RxNN: A Framework for Evaluating Deep Neural Networks on Resistive Crossbars

Shubham Jain, Abhronil Sengupta, Member, IEEE, Kaushik Roy, Fellow, IEEE, and Anand Raghunathan, Fellow, IEEE

Abstract—Resistive crossbars designed with nonvolatile memory devices have emerged as promising building blocks for deep neural network (DNN) hardware, due to their ability to compactly and efficiently realize vector–matrix multiplication (VMM), the dominant computational kernel in DNNs. However, a key challenge with resistive crossbars is that they suffer from a range of device and circuit level nonidealities, such as driver resistance, sensing resistance, sneak paths, interconnect parasitics, nonlinearities in the peripheral circuits, stochastic write operations, and process variations. These nonidealities can lead to errors in VMMs, eventually degrading the DNN’s accuracy. It is therefore critical to study the impact of crossbar nonidealities on the accuracy of large-scale DNNs (with millions of neurons and billions of synaptic connections). However, this is challenging because the existing device and circuit models are too slow to use in application-level evaluations. We present RxNN, a fast and accurate simulation framework to evaluate large-scale DNNs on resistive crossbar systems. RxNN splits and maps the computations involved in each DNN layer into crossbar operations, and evaluates them using a fast crossbar model (FCM) that accurately captures the errors arising due to crossbar nonidealities while being four-to-five orders of magnitude faster than circuit simulation. FCM models a crossbar-based VMM operation using three stages—nonlinear models for the input and output peripheral circuits (digital-to-analog and analog-to-digital converters), and an equivalent nonideal conductance matrix for the core crossbar array. We implement RxNN by extending the Caffe machine learning framework and use it to evaluate a suite of six large-scale DNNs developed for the ImageNet Challenge (ILSVRC). Our experiments reveal that resistive crossbar nonidealities can lead to significant accuracy degradations (9.6%–32%) for these large-scale DNNs. To the best of our knowledge, this article is the first quantitative evaluation of the accuracy of large-scale DNNs on resistive crossbar-based hardware. We also demonstrate that RxNN enables fast model-in-the-loop retraining of DNNs to partially mitigate the accuracy degradation.

Index Terms—Analog computing, artificial intelligence, crossbar modeling, crossbar nonidealities, deep neural networks (DNNs), in-memory computing, machine learning, nonvolatile memory, resistive crossbar, vector–matrix multiplication (VMM).

I. INTRODUCTION

DEEP neural networks (DNNs) have transformed the field of artificial intelligence in the past decade, and are currently used in several real-world products and services for speech recognition, image analysis, natural language processing, search engines, recommendation systems, and more [1], [2]. However, the large and rapidly growing computation and storage requirements of DNNs pose severe challenges to the systems on which they are deployed.

Resistive crossbars have garnered significant interest in realizing DNNs due to their ability to perform the underlying computational kernel, viz., vector–matrix multiplication (VMM), efficiently. They may be designed using a range of emerging devices, including resistive RAM (ReRAM), phase change memory (PCM), and Spintronics [3]–[6]. These devices have several desirable characteristics, such as high density, nonvolatility, and low voltage operation, enabling highly compact and energy-efficient DNN implementations. Consequently, several research efforts have explored resistive crossbar-based hardware at various levels of design abstraction [4], [7]–[37].

A key challenge with resistive crossbars is that the performed operation is only an approximation of the desired VMM. For example, consider the process where digital inputs are first converted to voltages and applied to the rows of the crossbar (which is programmed with the weights as conductances), and the resulting column currents are digitized to generate the digital outputs. Various device and circuit level nonidealities, viz., driver resistance, sensing resistance, sneak paths, interconnect parasitics, analog-to-digital converters (ADCs) and digital-to-analog converter (DAC) non-linearity, stochastic write operations, and process variations may lead to errors in the computed VMMs. These errors can degrade the overall accuracy of a DNN realized on a resistive crossbar system. Although DNNs are resilient to some imprecision in their computations [38]–[40], this resilience is not unlimited. Therefore, it is necessary to evaluate the impact of resistive crossbars nonidealities at the application level in order to establish their feasibility as the building blocks of DNN hardware.
Most previous efforts on resistive crossbar-based DNN implementations either do not consider nonidealities or model nonidealities in a very limited manner (e.g., as limited precision). Moreover, they focus their analysis on very simple networks and datasets (e.g., MNIST). Thus, they leave open the question of how nonidealities impact the accuracy of large-scale neural networks and more complex tasks. State-of-the-art DNNs often contain tens to hundreds of layers, millions of neurons, and billions of synaptic connections. Evaluating such networks requires a fast and scalable, yet accurate, model for resistive crossbars that can be integrated into state-of-the-art DNN software frameworks. Unfortunately, such a framework is currently unavailable. Device and circuit simulation (SPICE) models of resistive crossbars are accurate but extremely slow and infeasible for the large-scale network evaluation. On the other hand, architectural models of resistive crossbars [27], [29] target design space exploration and use highly simplified error models that are reasonable for their context, but inadequate for evaluating application-level accuracy of DNNs. For example, these models do not consider error dependence on the crossbar inputs, programmed conductances, and the crossbar column performing the computation.

We propose RxNN, a fast and accurate simulation framework to enable the functional evaluation of large-scale DNNs on resistive crossbars. RxNN splits and maps the DNN’s computations into crossbar operations, and evaluates them using a fast crossbar model (FCM) that accurately captures the errors arising due to crossbar nonidealities while being four-to-five orders of magnitude faster than circuit simulation. FCM models a crossbar-based VMM operation using three stages—nonlinear models for the input and output peripheral circuits (digital-to-analog and analog-to-digital converters), and an equivalent nonideal conductance matrix for the core crossbar array. The nonideal conductance matrix for each crossbar array is derived by presolving the voltage-current equations (Kirchoff’s loop law and Ohm’s law) for the array, requiring only a VMM to subsequently transform the input voltages to output currents.

We realize FCM using the well-known basic linear algebra subprograms (BLAS) interface and develop RxNN based on the Caffe [41] deep learning framework. We use RxNN to evaluate six large-scale DNNs for classifying the ImageNet [42] dataset and three simpler networks for classifying the CIFAR-10 and MNIST datasets. Our evaluation suggests that crossbar nonidealities impact accuracy much more significantly as we move from the smaller networks (LeNet and ConvNet) and simpler datasets to the larger networks (e.g., ResNet) and ImageNet, motivating the need for further research in cross-layer mitigation and compensation techniques.

In summary, the key contributions of this article are as follows.

1) We study the cumulative effect of resistive crossbar nonidealities by characterizing the resulting errors in the realized VMM operations. We find that the errors show significant data and hardware-instance dependence that should be considered for accurately modeling resistive crossbars.

2) We propose FCM, a fast and accurate functional crossbar model to capture the effects of crossbar nonidealities.

3) We develop RxNN, a software framework that can evaluate large-scale DNNs on resistive crossbar systems and help retrain to compensate for the effects of nonidealities.

