetitive cycles. The RC-1 reaches a maximum of 15% accuracy on the 50,000th training step, with the accuracy almost reaching a steady oscillation state from 13,000 steps onwards. The RC-3, RC-5, and RC-7 follow a similar trend, reaching a maximum accuracy of approximately 30%, 60%, and 68%, respectively. The accuracy improvement from RC-3 to RC-5 is significantly higher than when compared to the improvement from RC-5 to RC-7. Hence, it is evident that the increment

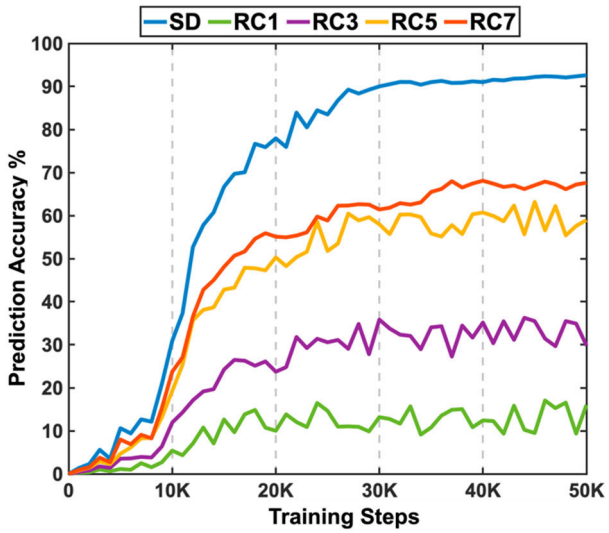


FIGURE 9. The prediction accuracy variation up to 50K training steps and four repetitive cycles (RC1, RC3, RC5, and RC7) is plotted. Simulation framework parameters are set as: Group 1 – 10 convolution layers, 10 image categories, and the 10μA RRAM resistance distribution encoded dataset.

in the RC does not translate to significant accuracy improvements and the contribution to the effects on the prediction accuracy lies more on the underlying RRAM resistance overlap range between the LRS and HRS rather than the RC. The idea of using repetitive cycles to improve RRAM logic element variability is necessary only for low RRAM current compliance since the repetitive logic tends to increase the power budget compared to the SW pipeline implementation for the higher I_{comp} devices.

The power consumption is compared for high RC - low compliance versus low RC - high compliance scenario considering the median LRS resistance for the 50μA and 5μA distributions from Fig.5(a) with memory operating voltage taken as 1.2V (CMOS Voltage). As known, more clock cycles are required for high RC as the same operation is repeated more times. The power difference is quantified for scenario A: - low RC - high compliance (RC1@50μA) and scenario B:- high RC - low compliance (RC7@5μA) using the standard power law equation with the parameters such as current (50μA and 5μA), voltage (1.2V), LRS median resistance ($\text{Log}_{10}(4)$ and $\text{Log}_{10}(5.5)$), and RC (1 and 7). Scenario B operates with 6-clock cycles more than Scenario A to perform a single memory operation (read); still, Scenario B consumes only 8.6% power of the total Scenario A setup.

Following the simulation of different RC logics, the accuracy trend between the HW and SW pipeline is estimated by reducing the number of hidden layers. The simulation starts out with 10 layers, followed by a reduction of 2 hidden layers for each simulation cycle. The layer reduction is performed by five different groups as shown in Fig. 10. Group 1 has the greatest number of layers and convolution elements, whereas Groups 2, 3, 4, and 5 are constructed with lower filter sizes ranging from 32 to 256. The filter size of the given 10 layers is 32 for layers 1 and 2, 64 for layers 3 and 4, 128 for layers 5 and 6, 256 for layers 7 and 8 and 512 for layer 9 and 10. The application of a 10-layered CNN in autonomous vehicles and process industry settings do function effectively while the backpropagation training performed well with a sufficiently large volume of the dataset, which is our basis for choosing or setting "10 layers" as our upper limit in our simulation framework to quantify the RRAM device variability on the prediction error rate for the layer configurations {2∼10}.

It is to be noted that a very deep CNN is not optimal from a hardware (RRAM-based in-memory circuit) perspective due to compounded variability as the number of hidden layers increases. Still, a very shallow network may not have high prediction accuracy due to the fewer feature extraction logics used to classify the input image. On the other hand, a full software implementation (without RRAM in-memory logic) prefers to use deep networks as much as possible to achieve high accuracy at the cost of high computing power. The color scale depth of the input image (RGB or Monochrome) also contributes to the computational intensity in every layer of the network. Hence, a delicate balance must be met for choosing the right network size considering the available power and required prediction accuracy trade-offs as defined by the end use application.

It is obvious that the reduction of the hidden layer count will increase the performance of RRAM logic elements; and in the recent past, there are several studies [47], [48] conducted on exploring the usage of tiny CNN modules for low power embedded applications with less number of hidden layers to perform simple recognitions in real-time. The references [47], [48] use a smaller number of layers {VGG-1-16 (6 layers) [47] and 5 layers with filter size from 32∼4 [48]} demonstrating image classification, and further, no benchmark comparison was conducted on the implementation methodology of these references with our simulation methodology. The study for different hidden layer count complements with using CNN on IoT low power applications by quantifying the achievable prediction accuracy with the given low power RRAM logical element. The simulation shown below was performed using the parameters that include 10μA RRAM I_{comp} HW pipeline; 5 repetitive cycles; 10 different road sign categories and 5 different hidden layer groups. The obtained prediction accuracy is plotted in the Y-axis and the number of training steps on the X-axis in Fig. 10. The network prediction accuracy decreases with the reduction of the hidden layers, since the overall feature extraction and mapping is reduced and more importantly, we observe that the training curve trend differences between the SW and HW pipeline also reduce as the filter size is lowered, as seen in Fig. 10, which shows the quantified impact of RRAM variability over the different filter size and computational logic. The SW and HW pipeline prediction accuracy with Group 1 (10 layers) achieved are 92% and 59% respectively for parameters of 10μA I_{comp} and 5 repetitive cycles. The SW and HW pipeline accuracy results of Groups 2, 3, 4, and 5 with the same parameters