4) We evaluate the application-level accuracy of six state-of-the-art DNNs, viz., ResNet-50, VGG-16, GoogleNet, AlexNet, OverFeat, and NiN on a resistive crossbar-based system. Our evaluation reveals that the degradation in accuracy due to nonidealities can be significant (9.6%–32%) for large-scale DNNs. This degradation can be partially alleviated by retraining, but calls for further research in compensation techniques.

The remainder of this article is organized as follows. Section II overviews the prior efforts related to this article. Section III provides the necessary background on resistive crossbars. Section IV discusses crossbar nonidealities and demonstrates their impact on VMMs realized using crossbars. Section V describes the proposed FCM models. Section VI presents the RxNN software framework. Section VII details the experimental methodology. We present experimental results in Section VIII and conclude this article in Section IX.

II. RELATED WORK

Resistive crossbars have attracted significant interest in recent years due to their ability to efficiently realize VMMs, the key computational kernel in DNNs [3], [43]–[45]. Prior efforts toward realizing DNNs on resistive crossbar systems can be broadly classified into specialized hardware accelerator designs [7]–[9], [11]–[13], nonideality mitigation schemes [14]–[18], [22]–[26], and design tools for resistive crossbar systems [27]–[30].

Specialized Hardware Accelerators: Resistive crossbar-based specialized hardware systems have been proposed for accelerating DNN inference [7]–[11] and training [12], [13]. These efforts focus on the evaluation of the proposed architecture using performance, energy, and area as their metrics, and either do not consider nonidealities or model only the quantized nature of crossbar-based VMM operations.

Nonideality Mitigation Schemes: Prior efforts have also proposed methods to mitigate the impact of crossbar nonidealities. These include: 1) (re-)training [16]–[21]; 2) optimized weight to conductance conversion [14]; 3) rank clipping to reduce the effects of nonidealities by lowering crossbar dimensions [25]; 4) schemes to alleviate the effect of hard failures [26]; and 5) hardware solutions to address low-voltage induced drift [15], programming errors [23], and IR drop [24]. The focus of all these efforts is to evaluate and mitigate errors due to crossbar nonidealities. However, they are restricted to simple networks and small datasets. This limitation is in large part due to the lack of a scalable simulation framework. Furthermore, many of these efforts also lack a detailed crossbar model and consider only a subset of crossbar nonidealities.
RxNN complements the above efforts on hardware accelerators and nonideality mitigation schemes. Moreover, it overcomes their limitations (such as considering only a limited set of nonidealities), and can accurately model crossbar-based systems while maintaining very high simulation speed (several orders-of-magnitude faster than SPICE).

**Design Tools:** To aid design space exploration, prior efforts [27]–[30] have proposed circuit-level macro models to evaluate crossbar systems. These efforts include: 1) MNSIM [27], a simulation platform to evaluate inference accelerators designed using resistive crossbars; 2) NeuroSim [28], a framework to evaluate crossbars systems designed for on-chip training; 3) a technology exploration tool for resistive crossbars [29]; and 4) AutoNCS [30], a tool to optimize the utilization and efficiency of a resistive crossbar system. While the primary focus of these tools has been performance, energy, and area evaluation of resistive crossbar systems, they also include simplistic accuracy/error models that are reasonable for design space exploration, but inadequate for evaluating application-level accuracy of large-scale DNNs. RxNN complements these efforts by focusing on accurately and efficiently modeling crossbar nonidealities in the context of large-scale DNNs.

**III. Preliminaries**

In this section, we provide a brief background on resistive crossbars and how they evaluate VMM.

Fig. 1 presents a typical resistive crossbar array design for realizing VMMs. It consists of a 2-D array of synaptic devices, DACs, and ADCs, and write peripheral circuitry. It supports two main operations: 1) programming, i.e., a write operation performed sequentially on a subset of synaptic devices and 2) evaluation, i.e., the vector–matrix multiply operation. The synaptic element at the intersection of each row and column is programmed by enabling the corresponding write circuits along the write wordline (WWL) and the bitline (BL), to apply the necessary current and set it to the desired conductance.

A VMM is performed by using DACs to convert the digital inputs into voltages on the RWLs, and sensing the resulting current flowing through the BLs using ADCs.

Fig. 1. Resistive crossbar array.

Synaptic devices are programmable resistors that are commonly realized using emerging nonvolatile memory technologies, such as PCM, ReRAM, and Spintronics [3]–[6]. Fig. 1 illustrates an example of a spintronic synaptic device [5] consisting of a magnetic tunneling junction (MTJ) and an underlying heavy metal (HM) layer. The three-terminal device is programmed through the HM layer and sensed through the MTJ. The position of the domain wall determines the conductance of the MTJ that lies between $G_{\text{MIN}}$ (when the domain wall is to the far right) and $G_{\text{MAX}}$ (when the domain wall is to the far left). The number of unique locations at which the domain wall can reside determines the precision of the device. Although we consider this spintronic synaptic device in our explanations, the RxNN framework is applicable to the wide range of resistive devices used to design crossbars.

Equation (1) specifies the ideal vector–matrix multiply operation for an $M \times N$ dimensional crossbar. $V_{\text{in ideal}}$ is an $1 \times M$ vector consisting of the input voltages, $G$ is an $M \times N$ matrix comprising of the synaptic conductances, and $I_{\text{out ideal}}$ is a $1 \times N$ vector containing the output currents.

$$I_{\text{out ideal}} = V_{\text{in ideal}} \times G_{\text{ideal}}.$$  

**IV. CROSSBAR NONIDEALITIES**

In this section, we analyze nonidealities in resistive crossbars and examine their impact on VMMs.

**A. Crossbar Nonidealities**

Fig. 2(a) presents an equivalent circuit for the crossbar array and the peripherals (DACs and ADCs) that includes various nonidealities. The key sources of nonidealities are—
1. wire resistances of the crossbar interconnects;
2. sensing resistances of the circuits that sense the output currents;
3. driver resistances of the circuits that drive the crossbar rows;
4. sneak paths;
5. variance in synaptic conductance due to process variations and imperfect programming; and
6. nonideal DACs. While we consider all these nonidealities in subsequent sections, we select the nonidealities due to DACs and sneak paths for a more detailed treatment below, in order to illustrate the complexity of error modeling.

**Nonideal DAC:** Fig. 2(a) shows the equivalent circuit for a DAC that is represented using a resistive divider circuit with an input determined resistance ($R_{\text{DAC}}$) and a fixed resistance ($R_{\text{PD}}$). An applied digital input determines the value of $R_{\text{DAC}}$ and subsequently decides the DAC’s output voltage ($V_{\text{out DAC}}$). Note that, $V_{\text{out DAC}}$ also depends on the effective resistive load ($R_{\text{load}}$), leading to deviations from the ideal value. $R_{\text{load}}$ is a function of the synaptic conductances within the crossbar array and therefore varies with the crossbar state (the values of all synaptic conductances). The equation in Fig. 3(a) shows the error incurred due to DAC nonidealities which is a function of both applied inputs ($R_{\text{DAC}}$) and synaptic conductances ($R_{\text{load}}$).

**Sneak Paths During VMM:** Ideally, currents in resistive crossbars would be expected to flow from left to right along the
Fig. 2. Crossbar nonidealities: (a) resistive equivalent circuit for a crossbar; (b) and (c) sensitivity to crossbar dimensions with all synaptic conductances programmed to $G_{MIN}$ and $G_{MAX}$, respectively; (d) sensitivity to synaptic conductances; (e) sensitivity to applied inputs; and (f) sensitivity to process variation and imperfect programming.

Fig. 3. Example of nonidealities in resistive crossbar: (a) Nonideal DAC; and (b) Sneak paths during VMM.

rows and from top to bottom through the columns. However, due to the nonidealities described above (specifically, wire resistances), internal node voltages within the crossbar may vary, resulting in additional current paths, which we refer to as sneak paths. Fig. 3(b) illustrates sneak paths during VMMs for a $3 \times 2$ crossbar array. We consider a crossbar state with all synaptic devices programmed to 20 k$\Omega$, and the applied input voltages at the rows are 0.2, 0.01, and 0.2V, respectively. For this crossbar state, we observe that the direction of current between nodes $a_{22}$ and $b_{22}$ is flipped, i.e., the current flows from $b_{22}$ toward the input (Vin2), instead of the expected direction. Sneak paths are a function of both the crossbar state and the applied inputs, and therefore further contribute to the overall dynamism in errors due to nonidealities.

B. Errors Due to Nonidealities

Next, we study the impact of nonidealities on the computational accuracy of the VMM realized using resistive crossbars. To this end, we compare the outputs of VMMs obtained from HSPICE simulations of nonideal crossbar arrays with the ideal computations (1) and analyze the sensitivity of the errors to various parameters.

Sensitivity to Crossbar Size: We first examine how the errors incurred due to individual nonidealities (WIRE, SENSE), combinations of nonidealities (DAC+DRIVER, WIRE+SENSE), and the cumulative effect of all nonidealities (ALL) vary with the crossbar dimension. Figs. 2(b) and (c) show the errors incurred during the VMM realized using crossbars, with all synaptic conductances programmed to $G_{MIN}$ and $G_{MAX}$, respectively. In both graphs, the y-axis represents the error in the last ($N^{th}$) column of an $N \times N$ crossbar, and the x-axis represents the crossbar dimension ($N$). In both cases, we observe that the errors due to all nonidealities (ALL), individual nonidealities, and subsets of nonidealities, increase with the crossbar dimension. This is expected because: 1) the overall wire resistances increase with crossbar array size; 2) the sensing resistance contribution to the overall BL resistance increases; and 3) the DAC nonideality increases due to a decrease in the effective load resistance. Furthermore, we also observe that for smaller crossbars, the nonideality due to DAC is predominant, whereas, for larger crossbars, the wire and sensing resistance effect becomes equally significant.

Sensitivity to Crossbar State: Next, we characterize errors’ dependence on the crossbar state, i.e., the conductances of all synaptic devices. To this end, we fix the inputs to a $64 \times 64$ crossbar array and vary the conductances of the synaptic devices to obtain different crossbar states. Fig. 2(d) shows the maximum (MAX), minimum (MIN), and average (AVG) errors across columns of the crossbar over 1000 random crossbar states. We observe that the errors show significant variation across these states. In Fig. 2(d), we also plot the errors for a sample crossbar state (Sample-Run) to demonstrate the variation of errors across crossbar columns. Note that this pattern is quite different from the patterns observed for MAX, MIN, and AVG errors.

Sensitivity to Crossbar Inputs: To analyze the errors’ dependence on the applied inputs, we fixed the conductances of all synaptic devices and varied the inputs. Fig. 2(e) shows the variations in errors across inputs. We observe that the variance across inputs (MAX and MIN) for a particular column

2Higher crossbar dimensions have more columns leading to increase in parallel paths, consequently lowering the effective load resistance.
is noticeable, but small in comparison to the variance across crossbar states.

Sensitivity to Crossbar Columns: Fig. 2(d) and (e) depicts how errors vary across crossbar columns. While there is a slight trend of increase in error as we go from the first to the last column, it is not always the last column that incurs the maximum error. Rather any column can incur the maximum error depending on the crossbar state and the applied inputs.

Sensitivity to Process Variation and Imperfect Programming: Finally, we also evaluate the impact of variations by performing Monte Carlo simulation on a sample set of 10,000 crossbar states obtained by considering variations in synaptic conductances ($\sigma/\mu = 10\%$) [46]. Fig. 2(f) shows the maximum, minimum, and average error observed on a $64 \times 64$ crossbar array across these samples. The variations in synaptic conductances can occur due to two prominent reasons: 1) process variations and 2) imperfect programming, i.e., errors during write operations.

In summary, the nonidealities in resistive crossbars can have a significant impact on the computations that they perform. Furthermore, the errors due to nonidealities are highly dependent on various factors, including the conductances, applied inputs, crossbar column, and process variations. Thus, a crossbar model should consider these factors in order to accurately capture the impact of nonidealities on application-level accuracy.

V. CROSSBAR MODELING

In this section, we present an FCM that accurately captures the impact of resistive crossbars nonidealities on the performed VMMs.

A. FCM Overview

Fig. 4 overviews the proposed FCM that consists of two phases: 1) model generation and 2) model evaluation. Model generation is a one-time step for each DNN, whereas model evaluation is invoked to evaluate each inference operation using the DNN. The key idea in FCM is to first abstract nonidealities in the core crossbar array by transforming a weight matrix ($W$) into a nonideal conductance matrix ($G_{\text{non-ideal}}$). Subsequently, using the generated $G_{\text{non-ideal}}$ matrix and nonlinear peripheral (ADC and DAC) models, FCM emulates the nonideal vector–matrix multiplications on resistive crossbars. We discuss these phases of FCM in detail.

Crossbar Model Generator: FCM uses a crossbar model generator to abstract the nonidealities arising due to process variations, the nonlinear synaptic conductance characteristics, the sensing resistance, and the wire resistances. The model generator takes crossbar parameters and a weight matrix ($W$) as inputs and generates a nonideal conductance matrix ($G_{\text{non-ideal}}$) as the output. Using a three-step transformation mechanism (listed in Fig. 4), it converts $W$ to $G_{\text{non-ideal}}$ based on crossbar parameters, including synaptic device characteristics ($G_{\text{min}}, G_{\text{max}}, \text{precision}$), interconnect ($R_{\text{sense}}, r_{\text{row}}, r_{\text{col}}$) and circuit (crossbar size) parameters, and the chip variation profile. Fig. 5 illustrates the model generation process in greater detail using an example, where we consider the mapping of a $P \times Q$ weight matrix to crossbars of size $M \times N$. In step 1, the model generator slices the matrix $W$ into fragments and maps them to specific crossbar instances. The fragment size is same as the crossbar dimension and to achieve this for all fragments the corners of the matrix $W$ are zero padded if required. Note that, we need $t = \lceil P/(M \times N) \rceil$ zero-padded rows, and $Q$ zero-padded columns. Next, in step 2 (described in Fig. 6), weights are converted from digital floating-point (FP) values to conductances ($G$) considering device conductance characteristics, parameters ($G_{\text{min}}, G_{\text{max}}$, precision), and the variation profile. FCM supports synaptic devices with both linear [5], [47] and nonlinear [3], [18] conductance characteristics, either using equations or lookup tables. We sample the variation profile to obtain a unique variation factor (VF) for each synaptic element within each crossbar instance. At the
end of step 2, we obtain a conductance matrix \((G_i)\) for each crossbar instance. Finally, in step 3, the generator abstracts interconnect nonidealities \((R_{\text{sense}}, r_{\text{col}}, r_{\text{row}})\) and transforms the conductance matrices \((G_i)\) to the corresponding nonideal conductance matrices \((G_{\text{non-ideal}})\). Subsequently, these nonideal conductance matrices \((G_{\text{non-ideal}})\) are merged to obtain one nonideal matrix corresponding to the weight matrix \(W\). The transformation of \(G_i\) to \(G_{\text{non-ideal}}\) is exact, and we provide the mathematical proof in Section V-B.