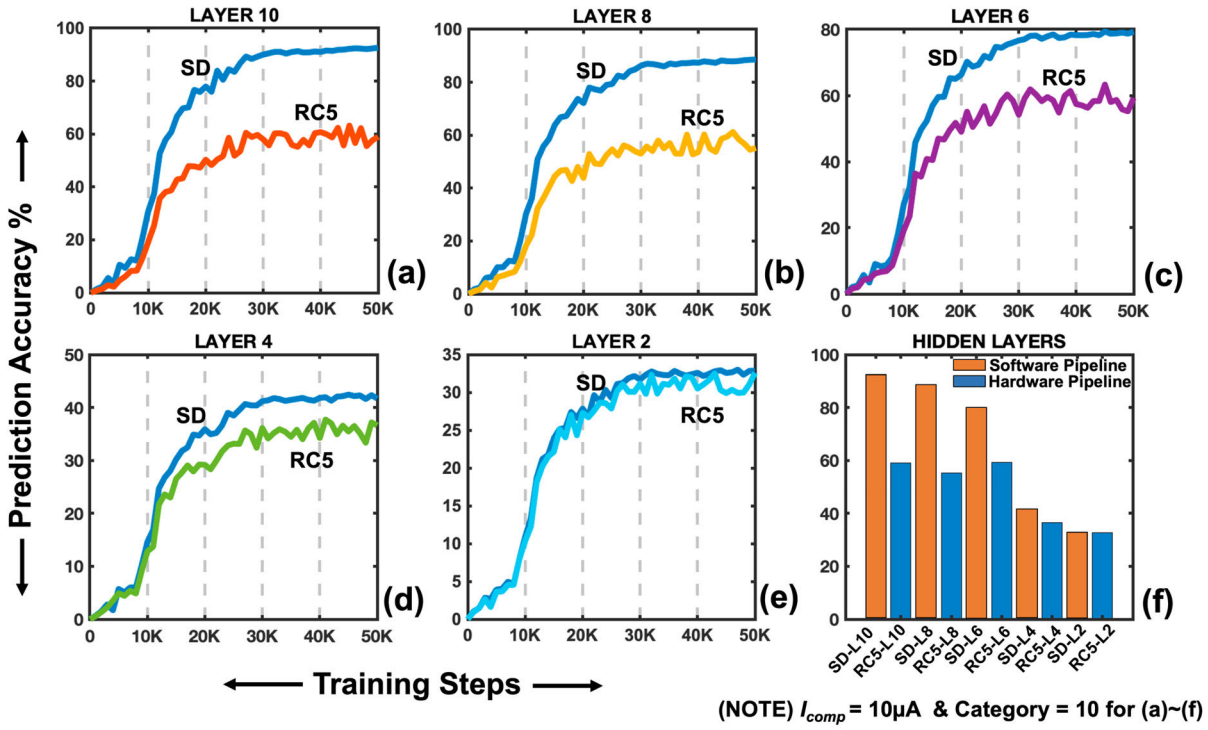


FIGURE 10. Training accuracy trend plotted for the varying hidden layer count from 10 Layers to 2 Layers with an arbitrary logic repetitive cycle (RC) of 5 and I_{comp} of 10μA. (a) Group 1 with 10 layers of filter sizes from 32 to 512, (b) Group 2 with 8 layers of filter sizes from 32 to 256, (c) Group 3 with 6 layers of filter sizes from 32 to 128, (d) Group 4 with 4 layers of filter sizes from 32 to 64, (e) Group 5 with 2 layers of filter size of 32, (f) shows the overall prediction accuracy achieved at 50K steps for Groups1 ∼ 5.

settings are recorded as (89/58) %, (80/60) %, (42/37) %, and (32/31) %, respectively. Interestingly, the SW pipeline accuracy decreased for lower number of hidden layers (as expected) resulting from the lower count of features mapped, while the accuracy of the HW pipeline increased with lesser number of layers; the SW prediction accuracy drastically reduced from 6-layers down to 4 and 2 layers due to the fact that such low number of layers have smaller convolution filter sizes of 64 and 32, respectively.

To be more qualitative, consider a practical application of an IoT-enabled battery-powered smart bin sensor with camera and inferencing modules deployed to assist the waste management industry in increasing the re-cycling index and contributing towards a sustainable environment. For the said application, RC5-L4 (Repetitive Cycle = 5 and CNN Layer = 4) from Fig. 10(f) is considered more suitable for a smart bin sensor with an in-memory edge computation system running on battery power.

The computation power of gradient calculation for the convolution operator's filter data is a function of the training image data set, the number of hidden layers, total training steps, and the total number of labeled categories. We consider regrouping our labeled categories used in the simulation and retraining the other training parameters such as image data set size and the number of hidden layers, keeping the training steps as a constant. The intention of the regrouped image category simulation is to further study the impact of RRAM logic element variability on the training process for computing the weights with highest achievable prediction accuracy. The training data set consists of 10 categories *(No Entry, Stop Sign, No Left, No Right, No Parking, Entry, Parking, Left Turn, Right Turn, and Pedestrian Crossing),* and based on the driving instruction/operation, the given 10 categories are regrouped into 6 category and 2 category groups, as shown in Table 4 for generating the training trend of SW and HW pipelines. The 6 categories group is formulated with the following driving instruction logic, (1) cannot move further *[No Entry and Stop Sign]*, (2) only allowed to go straight *[No Left, No Right]*, (3) do not stop *[No Parking],* (4) allowed to enter *[Entry and Parking],* (5) allowed to turn left or right *[Left Turn and Right Turn],* (6) Watch out for pedestrians crossing. An assumption made here is that there will not be any T-Junctions and crossroads while driving in order to resolve the tie between left or right direction sub-condition in (2) and (5). Note that category (3) ''No-Parking'' and (6) ''Pedestrian Crossing'' are not clubbed together in the 10-category and 6-category group. For the 2-category group, we classify the entire data sets into *''Can Do''* and *''Don't Do''* categories. The 2-category group setting is not practical to be considered for the use in the primary navigation system for autonomous driving as it is necessary for the recognition and features to be mapped to a much broader class/conditions for more accurate prediction and precise decision-making. Still, we intend to simulate the 2-category group for comparative study on the variability trend obtained in the prediction. However, the 2-category group-based trained weights can be used as low

TABLE 4. Training image label re-grouping for the three categories.

Category 10	Category 6	Category 2
No Entry	No Entry_Stop	Don't_Do
Stop Sign		
No Left	No_Left_Right	
No Right		
No Parking	No Parking	
Entry	Entry_Parking	Can_Do
Parking		
Left Turn	Left_Right Turn	
Right Turn		
Pedestrian Crossing	Pedestrian Crossing	

powered secondary navigation system to complement the primary navigation system. The constant training simulation parameters are 10 CNN layers, NAND logic repetitive cycles of RC = 5 and 10-different road sign categories, and RRAM devices with I_{comp} ranging from 5 - 50μA. Based on the given training parameters, the simulation was performed for 50,000 steps, and the obtained training accuracy is plotted in Fig. 11. We see the training accuracy percentage for $I_{comp} = 50\mu$A increase from 92% (Category 10) to 97% (Category 6) and further to 98% (Category 2). A similar trend is observed for 5μA and 10μA with a fractional increment in training accuracy from their base accuracy. Hence, the reduction in label category results in increased prediction accuracy for the given RRAM logic element-based computation. There is another trend that is to be noted in Fig. 11, where we see the slope to reach the peak of the accuracy curve saturation increasing as the category count decreases. The maximum prediction accuracy is attained faster in fewer thousand training steps for Category 2 when compared to the other two categories, meaning that the training for a smaller number of categories can be stopped much earlier while achieving the maximum training accuracy with reduced computation power. However, the accuracy can still be further improved with more training data set by continuing the learning process for further performance improvement.

Precision and *Recall* are useful metrics to understand and further tune the training accuracy trend with appropriate parameters and training data settings. The precision metric is the ratio of total true positive results to the sum of all true positive and false-positive results, while on the other hand the recall metric is computed using a true positive and false negative ratio. A confusion matrix is a summary of the prediction results for the given classification using the number of correct and incorrect predictions which are summarized with count values and broken down by each class. The confusion matrix plot is intended to display the Precision and Recall for the three-category groups of 10, 6, and 2. Fifty images are used to compute the prediction results for the 3 category groups; and together with the application of the trained weight obtained from 50,000 training steps and the HW pipeline configured with 50μA RRAM logical element with 5 repetitive cycles for every NAND logic, the results are obtained and plotted in Fig. 12. The X-axis of the confusion matrix represents the predicted labels results, the Y-axis shows the labels of the actual input images, while the diagonal represents the correct prediction, and other elements in the matrix represent incorrect prediction. To explain it further by looking into Fig. 12(a), the prediction for NO- Entry label was 92.46% accurate from the result of 50 test images and the breakdown of the 8% incorrect labeling for the given NO-Entry images classification includes NO-Left (2.47%), No-Parking (0.67%), Parking (3.17%) and Pedestrian-Crossing (1.23%). A similar interpretation can be used for the Category 6 and Category 2 groups from Figs. 12(b) and (c), respectively. The average prediction accuracy calculated with the diagonal elements (dark blue) from the confusion matrix in Fig. 12 shows the trend whereby the prediction accuracy using defective hardware (RRAM) logic improves for a reduction in labeled image category from 10 to 2. The average prediction accuracy for Category 10 is 91.6%, Category 6 is 94.7%, and Category 2 is 97.34%. While the prediction accuracy increases for the given RRAM variability by 5-6%, the overall classification to more delicate labels is significantly compromised from Category 10 down to Category 2 which might nullify the benefits of the end use application.

B. EMULATING NEUROMORPHIC CIRCUIT USING RRAM DIGITAL UNIVERSAL GATES AND POWER CONSUMPTION BENCHMARK

The truth table of the universal digital gates is presented using a 1T2R RRAM logic in sections II and Fig.5(d). However, the proposed 1T2R structure requires a few additional passive and active electronic components to make the circuit useful for any practical applications. As shown in Figs. 14(a) and 14(b), simple modifications are performed on the given 1T2R structure: an additional bias resistor (bias) added in series to the input restores A and B giving a 1T4R structure as in Fig. 13(a). The bias resistor acts as a voltage divider when a voltage / logic changes across input resistors A / B and maintains the required gating threshold voltage to the ON/OFF Transistor switch. The load resistor acts as a strong pull-up resistor for the output (OUT). The 2T3R configuration is a practical circuit as well; here, the load resistor is replaced with a second transistor to reduce the power dissipated on the load resistor for a higher load rating. The bias resistor draws significantly less current to gate the transistor. Hence, all the resistors in Figs. 13(a, b) are selected with mega-Ohm range of resistances to keep the power dissipation very minimal and ensure the practicability of the use of the circuit. The following sections show the various resistance values and their design considerations with SPICE simulations. Today's modern NAND CMOS circuit, shown in Fig. 13(c), is implemented using 4 transistors to perform the required switching operations to exhibit the NAND truth

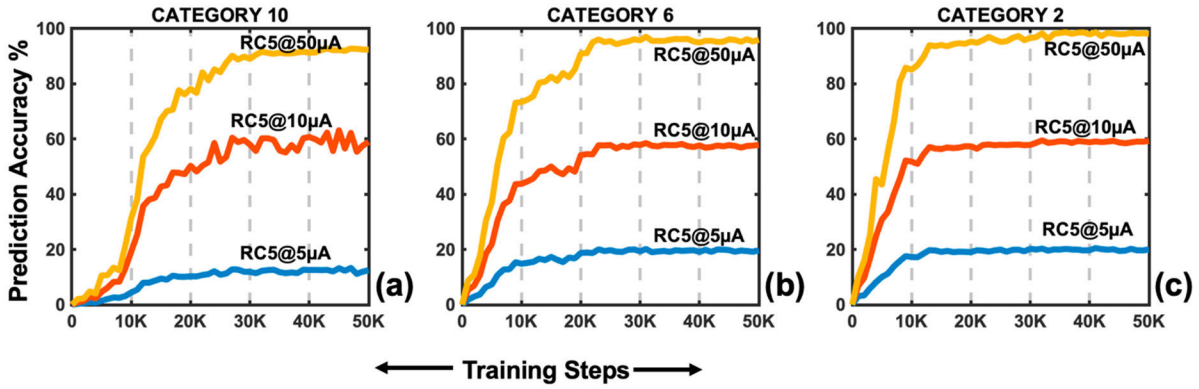


FIGURE 11. Prediction accuracy trend plotted by varying the labeled image category size from 10 to 2 for I_{comp} (5,10, and 50 μA) with the repetitive cycle of 5 and hidden layer count of 10, (a) 10 label group, (d) 6 label group, (c) 2 label group.

table for the give (1-bit) inputs. Between Fig.14 (b) – 2T3R and Fig.14(c) – 4T, there is a 2-transistor saving in each NAND gate implementation of 2T3R. The practical design for repetitive cycle 3 (RC3) abstraction circuit is proposed as shown in Fig. 14. The MCBC circuit function works as a First-in Max-Out (FIMO) memory to cache the input data and a combinational logic to output the maximum occurred input data at any given time. We propose using the RRAM elements with external program and bias voltages, and a transistor structure to demonstrate a cache memory that outputs the best of the occurrence from the input. The design of the bias, program voltage sources, and select 1 & 2 (SEL 1&2) logic switches are out of our current simulation scope. We assume these peripheral circuits to be designed as a global circuit by which few circuits can be used to program and reset the entire array of RRAM NAND gates.

The select-1 (SEL 1) switch is used to choose between the bias (operation voltage or input) and RESET voltage of the RRAM devices while the select-2 (SEL 2) switch functions as a toggle switch, programming one RRAM every clock cycle, and for RC=3, three clock cycles used to program the three RRAM arrays. The bias resistor connected in serial to the input resistors R_1, R_2 and R_3, supplies the required gating threshold voltage (V_{GS}) for the transistor (T1). The input data holding resistors R_1, R_2, and R_3 are connected in parallel, and the overall equivalent resistance (R_{eq}) varies for the different resistance values of R_1, R_2, and R_3. The three parallel resistors must replicate the cache memory by taking 3 input data and output the most frequently occurring bit. The 3 RRAMs programming sequences are shown in 4 states as in Fig. 14, which mimics the cache memory to output the most frequently received bit. With T1 being supplied with a voltage equal or greater than V_{GS}, it will be turned ON, by which the V_{CC} and the output are pulled low (logic-0) through T1. The T1 operates vice versa for V_{GS} less than the gating threshold by which the output is pulled high (logic-1). All the 4 combinations of R_1, R_2, and R_3, and their corresponding resistance values are shown in Fig. 14. The resistance values for logic-0 are considered as less than or equal to 1MΩ, while the logic-1 range is considered as greater than or equal to 100MΩ. During State-1 (S1), where all three resistances are of 1MΩ, the equivalent resistance is about 333 kΩ, and the bias resistor is fixed to 500 kΩ; hence the $V_{GS} \sim 0.72$V, which is greater than the gating voltage of 0.60V, turns on the T1. Now, for state S2, $R_1 = R_2 = 1$MΩ and $R_3 = 100$MΩ resulting in $R_{eq} \sim 497$ kΩ with a gating voltage of 0.602V that sees across the 500 kΩ bias resistor, keeping the T1 ON. Here for both S1 and S2, the most frequently occurring resistance value is 1 MΩ (defined by logic-0), and both also obtained state logic-0 at the outputs. A similar operation sequence is seen at states S3 and S4, where the most occurred resistance value is 100MΩ, corresponding to logic-1 at the outputs.