Peripheral (ADC and DAC) Models: Fig. 4 details the ADC and DAC models used by FCM to incorporate ADC and DAC nonidealities. The DAC model is composed of a resistive divider circuit with a digital input (Inp) dependent resistance \((R_{\text{DAC}})\) and a fixed resistance \((R_{\text{PD}})\). The resistive divider is connected to a variable effective load conductance \((G_{\text{Load}})\) whose value is dependent on the crossbar state (synaptic conductances). FCM uses the equation shown in Fig. 4 to compute the nonideal input voltages \((V_{\text{in-ideal}})\). In the equation, \(R_{\text{DAC}}\) is determined using the digital inputs (Inp), and \(G_{\text{Load}}\) is computed using the \(G_{\text{non-ideal}}\) matrix. We note that \(V_{\text{in-ideal}}\) captures the data-dependence of the errors arising due to a nonideal DAC, since \(R_{\text{DAC}}\) and \(G_{\text{Load}}\) are dependent on the applied inputs and the crossbar state, respectively. Using matrices \(G_{\text{non-ideal}}\) and \(V_{\text{in-ideal}}\), FCM computes the nonideal VMM realized in crossbars to obtain nonideal output currents \((I_{\text{out-ideal}})\). The ADC model shown in Fig. 4 (which can model nonlinearity) is then used to convert the \(I_{\text{out-ideal}}\) to digital outputs (Out).

In summary, FCM abstracts both device and circuit nonidealities into crossbar models that achieve several orders-of-magnitude speed-up over SPICE, without compromising on the modeling accuracy (in our experiments, FCM models are functionality within 0.28% of HSPICE). We note that achieving such simulation speed is not possible without some abstraction, and that FCM provides a good tradeoff between fidelity versus simulation speed (detailed in Section VIII-A). FCM further derives simulation speed by realizing algebraic operations using well-optimized BLAS libraries.

### B. Abstraction of Interconnect Nonidealities

In this section, we provide the mathematical formulation for the abstraction of interconnect nonidealities (step 3 of crossbar model generation). We recall that in this step the generator abstracts interconnect nonidealities \((R_{\text{sense}}, r_{\text{col}}, r_{\text{row}})\) and transform the conductance matrix \((G_i)\) associated with the \(i\)th crossbar instance to the corresponding nonideal conductance matrix \((G_{\text{non-ideal}})\). We achieve this transformation by leveraging circuit laws (Kirchhoff’s loop laws and Ohm’s law) and linear algebraic operations (direct sum, row switching, vector concatenation, row reduction, etc.).

We now explain the formulation using Fig. 7 that shows the equivalent resistive circuit of an \(M \times N\) crossbar array. \(V_{\text{in}}\) represents the input voltage at the \(i\)th row of the crossbar, \(V_{\text{ai,j}}\) denotes the voltage at the node \(a_{i,j}\), and \(V_{\text{b}_{i,j}}\) is the voltage difference between the node \(a_{i,j}\) and the node \(b_{i,j}\). \(G_{i,j}\) is the conductance of the synaptic device at the \(i\)th row and the \(j\)th column. \(R_{\text{sense}}, r_{\text{col}}, r_{\text{row}}\) depict the sensing and distributed wire resistances, respectively, and \(I_{\text{out}}\) indicates the output current of the \(j\)th column.

#### Step 1 (Formulate Column Linear Systems): We first formulate column linear systems (LSCol\(_1\) to LSCol\(_N\)) using each vertical slice of the crossbar, shown in Fig. 7. Let us consider the \(j\)th vertical slice corresponding to the LSCol\(_j\) system (2)–(4). Using Kirchhoff’s current law (KCL) at all nodes \(b_{i,j}\) present in the \(j\)th column, we obtain (2) and (3). Equations (2) and (3) are then combined to obtain the linear system in (4).

\[
A_j \ast V_{\text{col}} = V_{\text{Acol}} - J \ast I_{\text{out}} \tag{2}
\]

\[
I_{\text{out}} = \sum_{x=1}^{M} G_{x,j} V_{x,j} \tag{3}
\]

\[
(A_j + J \ast K_j) V_{\text{col}} = V_{\text{Acol}} \tag{4}
\]
Step 2 (Merge Column Linear Systems): Next, the column linear systems (LSCol1 to LSColN) are merged to form a larger column linear system (merged-LSCol) as shown in (5) and (6). We achieve this by using the direct sum (⊕) matrix operation on matrices \( (A_j + J^*K_j) \) and \( K_j \) to obtain block matrices COLmat and Gmat, respectively. In (5), CVcol and CVAcol are vectors formed by concatenating Vcolj and VAcolj vectors, respectively. Note that, vectors Vcol and VAcol are obtained in step 1 [(2) and (4)]. Furthermore, loutnon-ideal in (6) is a vector representing the output currents

\[
\text{COLmat} \ast \text{CVcol} = \text{CVAcol} \quad \text{Gmat} \ast \text{CVcol} = (\text{loutnon-ideal})^T
\]

where

\[
\text{COLmat} = \bigoplus_{j=1,2,\ldots,N} (A_j + J^*K_j)
\]

\[
\text{Gmat} = \bigoplus_{j=1,2,\ldots,N} K_j
\]

\[
\text{loutnon-ideal} = [\text{lout}_1 \ \text{lout}_2 \ \ldots \ \text{lout}_N]
\]

\[
\text{CVcol} = [\text{Vcol}_1 \ \text{Vcol}_2 \ \ldots \ \text{Vcol}_N]^T
\]

\[
\text{CVAcol} = [\text{VAcol}_1 \ \text{VAcol}_2 \ \ldots \ \text{VAcol}_N]^T
\]