Hence, the given 1T4R logic mimics a cache memory to output the most recurring bit from the input. A similar sequence can be used for RC-5 and RC-7 by using 5 and 7 parallel input RRAMs. To keep the power dissipation negligible, the resistance range must be maintained in the range of MΩ and above. The LT-Spice tool is used for the simulation and analysis of the RRAM-NOR circuit. A 65nm MOS transistor model from the Arizona State University (ASU) repository was used to build a SPICE circuit of RRAM-NOR gate with two abstraction layers (MCBC RC=3) connected to the input port of the NOR gate, as shown in Fig. 15. Here the blue shaded area shows the NOR circuit, and orange/green shaded regions will be the two MCBC circuits. From Fig. 15, the variable resistors represent the RRAM device $\{R_5 \sim R_8, R_{10} \sim R_{13}\}$ and the fixed resistors are $\{R_2, R_1 = 120\text{M}\Omega\}$, $\{R_9, R_4, R_{14} = 1\text{M}\Omega\}$ and $R_3 = 150$MΩ. The resistance values of the MCBC circuits are programmed as per the input sequence using the select switch 1 and 2, as explained in the earlier section. The NOR operation is demonstrated for 10 clock cycles or steps for the given SPICE circuit shown in Fig. 16. Cycles 1∼4 program the resistance of the RRAMs in the given 1T-4R structure, following this, in cycle 5∼6 the 1T-4R structure switched back to the operational mode after being programmed with the required resistance values for operating as RRAM-NOR logic when reaching the 7^{th} clock cycle. Hence clock cycles 1∼6 are one-time configuration cycles

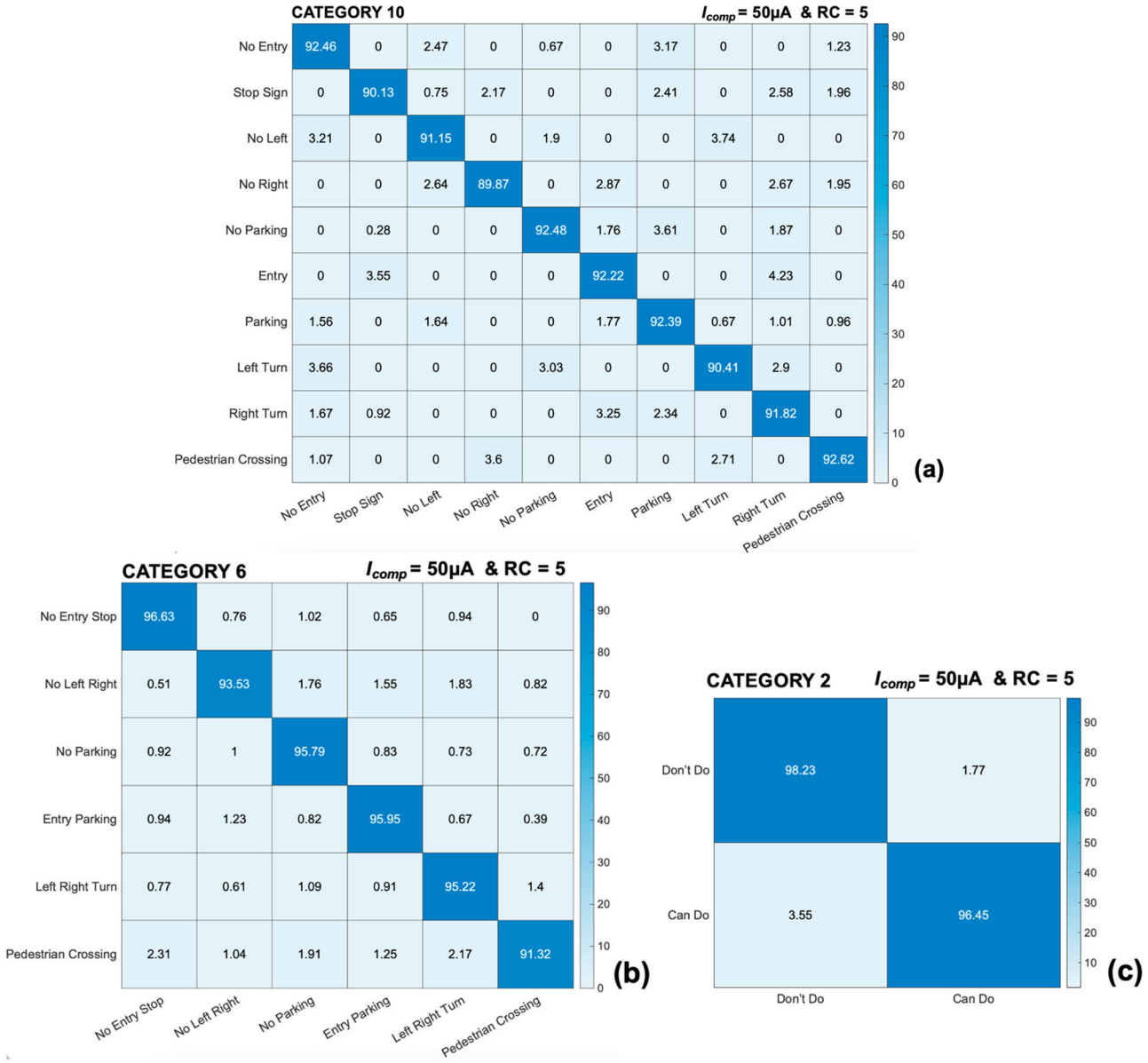


FIGURE 12. The confusion matrix presented for the true-positive, false-positive, and false-negative predictions (precision and recall). Note that the numbers in the matrix are shown in percentage and obtained from a test data set size of fifty random images. (a) 10 category data set, (b) 6 category data set, (c) 2 category data set. All the analyses presented here is for $I_{comp} = 50\mu$A and RC = 5.

during the chip power ON. Once the chip is configured with appropriate RRAM resistance values and functions as a computational logic to perform the convolution operation, the latency of the RRAM-NOR gate will be just 1 cycle, which is demonstrated from clock cycle 7∼8 (NOR truth table). A large sequence of RRAM-NOR gates is combined to operate as combinational and sequential circuits to achieve 10 layers of convolution operation during the forward pass. The clock cycles 7 to 10 demonstrate the NOR truth logic where for a logic-0 (0 Volts) at inputs A and B, the gating voltage will be 0V, keeping T1 OFF and thus a logic-1 set at the output. Similarly, for the other input sequences of A and B, the gating voltage varies from 0.66V to 0.85V which keeps the T1 in the ON state, and a logic-0 state is obtained at the output; thus, demonstrating the NOR operation.

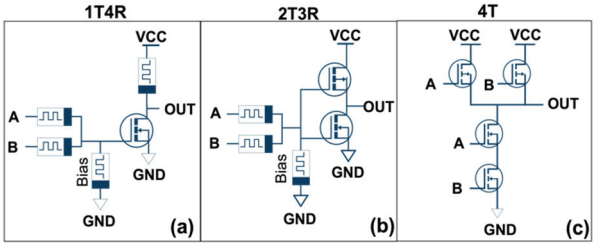
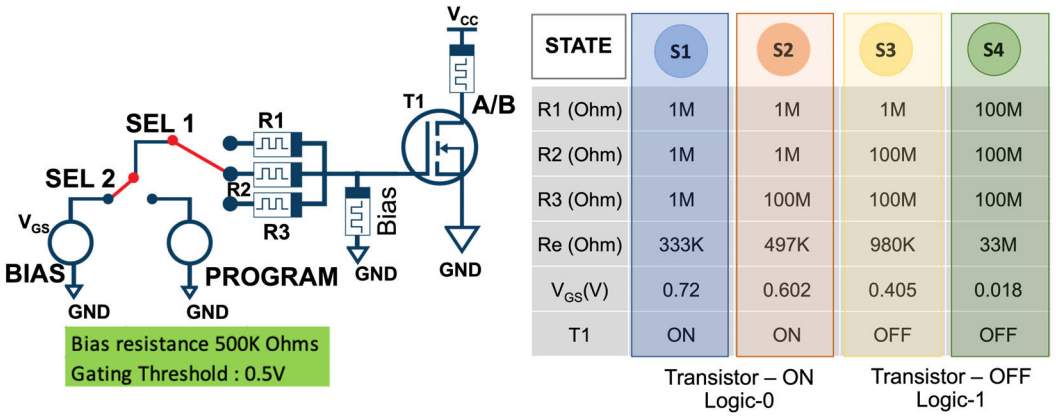


FIGURE 13. Practical CMOS transistor-based NAND/NOR (universal) gate circuit design shown with required bias resistor. (a) one transistor and four resistors (RRAM) circuit, (b) two transistor and three resistors (RRAM), here the weak pull up (VCC) resistor is replaced with a CMOS transistor, (c) typical conventional four transistor circuit.