Step 3 (Formulate Row Linear Systems): Similar to step 1, the row linear systems (LSrow1 to LSrowM) are formulated considering horizontal slices of the crossbar. We use KCL at nodes ai,j present in the ith horizontal slice to obtain (7) which represents the LSrow system. In case of an \( M \times N \) crossbar, we have M such row linear systems (LSrow1 to LSrowM)

\[
B_i \ast \text{Vrow}_i = \text{VrowIN}_i - \text{VARow}_i
\]

where

\[
B_i = \begin{bmatrix}
G_{i,1} & G_{i,2} & G_{i,3} & \ldots & G_{i,N} \\
G_{i,1} & G_{i,2} & G_{i,3} & \ldots & G_{i,N} \\
G_{i,1} & G_{i,2} & G_{i,3} & \ldots & G_{i,N} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
G_{i,1} & G_{i,2} & G_{i,3} & \ldots & G_{i,N} \\
\end{bmatrix}
\]

\[
\text{Vrow}_i = \begin{bmatrix}
V_{i,1} \\
V_{i,2} \\
\vdots \\
V_{i,N} \\
\end{bmatrix}, \ \text{VARow}_i = \begin{bmatrix}
V_{a_{i,1}} \\
V_{a_{i,2}} \\
\vdots \\
V_{a_{i,N}} \\
\end{bmatrix}, \ \text{VrowIN}_i = \begin{bmatrix}
V_{in_1} \\
V_{in_2} \\
\vdots \\
V_{in_N} \\
\end{bmatrix}
\]

Step 4 (Merge Row Linear Systems): Next, the row linear systems obtained in step 3 are merged to obtain a larger row linear system (merged-LSrow) as shown in (8). ROWmat is a block matrix obtained by performing the direct sum (⊕) matrix operation on the matrix (Bi). Moreover, CVrowIN, CVrow, and CVArOW are vectors formed by concatenating vectors (obtained in step 3) VrowINIj, VARowi, and Vrowi, respectively

\[
\text{CVrowIN} - \text{ROWmat} \ast \text{CVrow} = \text{CVArOW}
\]

where

\[
\text{ROWmat} = \bigoplus_{i=1,2,\ldots,M} (B_i)
\]

\[
\text{CVrow} = \begin{bmatrix}
\text{Vrow}_1^T \\
\text{Vrow}_2^T \\
\vdots \\
\text{Vrow}_N^T
\end{bmatrix}^T
\]

\[
\text{CVArOW} = \begin{bmatrix}
\text{VARow}_1^T \\
\text{VARow}_2^T \\
\vdots \\
\text{VARow}_N^T
\end{bmatrix}^T
\]

\[
\text{CVrowIN} = \begin{bmatrix}
\text{VrowIN}_1^T \\
\text{VrowIN}_2^T \\
\vdots \\
\text{VrowIN}_N^T
\end{bmatrix}^T.
\]

Step 5 (Eliminate Internal Variables): Next, the vectors CVAcol and CVcol comprising of internal variables Vai,j and Vi,j, respectively, are eliminated. In order to eliminate these variables, we use the merged-LScol and merged-LSrow systems obtained in steps 2 and 4, respectively. However, the merged-LScol and merged-LSrow equations cannot be used directly due to the mismatch in their right-hand sides (RHS) (CVAcol ≠ CVArOW). We resolve this mismatch by performing elementary row operations on (8) to obtain (9). Note that, the CVrowIN vector and the ROWmatA matrix are obtained by performing row switching, i.e., an elementary row operation, on the CVrowIN vector and the ROWmat matrix, respectively. Next, the CVAcol vector is eliminated using (5) and (9) to obtain (10). Subsequently, the CVcol vector is eliminated using (6) and (10) to yield (11). Note that, (12) details the NETmat matrix introduced in (11)

\[
\text{CVrowIN} - \text{ROWmatA} \ast \text{CVcol} = \text{CVAcol}
\]

\[
(\text{COLmat} + \text{ROWmatA}) \ast \text{CVcol} = \text{CVrowIN}
\]

\[
(\text{loutnon-ideal})^T = \text{NETmat} \ast \text{CVrowIN}
\]

\[
\text{NETmat} = \text{Gmat} \ast (\text{COLmat} + \text{ROWmatA})^{-1}
\]

Step 6 (Reduce Matrix Dimension): Finally, we reduce the size of matrices NETmat and CVrowIN by leveraging a key property of the CVrowIN vector, i.e., it contains repeated elements. Recall that, the CVrowIN vector is formed by concatenating the VrowINj vectors (step 4), and the CVrowIN vector is obtained by performing row switching operations on the CVrowIN vector. Since the VrowINj vector (step 3) has repeated elements, consequently, the vectors CVrowIN and CVrowIN also have repeated elements. Exploiting this property, the columns of the NETmat matrix that are to be multiplied by same elements in CVrowIN can be summed using elementary column operations to yield a compressed
NETmatC matrix [shown in (13)]. Moreover, removing redundancies in vector CVinn1 assumes the Vinnon-ideal vector. Furthermore, (13) can be rewritten as (14) to obtain the $G_{\text{non-ideal}}$ matrix. Note that, $G_{\text{non-ideal}}$ is a function of $(G, R_{\text{sense}}, r_{\text{col}},$ and $r_{\text{row}})$, and therefore can be constructed using the intermediate matrices $\text{COLmat, ROWmat,}$ and $\text{Gmat}$.

$$\begin{align*}
(I_{\text{out, non-ideal}})^T &= \text{NETmatC} \ast (\text{Vin}_{\text{non-ideal}})^T) \\
I_{\text{out, non-ideal}} &= \text{Vin}_{\text{non-ideal}} \ast G_{\text{non-ideal}}
\end{align*}$$

where

$$\begin{align*}
\text{Vin}_{\text{non-ideal}} &= [\text{Vin}_1 \ \text{Vin}_2 \ \ldots \ \text{Vin}_M] \\
G_{\text{non-ideal}} &= \text{NETmatC}^T = f(G, R_{\text{sense}}, r_{\text{row}}, r_{\text{col}}).
\end{align*}$$

VI. RXNN FRAMEWORK

In this section, we present the overall RxNN framework that enables the evaluation of large-scale DNNs on resistive crossbar systems. RxNN is a functional simulator obtained by modifying the Caffe [41] deep learning framework to mimic nonideal VMMs realized on resistive crossbars. Caffe models the convolution and fully connected layers of DNNs as matrix–matrix and VMMs. RxNN maps these matrix–matrix and VMMs to a resistive crossbar system and evaluates the application-level accuracy of DNN inference operations. It takes the trained DNN network and weights, resistive crossbar system description, and crossbar parameters as inputs, and evaluates the DNN inference operation using FCM models. RxNN’s primary objective is to evaluate the application-level accuracy of DNNs, however, it is also capable of generating execution traces to enable performance and energy estimation. RxNN can also be used for model-in-the-loop retraining to improve DNN inference accuracy in the presence of nonidealities.