The layout design for the simulated SPICE circuit is shown in Fig. 17, where the unit cell consists of a NOR (1T3R)



STATE	S1	S2	S3	S4
R1 (Ohm)	1M	1M	1M	100M
R2 (Ohm)	1M	1M	100M	100M
R3 (Ohm)	1M	100M	100M	100M
Re (Ohm)	333K	497K	980K	33M
V_{GS}(V)	0.72	0.602	0.405	0.018
T1	ON	ON	OFF	OFF

FIGURE 14. RRAM implemented (3-bits) as arbitrary logic and demonstrated using four states S1, S2, S3, and S4. 1M Ohm represents Logic-0 state and 100M Ohm as Logic-1. The select switches (SEL1 and SEL2) are programmed to choose between the bias or programming source for the RRAM (R1, R2, and R3).

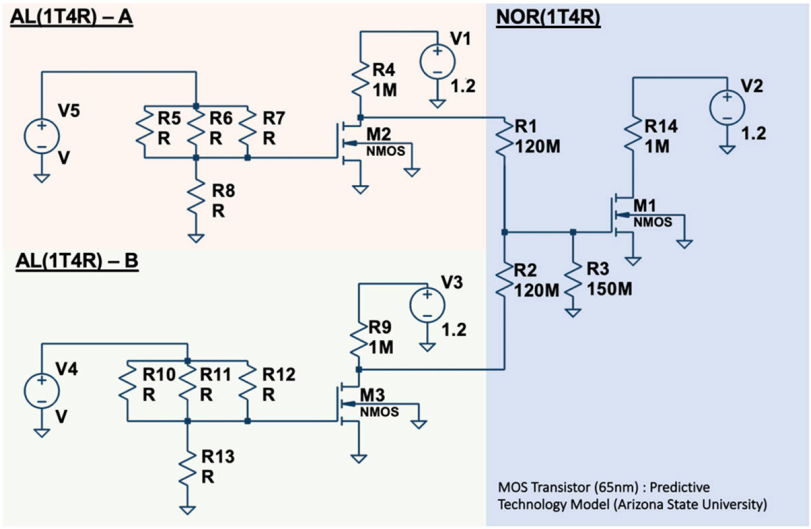


FIGURE 15. RRAM-NOR circuit with two MCBC RC3 logic shown. A 65nm MOS transistor SPICE model (Predictive Technology Model (from Arizona State University)) is used to design and simulate the in-memory unit cell.

and two MCBC circuits - MCBC-A (1T4R) on the left and MCBC-B (1T4R) on the right. The intention is to compute the unit cell dimension and estimate the overall chip area and heat dissipation for 5 different groups of the CNN (Group 1 of 10 layers to Group 5 of 2 layers). The layout design rules are as follows: (1) The MOS transistor dimension is about 140×140 nm^2 and was extracted from ST Microelectronics 65nm design toolkit [49], (2) The RRAMdimension extracted from Fantini et al. [15] of 18×18 nm^2, (3) The component to component spacing is about 190nm and is based on the ST Microelectronics 65nm design [49], (4) the other metal layers and interconnecting dimensions follow as shown in Fig. 17. From the layout design, the unit cell's overall dimension is 3190×1230 nm^2. The NOR and NAND RRAM gates have the same structure except for the variation in serial and parallel resistor values; therefore, this layout design is common for the universal gates. We omit the select switch circuit design in the layout drawing and reframe our analysis to the primary logic unit cell.

The proposed layout drawing here is an initial design to illustrate the methodology of an RRAM-based NOR unit cell structure. We certainly have room for further optimizing the unit cell layout, which we intend to explore in our future work. Furthermore, the layout drawing can be optimized by choosing the foundry's design recommendations, with fine-tuned standard cell design for the specific foundry fabrication process. The above simulation results and dimensions of the logical compute element (RRAM gate) were further used to estimate and project the overall chip area, power consumption, heat dissipation, and battery life (IoT application) for the given 5 hidden layer groups (Group 1 of 10 layers to Group 5 of 2 layers). The standard formula used to compute the size and performance of the different convolution groups is shown in Eqns. (1) $\sim$ (4) below. The

Operation Seq.	CLOCK	1	2	3	4	5	6	7	8	9	10
RRAM Reset	RESET RRAM	-Ve									
RRAM Program	PROG 1		+Ve (R1)			OFF					
	PROG 2			+Ve (R2)		OFF					
	PROG 3				+Ve (R3)	OFF					
Set Operation Mode	OPR SEL					ENABLE	+1.2V	+1.2V	+1.2V	+1.2V	+1.2V
Input A	A (V)							0V	0V	+1.2V	+1.2V
	LOGIC-A							LOGIC-0	LOGIC-0	LOGIC -1	LOGIC -1
Input B	B (V)							0V	+1.2V	0V	+1.2V
	LOGIC-B							LOGIC-0	LOGIC-1	LOGIC-0	LOGIC-1
Gate Threshold	V_{GS} (V)							0V	+0.66V	+0.66V	+0.85V
Output	V_{DS} (V)							0V	+1.0V	+1.0V	+1.2V
	LOGIC-0							LOGIC -1	LOGIC-0	LOGIC-0	LOGIC-0
		AL– RRAM Programing Cycle				AL–Set Operation Mode					

FIGURE 16. Ten (clock) operation sequences for MCBC RC3 RRAM reset/program cycle and RRAM-NOR truth table verification are shown using the SPICE transistor model. Cycle 1∼4 shows the RRAM programming setup, 5∼6 will be MCBC preparation mode, and 7∼8 demonstrates the NOR logic.

hidden layers use 3 × 3 and 1 × 1 generic convolution block with different filter matrices sized from 32 to 512; the estimations of the total NAND logic count to construct the 3×3, and 1×1 is 5.3M and 1.7M, respectively. The gate count for the convolution modules was extracted from our previous work [46]. The total RRAM NAND gates are given by the sum of the product of all the convolution modules gate count and the given filter size as shown in Eqn. (1).

$$T_{lgc} = \sum_{Layers=1}^{n} (Conv_{gc} \times Q_{fms}) \tag{1}$$

T_{lgc} = Total Logic gates in all convolution layers
$Conv_{gc}$ = Convolution operator gate count per layer
Q_{fms} = Convolution filter size of a specific layer
n = Total number of hidden convolution layers

The dimension for a unit cell of one NOR/NAND gate with two Maximum Count Binary Comparator Layers of size RC=3 is estimated as (3190 × 1290 nm^2) from the layout design. Hence, the total RRAM NAND gate count and the unit cell area give the overall chip area of the respective hidden layer count. Other peripheral components such as interconnecting bus, clock bus, input/output buffers, pads etc. are not being considered during the calculation of the overall size since these peripheral components are foundry specific. Hence, we omit it to keep our estimation only for the required RRAM NAND logic. Based on the above assumptions, the overall chip size was calculated using the Eqn. (2).

$$T_{ucs} = T_{SC} \times T_{lgc} \tag{2}$$

T_{ucs} = Total area of all the unit cells (RRAM NAND)
T_{SC} = Size of a single standard RRAM NAND gate with RC3 (3190 × 1290 nm^2)

Eqn. (3) provides an estimate for the power consumption for the given hidden layer count at a simulation clock frequency of 100MHz.

$$E_{iter} = V_{oper} \times I_{mem} \times \frac{1}{F_{clk}} \times T_{ucs} \tag{3}$$

E_{iter} = Estimated power to process one image (Pixel size:224 × 224) at 100MHz operation
V_{oper} = Digital logic 1 threshold voltage
I_{mem} = RRAM operating current compliance
F_{clk} = Simulation digital clock speed

Today, the IoT applications are often supported by an external battery source. Hence, we have attempted to calculate the achievable battery life during the execution of the given RRAM NAND-based convolution operations on the edge using a 48V and 14Ah battery source. The size of the battery was chosen from a standard and typical available portable battery pack used to power up the e-scooters. Eqn. (4) estimates the battery pack lifetime for different convolution operations. Last but not the least, heat dissipation is one of the primary parameters which defines the reliability and lifespan of an IoT application. The power estimation in relation to the battery life is shown for the RRAM-NAND logic alone, whereas the additional power used for the synaptic weights read /write is not considered (excluded) here as our framework focuses only on the simulation of the in-memory mathematical logic/operator for performing the convolution operation.

$$B_{lpc} = \frac{E_{src}}{E_{iter}} \tag{4}$$

B_{lpc} = Estimated battery life to perform convolution in all hidden layer with source (48V14Ah)
E_{src} = Battery source rating (48V14Ah)

A consolidation of the energy consumption and estimated battery lifetime using Eqns. (1) ∼ (4) for our simulated in-memory circuit for the different counts of the hidden layer

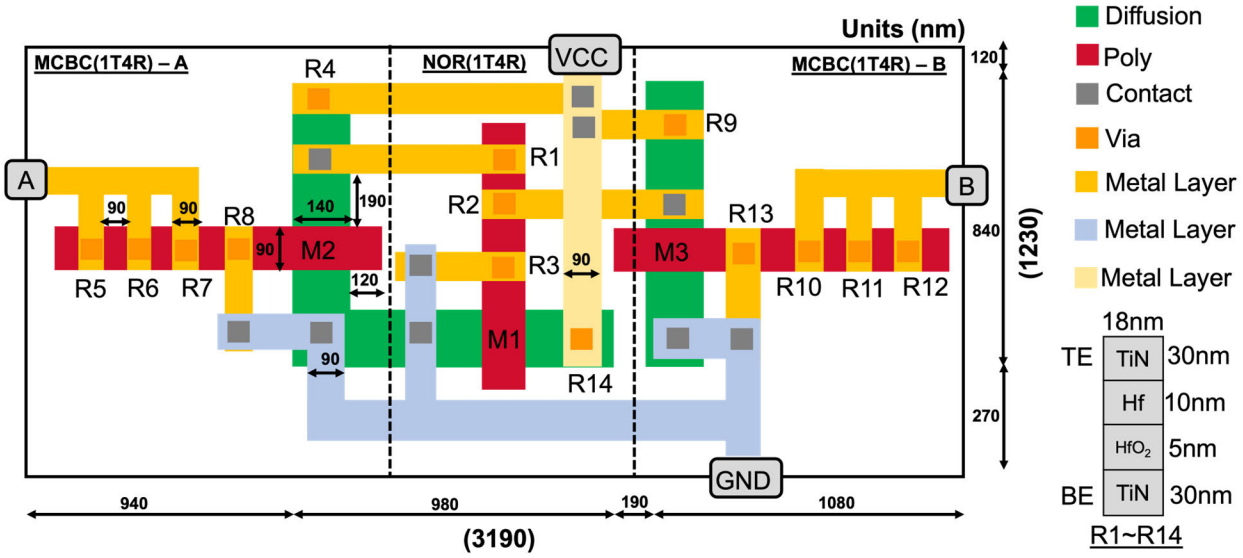


FIGURE 17. Unit (standard) cell design shown for two MCBC layers and one NOR gate structure using RRAM as the main design block.

TABLE 5. (Emulated) RRAM-based neuromorphic chip energy budget and battery life specification.

Group (Layers)	(T_{lgc}) Total RRAM ($\times 10^{12}$)	(T_{ucs}) Chip Area ~(mm^2)	(E_{iter})Energy Consumption (J/Image@100MHz)			(B_{lpc})Battery Lifetime (Hours) Source 48V14Ah		
			5µA	10µA	50µA	5μA	10μA	50µA
1(10)	7.2	28.21	3.46	6.91	34.56	194.44	97.22	19.44
2(8)	3.5	13.88	1.68	3.36	16.80	400.20	200.15	40.00
3(6)	1.7	6.71	0.82	1.63	8.16	823.53	411.67	82.35
4(4)	0.8	3.13	0.38	0.77	3.84	1750	875.12	175
5(2)	0.3	1.34	0.14	0.29	1.44	4666.67	2333.33	466.67

groups and filter sizes is listed in Table 5. The table shows that the total computational energy and the chip estate area consumed reduce by as much as 10-20X as the hidden layer count and the current compliance of the RRAM decrease. However, the design challenge here is to strike a balance whereby the required battery lifetime is met for the right combination of the hidden layers and the RRAM switching current. There are several IoT applications that need to work with low maintenance frequency due to their use in remote deployment sites and such applications should consider using 10μA RRAM with a 6-layer CNN or a 50μA RRAM with 4-layer CNN for a 6 month to 1 year frequency of maintenance schedule provided the prediction accuracy drop brought over by RRAM device variability is within the acceptable range as defined by the end application.

IV. CONCLUSION OF THE STUDY

We have simulated and quantified the impact of RRAM variability on the training accuracy of a deep CNN, for which three different RRAM current compliances of 5, 10, and 50μA were used corresponding to soft to hard filamentation regimes. A suitable 65nm (TiN/HfO_2/Hf/TiN) OxRAM NAND/NOR logic was constructed and simulated for five different groups of hidden layers (convolution operation) to show the training accuracy trend by implementing a purely digital RRAM logic using the look-up table (LUT) approach. Our methodology of using LUT-based RRAM resistive encoding scheme was well demonstrated with a suitable neuromorphic simulation framework using Python and TensorFlow in the upper layer and Verilog HDL in the lower layer by construction of FP32MUL and FP32ADD using RRAM (1T4R / 2T3R) based NAND logics. We have demonstrated that adding MCBC logic with a standard RRAM NAND logic improves the overall device variability. The MCBC layer is estimated to consume an extra space of 1150 $\times$ 1230 nm^2 per logic gate per input, which results in an overall prediction accuracy improvement from 10% to 60% (RC1=10%, RC3=30%, RC5=45%, and RC7=60%). The estimated battery life can range anywhere between 19 to 466 days for CNN layers of 10 to 2 when configured with a 50μA RRAM switching current, considering the maximum prediction accuracy achieved

compared to the software training pipeline. Finally, the in-memory circuit design with overall chip area, power consumption, estimated battery lifespan, and heat dissipation are derived for IoT application deployment for a truly edge implementation.