Fig. 8 depicts the RxNN flow that consists of three steps. In step 1, RxNN maps the neural network to the specified target architecture. The weights are read from the trained Caffe model and virtually programmed into the crossbar array instances. Subsequently, the conductance matrices ($G$) corresponding to each resistive crossbar instance are generated, which are then transformed into the nonideal conductance matrices ($G_{\text{non-ideal}}$) by abstracting crossbar nonidealities. Next, in step 2, the $G_{\text{non-ideal}}$ matrices associated with each DNN layer are incorporated back into the Caffe’s original weight data structure. RxNN transparently utilizes Caffe’s underlying data structures and optimized BLAS libraries, which is key to its performance and scalability. We note that steps 1 and 2 are performed only once for a given DNN and crossbar-based architecture. Thereafter, in step 3, RxNN evaluates the DNN for the given set of test inputs using embedded $G_{\text{non-ideal}}$ matrices and peripheral (ADC and DAC) models. During network evaluation, the DAC/ADC models are invoked as pre- and post-processing steps on the inputs/outputs of each convolutional and fully connected layer.

Next, we describe retraining with RxNN to improve the inference accuracy of DNNs on resistive crossbar systems. The major challenges that arise during DNN retraining for crossbar systems are: 1) the data structures (inputs, outputs, and weights) should abide by the range and resolution constraints at all times and 2) errors and gradients computed during back-propagation should be appropriately scaled to ensure network convergence. RxNN meets these constraints by utilizing a crossbar-based forward pass and an FP-based backward pass. It appropriately converts and scales the data structures between forward and backward passes to ensure that the network retrains with minimal impact on the overall training time, which is extremely critical in the context of large-scale DNNs.

VII. EXPERIMENTAL METHODOLOGY

In this section, we describe the experimental setup used to evaluate the RxNN framework.

Device/Circuit Simulation: We use an in-house device model of the synaptic element [5] that is based on the solution of Landau–Lifshitz–Gilbert (LLG) magnetization dynamics and nonequilibrium-Green’s function (NEGF) electron transport. Circuit-level simulations are performed in HSPICE using the 45-nm bulk CMOS technology and the synaptic device model. Our simulations use the ADC and DAC circuits proposed in [48] and [49]. The interconnect parasitics ($r_{\text{row}}, r_{\text{col}}$) are extracted using the device and crossbar array layouts. Fig. 9 shows these layouts that are performed using the design rules specified in [50]. The table in Fig. 9 details the device, technology [51], and variation parameters [46] assumed in our experiments. We also characterize a resistive crossbar array to compute energy at the crossbar-level.

3 The stochastic-gradient descent solver assumes the forward and backward passes to be contiguous and differentiable. However, crossbar abstraction of VMM does not ensure these conditions.
TABLE I

| Benchmark DNN Applications | | | | | |
|---|---|---|---|---|---|
| Dataset | Network | #Conv Layers | #FC Layers | #Synapses (in billions) | #Neurons (in millions) | Relative Model Size |
| MNIST | LeNet | 2 | 2 | 0.0005 | 0.02 | 1 |
| CIFAR-10 | ConvNet | 3 | 2 | 0.01 | 0.05 | 20 |
| | N/N | 9 | 0 | 0.3 | 0.6 | 600 |
| ImageNet | AlexNet | 5 | 3 | 0.5 | 0.5 | 1000 |
| | N/N | 12 | 0 | 1.1 | 1.7 | 2200 |
| | OverFeat | 5 | 3 | 2.6 | 1.9 | 5200 |
| | VGG-16 | 13 | 3 | 15.5 | 13.6 | 31000 |
| | GoogleNet | 59 | 5 | 1.6 | 3.2 | 3200 |
| | ResNet-50 | 53 | 1 | 5.1 | 13.8 | 10200 |

Fig. 10. Computation errors observed in crossbar for various crossbar models.

which is used as a technology parameter in RxNN to estimate system-level energy consumption.

Application-Level Simulation: We evaluated the application-level accuracy and energy of several popular DNNs on the resistive crossbar system using RxNN. Table I provides details of the benchmark DNNs, including the number of convolution and fully connected layers, the targeted dataset, and the number of neurons and synaptic connections. We also present the relative model size to highlight the difference between these benchmark DNNs. To evaluate energy consumption, we use an architecture similar to [7].

VIII. RESULTS

We now present the experimental results to demonstrate the modeling accuracy and speedups achieved by FCM over circuit simulation. We also evaluate the application-level accuracy of large-scale DNNs on nonideal resistive crossbar systems using RxNN.

A. FCM: Crossbar-Level Evaluation

Modeling Accuracy: Fig. 10 shows the errors in VMMs realized using a $64 \times 64$ nonideal crossbar. We compute errors using three different crossbar models, viz., HSPICE, FCM, and MNSIM [27]. The x-axis represents the crossbar column, and the y-axis depicts the error incurred due to nonidealities in the VMM. We observe that the simple error model (MNSIM) deviates considerably from the HSPICE model. This is expected, as it does not consider the dependence of errors on several factors, including the applied inputs, the crossbar state, and the crossbar column. In contrast, the FCM model considers these dynamic factors and is therefore able to closely match the HSPICE model. The maximum deviation between the errors estimated by MNSIM and the errors computed using HSPICE is about 3.51%. In the case of FCM, the maximum deviation is found to be 0.28%, which is significantly smaller.

Speedup: To evaluate the speedup of FCM over HSPICE, we measure the execution time of FCM and HSPICE for various crossbar sizes. Fig. 11 details the speedup achieved using FCM over HSPICE. We observe a speedup of about five orders in magnitude. Moreover, as expected, the speedup increases for larger crossbar arrays.

Model Generation Overhead: Recall that FCM’s crossbar model generator transforms the weight matrix (W) to a non-ideal conductance matrix ($G_{\text{non-ideal}}$), which incurs a one-time overhead. In our evaluation, we found the modeling overhead to be 0.038, 1.2, and 61 s for $16 \times 16$, $32 \times 32$, and $64 \times 64$ crossbar arrays, respectively. While considerable for larger crossbars, these one-time overheads are amortized over a large number of inference operations typically processed by RxNN.

B. RxNN: Application-Level Evaluation

Next, we apply RxNN to evaluate the accuracy degradation due to crossbar nonidealities at the application level for the benchmark DNNs. We implement three different resistive crossbar systems designed using crossbars of size $16 \times 16$ (Cross16), $32 \times 32$ (Cross32), and $64 \times 64$ (Cross64). Fig. 12(a) shows the accuracy degradation for these designs with respect to our baseline, i.e., an ideal crossbar with no device and circuit-level nonidealities. We first compare the accuracy degradation of the Cross64 design across DNNs. We observe that for simple networks (LeNet and ConvNet) the accuracy degradation due to nonidealities is quite small. For example, LeNet and ConvNet networks suffer accuracy degradation of 0.05% and 2.2%, respectively. In contrast, the accuracy loss due to nonidealities is considerable for large-scale DNNs. For instance, VGG-16, OverFeat, and ResNet-50 networks incur accuracy losses of 25.6%, 27.8%, and 32%, respectively. We observe similar accuracy degradation trend across simple and large-scale DNNs for the Cross16 and Cross32 designs as well.