As a continuation of this work, we intend to build and simulate the backward pass computation system using the same RRAM NAND/NOR logic in our future work and quantify the impact of memristive performance variability seen in the final learning accuracy trend for a completely edge based forward and backward flow computation schema. This study provides a practical multi-faceted design tool for RRAM-based edge computing that enables the quantification of the impact of CNN architecture and RRAM operating current levels on the prediction accuracy, chip estate area, energy consumption and battery replacement frequency. The design tool can potentially be harnessed as a multi-objective design optimization decision making framework for RRAM edge compute applications depending on the end user-defined requirements for the application in context.

REFERENCES

[1] S. Zhu, K. Ota, and M. Dong, "Green AI for IIoT: Energy efficient intelligent edge computing for industrial Internet of Things," *IEEE Trans. Green Commun. Netw.*, vol. 6, no. 1, pp. 79–88, Mar. 2022.

[2] H. Lavi. *Measuring Greenhouse Gas Emissions in Data Centres: The Environmental Impact of Cloud Computing*. Accessed: Sep. 25, 2022. [Online]. Available: https://www.climatiq.io/blog/measure-greenhouse-gas-emissions-carbon-data-centres-cloud-computing

[3] Y. Wang, Q. Wang, S. Shi, X. He, Z. Tang, K. Zhao, and X. Chu, "Benchmarking the performance and energy efficiency of AI accelerators for AI training," in *Proc. 20th IEEE/ACM Int. Symp. Cluster, Cloud Internet Comput. (CCGRID)*, May 2020, pp. 744–751.

[4] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of GPU kernels using performance counters," in *Proc. Int. Conf. Green Comput.*, Aug. 2010, pp. 115–122.

[5] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and benchmarking of machine learning accelerators," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2019, pp. 1–9.

[6] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *J. Field Robot.*, vol. 37, no. 3, pp. 362–386, Apr. 2020.

[7] É. Zablocki, H. Ben-Younes, P. Pérez, and M. Cord, "Explainability of deep vision-based autonomous driving systems: Review and challenges," 2021, *arXiv:2101.05307*.

[8] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, and W. Shi, "Computing systems for autonomous driving: State of the art and challenges," *IEEE Internet Things J.*, vol. 8, no. 8, pp. 6469–6486, Apr. 2021.

[9] K. Roy, I. Chakraborty, M. Ali, A. Ankit, and A. Agrawal, "In-memory computing in emerging memory technologies for machine learning: An overview," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.

[10] S. Bavikadi, P. R. Sutradhar, K. N. Khasawneh, A. Ganguly, and S. M. P. Dinakarrao, "A review of in-memory computing architectures for machine learning applications," in *Proc. Great Lakes Symp. VLSI*, Sep. 2020, pp. 89–94.

[11] D. Ielmini and G. Pedretti, "Device and circuit architectures for in-memory computing," *Adv. Intell. Syst.*, vol. 2, no. 7, 2020, Art. no. 2000040.

[12] A. Sebastian, M. L. Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnol.*, vol. 15, pp. 529–544, Jul. 2020, doi: 10.1038/s41565-020-0655-z.

[13] R. Degraeve, A. Fantini, N. Raghavan, L. Goux, S. Clima, B. Govoreanu, A. Belmonte, D. Linten, and M. Jurczak, "Causes and consequences of the stochastic aspect of filamentary RRAM," *Microelectron. Eng.*, vol. 147, pp. 171–175, Nov. 2015.

[14] S. Clima, Y. Y. Chen, A. Fantini, L. Goux, R. Degraeve, B. Govoreanu, G. Pourtois, and M. Jurczak, "Intrinsic tailing of resistive states distributions in amorphous HfO_x and TaO_x based resistive random access memories," *IEEE Electron Device Lett.*, vol. 36, no. 8, pp. 769–771, Aug. 2015.

[15] A. Fantini, L. Goux, R. Degraeve, D. J. Wouters, N. Raghavan, G. Kar, A. Belmonte, Y. Y. Chen, B. Govoreanu, and M. Jurczak, "Intrinsic switching variability in HfO_2 RRAM," in *Proc. 5th IEEE Int. Memory Workshop*, May 2013, pp. 30–33.

[16] S. Song, T. Titirsha, and A. Das, "Improving inference lifetime of neuromorphic systems via intelligent synapse mapping," in *Proc. IEEE 32nd Int. Conf. Appl.-Specific Syst., Architectures Processors (ASAP)*, Jul. 2021, pp. 17–24.

[17] Y. Zhang, G. He, G. Wang, and Y. Li, "Efficient and robust RRAM-based convolutional weight mapping with shifted and duplicated kernel," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 2, pp. 287–300, Feb. 2021.

[18] S. Yin, X. Sun, S. Yu, and J.-S. Seo, "High-throughput in-memory computing for binary deep neural networks with monolithically integrated RRAM and 90-nm CMOS," *IEEE Trans. Electron Devices*, vol. 67, no. 10, pp. 4185–4192, Oct. 2020.

[19] G. Boquet, E. Macias, A. Morell, J. Serrano, E. Miranda, and J. L. Vicario, "Offline training for memristor-based neural networks," in *Proc. 28th Eur. Signal Process. Conf. (EUSIPCO)*, Jan. 2021, pp. 1547–1551.

[20] Z. Sun, E. Ambrosi, G. Pedretti, A. Bricalli, and D. Ielmini, "In-memory PageRank accelerator with a cross-point array of resistive memories," *IEEE Trans. Electron Devices*, vol. 67, no. 4, pp. 1466–1470, Apr. 2020.

[21] V. Milo, A. Glukhov, E. Perez, C. Zambelli, N. Lepri, M. K. Mahadevaiah, E. P.-B. Quesada, P. Olivo, C. Wenger, and D. Ielmini, "Accurate program/verify schemes of resistive switching memory (RRAM) for in-memory neural network circuits," *IEEE Trans. Electron Devices*, vol. 68, no. 8, pp. 3832–3837, Aug. 2021.

[22] E. Pérez, A. J. Pérez-Ávila, R. Romero-Zaliz, M. K. Mahadevaiah, E. P.-B. Quesada, J. B. Roldán, F. Jiménez-Molinos, and C. Wenger, "Optimization of multi-level operation in RRAM arrays for in-memory computing," *Electronics*, vol. 10, no. 9, p. 1084, May 2021.

[23] Y. Feng, P. Huang, Y. Zhao, Y. Shan, Y. Zhang, Z. Zhou, L. Liu, X. Liu, and J. Kang, "Improvement of state stability in multi-level resistive random-access memory (RRAM) array for neuromorphic computing," *IEEE Electron Device Lett.*, vol. 42, no. 8, pp. 1168–1171, Aug. 2021.

[24] J. An, S. Oh, T. V. Nguyen, and K.-S. Min, "Synapse-neuron-aware training scheme of defect-tolerant neural networks with defective memristor crossbars," *Micromachines*, vol. 13, no. 2, p. 273, Feb. 2022.

[25] W.-Q. Pan, J. Chen, R. Kuang, Y. Li, Y. H. He, G.-R. Feng, N. Duan, T.-C. Chang, and X.-S. Miao, "Strategies to improve the accuracy of memristor-based convolutional neural networks," *IEEE Trans. Electron Devices*, vol. 67, no. 3, pp. 895–901, Mar. 2020.