Next, we compare the accuracy degradation across designs with different crossbar sizes (Cross16, Cross32, and Cross64). As evident from Fig. 12(a), the accuracy degradation for the Cross16 design is less than the Cross32 design, which is in turn less than the Cross64 design. This trend is expected as the impact of nonidealities is lower for smaller crossbar arrays (Section IV-B). However, smaller crossbar arrays are not desirable in terms of energy efficiency. Fig. 12(b) depicts the normalized energy consumption per image for the
Cross16, Cross32, and Cross64 designs. The Cross16 design consumes higher energy than the Cross32 design, which in turn consumes higher energy than the Cross64 design for most cases. Since the major components of the energy consumed in resistive crossbar systems are peripherals (ADCs and DACs), larger crossbar arrays that amortize the energy cost of ADCs and DACs over more columns and rows have superior energy efficiency. Note that, for the LeNet and ConvNet DNNs, the energy of the Cross64 design is higher than the Cross32 design. This is because the crossbars in the Cross64 design are underutilized in case of these relatively small networks. Therefore, Cross64 suffers from energy overheads due to redundant computations performed in the unused rows/columns.

We next present the energy breakdown of three networks, viz., VGG-16, GoogleNet, and AlexNet realized on the Cross64 design. Fig. 13 shows the energy breakdown of these networks considering—read energy for inputs (CMOS-Mem-Read), write energy for outputs (CMOS-Mem-Write), and computation energy for VMMs (Cross-Computation). We observe that the major energy component is the VMMs (Cross-Computation) which is, in turn, dominated by the ADCs and DACs.

In summary, there exists a fundamental tradeoff between the application-level accuracy and the system energy which needs to be examined, in order to determine the architectures for future resistive crossbar systems. RxNN intends to drive these decisions by providing a software platform that can precisely evaluate crossbar architectures executing large-scale DNNs.

**RxNN Speed Versus Caffe:** We also evaluated the slowdown of RxNN with respect to the baseline Caffe framework (without any crossbar modeling), and found that it amounts to 2.5X and 2.75X for inference and retraining, respectively, across our benchmark applications. We believe this is a reasonable overhead given the highly optimized nature of Caffe, and the fact that much like Caffe, RxNN can also leverage multicores, GPUs, and clusters for increased processing throughput.

**C. Sensitivity of Accuracy to Nonidealities**

To further illustrate the impact of nonidealities on the application-level accuracy, we present a sensitivity analysis in Fig. 14. We plot the accuracies of six large-scale networks, viz., AlexNet, VGG-16, GoogleNet, NiN, Overfeat, and ResNet-50 for implementations differing in their degree of nonidealities. The implementations that we use are: 1) FP implementation realized on an x86 CPU architecture (FP32); 2) 6-b ideal crossbar design (Cross6) without any crossbar nonidealities; and 3) 6-b nonideal crossbar based designs with and without variations (NI-Cross6-64x64). Note that the FP32 CPU-based software implementation does not use crossbars and hence does not suffer from any nonidealities. As shown in Fig. 14, the accuracy drops from left to right as more nonidealities are incorporated. We observe two significant accuracy drops, one between FP32 and Cross6 implementations, and other between Cross6 and NI-Cross6-64×64 implementations. The degradation between FP32 and Cross6 is due to the limited precision of the synaptic devices, ADCs, and DACs. In contrast, the drop in accuracy from Cross6 to NI-Cross6-64×64 is due to the device and circuit-level nonidealities.

**D. Retraining DNNs Using RxNN**

Next, we show the effectiveness of RxNN in retraining large-scale DNNs for resistive crossbar systems. To that end, we retrained three networks, viz., AlexNet, VGG-16, and GoogleNet, as shown in Fig. 15. Our experiments show that with only 150 iterations of retraining RxNN can achieve ∼9%, ∼8%, and ∼26% improvement in accuracy for AlexNet, VGG-16, and GoogleNet, respectively. Notwithstanding these improvements, there is still a substantial drop in accuracy that cannot be recovered by retraining alone, calling for additional error mitigation and compensation techniques.
simulation framework to evaluate large-scale DNNs on resistive crossbar systems. Our experiments with RxNN indicate that accuracy degradation due to nonidealities is a significant concern for large-scale DNNs. Re-training can only partly restore the accuracy lost, necessitating a need for further error mitigation and compensation schemes.

![Fig. 15. Retraining using RxNN.](image)

![Fig. 16. Visual demonstration of errors using ConvNet.](image)

### IX. Conclusion

Resistive crossbars realized using nonvolatile memory devices promise to enable compact, energy-efficient hardware for DNNs. In this article, we evaluate the impact of various device and circuit nonidealities that are present in crossbars on the overall accuracy of large-scale DNNs. We propose FCM, a fast and accurate model to evaluate VMMs realized on resistive crossbars. Using FCM, we construct RxNN, a software
S. Jain and A. Raghunathan, “CXDNN: Hardware–software co-

B. Li, Y. Wang, Y. Wang, Y. Chen, and H. Yang, “Training itself: Mixed-

I. Chakraborty, D. Roy, and K. Roy, “Technology aware training in

T. Gokmen et al.

JAIN et al.

S. Jain

J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2009, pp. 248–255.

C. D. Schuman et al., “A survey of neuromorphic computing and neural networks in hardware,” 2017. [Online]. Available: arxiv.abs/1705.06963.

H. S. P. Wong et al., “Metal-oxide RRAM,” Proc. IEEE, vol. 100, no. 6, pp. 1951–1970, Jun. 2012.

M. Sharad, G. Panagopoulos, and K. Roy, “Spin neuron for ultra low power computational hardware,” in Proc. 70th Device Res. Conf., 2012, pp. 221–222.

A. F. Vincent et al., “Spin-transfer torque magnetic memory as a stochas-

J. Li et al., “A novel reconfigurable Sensing scheme for variable level storage in phase change memory,” in Proc. 3rd IEEE Int. Memory Workshop (IMW), May 2011, pp. 1–4.

J. Zhang, Z. Wang, and N. Verma, “A machine-learning classifier imple-

L. Gao et al., “Fully parallel write/read in resistive synaptic array for accelerating on-chip learning,” Nanotechnology, vol. 26, Oct. 2015, pp. 1–6.

T. Gokmen et al.

J. Zhang, T. Gokmen, and W. E. Haensch, “Analog CMOS-based resistive processing unit for deep neural network training,” 2017. [Online]. Available: arxiv.abs/1702.03443.

C. Liu, M. Hu, J. P. Strachan, and H. Li, “Rescuing memristor-based neuromorphic design with high defects,” in Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC), Jun. 2017, pp. 1–6.

L. Xia et al., “MNSIM: Simulation platform for memristor-based neu-

W. Wen et al., “An EDA framework for large scale hybrid neuromorphic computing systems,” in Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC), Jun. 2015, pp. 1–6.

B. Liu, W. Wen, Y. Chen, X. Li, C.-R. Wu, and T.-Y. Ho, “EDA challenges for memristor-corsbased memristor computing,” in Proc. 25th Ed. Great Lakes Symp. VLSI (GLSVLSI), 2015, pp. 185–188.

Y. J. et al., “NEUTRAMS: Neural network transformation and co-

S. Kim, T. Gokmen, H.-M. Lee, and W. E. Haensch, “Analog CMOS-based resistive processing unit for deep neural network training,” 2017. [Online]. Available: arxiv.abs/1706.06620.

T. Li, X. Bi, N. Jing, X. Liang, and L. Jiang, “Sneak-path based test and diagnosis for 1R RRAM crossbar using voltage bias technique,” in Proc. ACM 54th Annu. Design Autom. Conf. (DAC), Jun. 2017, pp. 1–6.

A. Ranjan, S. Jain, J. R. Stevens, D. Das, B. Kaul, and A. Raghunathan, “X-MANN: A crossbar based architecture for memory augmented neu-

Shubham Jain received the B.Tech. degree (Hons.) in electronics and electrical communication engineering from the Indian Institute of Technology Kharagpur, Kharagpur, India, in 2012, and the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 2019.

He is currently a Research Staff Member with IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. He worked as a Design Engineer with Bangalore Design Center, Qualcomm, Bengaluru, India, from 2012 to 2014. He also worked as a summer intern with IBM T. J. Watson Research Center from 2017 to 2018. His primary research interests include AI hardware, architecture for post-CMOS devices, in-memory computing, and approximate computing.

Dr. Jain research has received the Best Technical Paper Award in DAC 2018 and the best-in Session Award in TECHCON 2016. He has received the Mitacs Globalink Scholarship from Mitacs in 2011, the Andrews Fellowship from Purdue University in 2014, and the A. Richard Newton Young Student Fellowship from DAC in 2015.

Abhroril Sengupta (Member, IEEE) received the B.E. degree from Jadavpur University, Kolkata, India, in 2013, and the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 2018.

He is an Assistant Professor with the School of Electrical Engineering and Computer Science, Penn State University, State College, PA, USA. He worked as a DAAD (German Academic Exchange Service) Fellow with the University of Hamburg, Hamburg, Germany, in 2012, and as a Graduate Research Intern with Circuit Research Labs, Scottsdale, AZ, USA, Intel Labs, Hillsboro, OR, USA, in 2016, and Facebook Reality Labs, Pittsburgh, PA, USA, in 2017. He is pursuing an interdisciplinary research agenda with the intersection of hardware and software across the stack of sensors, devices, circuits, systems and algorithms for enabling large power event-driven cognitive intelligence. He has published over 45 articles in refereed journals and conferences and holds four granted/pending U.S. patents.

Dr. Sengupta has been awarded the IEEE SiPS Best Paper Award since 2018, the Schmidt Science Fellows Award nominee in 2017, the Bilsland Dissertation Fellowship in 2017, the CSPIN Student Presenter Award in 2015, Birck Fellowship in 2013, and DAAD WISE Fellowship in 2012. He serves on the Technical Program Committee of Design Automation Conference in 2019, the International Symposium on Quality Electronic Design in 2019, and the ACM Great Lakes Symposium on VLSI in 2019.
Kaushik Roy (Fellow, IEEE) received the B.Tech. degree in electronics and electrical communications engineering from the Indian Institute of Technology Kharagpur, Kharagpur, India, and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Illinois at Urbana–Champaign, Urbana, IL, USA, in 1990.

He was with the Semiconductor Process and Design Center, Texas Instruments, Dallas, TX, USA, where he worked on FPGA architecture development and low-power circuit design. He joined the Electrical and Computer Engineering Faculty, Purdue University, West Lafayette, IN, USA, in 1993, where he is currently an Edward G. Tiedemann Jr. Distinguished Professor. His research interests include spintronics, device-circuit co-design for nano-scale silicon and nonsilicon technologies, low-power electronics for portable computing and wireless communications, and new computing models enabled by emerging technologies. He has published more than 600 papers in refereed journals and conferences, holds 15 patents, graduated 60 Ph.D. students, and has coauthored two books on Low-Power CMOS VLSI Design (Wiley & McGraw Hill).

Dr. Roy received the U.S. National Science Foundation Career Development Award in 1995, the IBM Faculty Partnership Award, the ATT/Lucent Foundation Award, the SRC Technical Excellence Award in 2005, the SRC Inventors Award, the Purdue College of Engineering Research Excellence Award, the Humboldt Research Award in 2010, the IEEE Circuits and Systems Society Technical Achievement Award in 2010, the Distinguished Alumnus Award from Indian Institute of Technology, Kharagpur, the Fulbright-Nehru Distinguished Chair, the Best Paper Awards at 1997 International Test Conference, the IEEE 2000 International Symposium on Quality of IC Design, the IEEE Latin American Test Workshop in 2003, IEEE Nano in 2003, the IEEE International Conference on Computer Design in 2004, the IEEE/ACM International Symposium on Low Power Electronics and Design in 2006, the IEEE Circuits and System Society Outstanding Young Author Award (Chris Kim) in 2005, the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS Best Paper Award in 2006, the ACM/IEEE International Symposium on Low Power Electronics and Design Best Paper Award in 2012, and theIEEE Transactions on VLSI Best Paper Award in 2013. He was a Purdue University Faculty scholar from 1998 to 2003. He was a Research Visionary Board Member of Motorola Labs in 2002 and held the M. K. Gandhi Distinguished Visiting Faculty at Indian Institute of Technology (Bombay). He has been in the editorial board of IEEE DESIGN & TEST, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, and the IEEE TRANSACTIONS ON ELECTRON DEVICES. He was the Guest Editor for Special Issue on Low-Power VLSI in the IEEE DESIGN & TEST in 1994, the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS in June 2000, the IEEE Proceedings—Computers and Digital Techniques in July 2002, and the IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS in 2011.

Anand Raghunathan (Fellow, IEEE) received the B.Tech. degree from the Indian Institute of Technology Madras, Chennai, India, in 1992, and the M.A. and Ph.D. degrees from Princeton University, Princeton, NJ, USA, in 1994 and 1997, respectively.

He is a Professor of electrical and computer engineering and the Chair of the VLSI area with Purdue University, West Lafayette, IN, USA, where he directs research with Integrated Systems Laboratory. He was a Senior Research Staff Member with NEC Laboratories America, Princeton, NJ, USA, where he led projects on system-on-chip architecture and design methodology. He has also held the Gopalakrishnan Visiting Chair with the Department of Computer Science and Engineering, Indian Institute of Technology Madras. He has coauthored a book, eight book chapters, and over 200 refereed journal and conference papers, and holds 21 U.S. patents. His current areas of research include domain-specific architecture, system-on-chip design, computing with post-CMOS devices, and heterogeneous parallel computing.

Prof. Raghunathan received the IEEE Meritorious Service Award and Outstanding Service Award. His publications received the eight best paper awards and five best paper nominations. He received a Patent of the Year Award and two Technology Commercialization Awards from NEC, and was chosen among the MIT TR35 (top 35 innovators under 35 years across various disciplines of science and technology) in 2006. He has been a member of the technical program and organizing committees of several leading conferences and workshops, chaired premier IEEE/ACM conferences (CASES, ISLPED, VTS, and VLSI Design), and served on the editorial boards of various IEEE and ACM journals in his areas of interest. He is a Golden Core Member of the IEEE Computer Society.