[26] J. Feng, Y. Wang, X. Hu, G. Wen, Z. Wang, Y. Lin, D. Wu, Z. Ma, L. Zhao, Z. Lu, and Y. Xie, "A hybrid RRAM-SRAM computing-in-memory architecture for deep neural network inference-training edge acceleration," in *Proc. Silicon Nanoelectron. Workshop (SNW)*, Jun. 2021, pp. 1–2.

[27] J. Li, G. Zhou, Y. Li, J. Chen, Y. Ge, Y. Mo, Y. Yang, X. Qian, W. Jiang, H. Liu, M. Guo, L. Wang, and S. Duan, "Reduction of 93.7% time and power consumption using a memristor-based imprecise gradient update algorithm," *Artif. Intell. Rev.*, vol. 55, pp. 657–677, Aug. 2021.

[28] A. Majumdar, M. Bocquet, T. Hirtzlin, A. Laborieux, J.-O. Klein, E. Nowak, E. Vianello, J.-M. Portal, and D. Querlioz, "Model of the weak reset process in HfO_x resistive memory for deep learning frameworks," *IEEE Trans. Electron Devices*, vol. 68, no. 10, pp. 4925–4932, Oct. 2021.

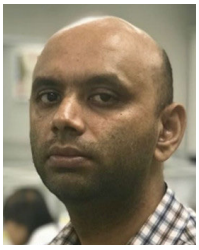
[29] Q. Wang, Y. Park, and W. D. Lu, "Device non-ideality effects and architecture-aware training in RRAM in-memory computing modules," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[30] K. Qiu, Z. Zhu, Y. Cai, H. Sun, Y. Wang, and H. Yang, "MNSIM-TIME: Performance modeling framework for training-in-memory architectures," in *Proc. IEEE 3rd Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Jun. 2021, pp. 1–4.

[31] S. Yu, W. Shim, X. Peng, and Y. Luo, "RRAM for compute-in-memory: From inference to training," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 7, pp. 2753–2765, Jul. 2021.

[32] H. Jiang, S. Huang, X. Peng, and S. Yu, "MINT: Mixed-precision RRAM-based IN-memory training architecture," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5.

[33] Y. Liu, B. Gao, F. Xu, W. Zhang, Y. Xi, J. Tang, and H. Qian, "A compact model for relaxation effect in analog RRAM for Computation-in-Memory system design and benchmark," in *Proc. 5th IEEE Electron Devices Technol. Manuf. Conf. (EDTM)*, Apr. 2021, pp. 1–3.
[34] M. Giordano, K. Prabhu, K. Koul, R. M. Radway, A. Gural, R. Doshi, Z. F. Khan, J. W. Kustin, T. Liu, G. B. Lopes, V. Turbiner, W.-S. Khwa, Y.-D. Chih, M. -F. Chang, G. Lallement, B. Murmann, S. Mitra, and P. Raina, "Chimera: A 0.92 TOPS, 2.2 TOPS/W edge AI accelerator with 2 MByte on-chip foundry resistive ram for efficient training and inference," in *Proc. Symp. VLSI Circuits*, Jun. 2021, pp. 1–2.
[35] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks," *Synth. Lect. Comput. Archit.*, vol. 15, no. 2, pp. 1–341, 2020.
[36] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," 2016, *arXiv:1603.07285*.
[37] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," 2018, *arXiv:1811.03378*.
[38] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, *arXiv:1312.4400*.
[39] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," 2015, *arXiv:1511.08458*.
[40] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: MIT Press, 2012.
[41] I. Goodfellow, Y. Bengio, and A. Courville, "Back-propagation and other differentiation algorithms," in *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, ch. 6.5, pp. 200–220.
[42] C. P. Robert, *The Bayesian Choice* (Springer Texts in Statistics), 2nd ed. New York, NY, USA: Springer, 2007, doi: 10.1007/0-387-71599-1.
[43] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[44] N. Raghavan, K. L. Pey, W. Liu, X. Wu, X. Li, and M. Bosman, "Evidence for compliance controlled oxygen vacancy and metal filament based resistive switching mechanisms in RRAM," *Microelectron. Eng.*, vol. 88, no. 7, pp. 1124–1128, 2011.
[45] X. Wu, D. Cha, M. Bosman, N. Raghavan, D. B. Migas, V. E. Borisenko, X. X. Zhang, K. Li, and K.-L. Pey, "Intrinsic nanofilamentation in resistive switching," *J. Appl. Phys.*, vol. 113, no. 11, 2013, Art. no. 114503.
[46] N. L. Prabhu and N. Raghavan, "Generalized convolution simulation stack for RRAM device based deep learning neural network," in *Proc. IEEE Int. Symp. Phys. Failure Anal. Integr. Circuits (IPFA)*, Jul. 2020, pp. 1–6.
[47] Y. Wei, X. Pan, H. Qin, W. Ouyang, and J. Yan, "Quantization mimic: Towards very tiny CNN for object detection," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 267–283.
[48] A. Jahanshahi, "TinyCNN: A tiny modular CNN accelerator for embedded FPGA," 2019, *arXiv:1911.06777*.
[49] F. Arnaud, F. Boeuf, F. Salvetti, D. Lenoble, F. Wacquant, C. Regnier, P. Morin, N. Emonet, E. Denis, J. C. Oberlin, and D. Ceccarelli, "A functional 0.69 μm2 embedded 6T-SRAM bit cell for 65 nm CMOS platform," in *Symp. VLSI Technol. Dig. Tech. Papers*, Jun. 2003, pp. 65–66.



NAGARAJ LAKSHMANA PRABHU is currently pursuing the Ph.D. degree with the Singapore University of Technology and Design (SUTD), examining the impact of RRAM device level variability as an in-memory computational element for deep learning neural network (DNN) applications. He has over 18 years of experience in industrial product design and development, specializing in machine vision and cloud computation. He has six scientific publications to his credit relating to construction methodology for look-up table (LUT)-based RRAM neuromorphic implementation for deep neural networks accounting for RRAM device level and cycle-to-cycle variability. He also serves as one of the Director of the ALAI Laboratories, focusing on design and development of vision-based IoT products and solutions (on the cloud and on the edge) for general use daily applications.

NAGARAJAN RAGHAVAN (Member, IEEE) received the Ph.D. degree in microelectronics from the Division of Microelectronics, Nanyang Technological University (NTU), Singapore, in 2012. He is currently a Tenure-Track Assistant Professor at the Engineering Product Development (EPD) Pillar, Singapore University of Technology and Design (SUTD). Prior to this, he was a Postdoctoral Fellow at the Massachusetts Institute of Technology (MIT), Cambridge, and at IMEC, Belgium, in joint association with Katholieke Universiteit Leuven (KUL). His work focuses on prognostics and health management for electromechanical failures, design for reliability, lifecycle management of nanoelectronic devices, physics of failure, optimization of polymer nanocomposites, and uncertainty quantification for additive manufacturing. He was a recipient of the IEEE EDS Early Career Award for 2016, Asia-Pacific recipient for the IEEE EDS Ph.D. Student Fellowship, in 2011, and the IEEE Reliability Society Graduate Scholarship Award, in 2008. To date, he has authored/coauthored more than 250 international peer-reviewed publications and five invited book chapters as well. He served as the General Chair for IEEE IPFA 2021 at Singapore and has consistently served on the Review Committee for various IEEE journals and conferences, including IRPS, IIRW, IPFA, and ESREF. He is an Associate Editor of the IEEE Access and *Microelectronic Engineering* journals as well. He was an Invited Member of the IEEE GOLD Committee (2012–2014).