Real-Time Compensation Framework for Large-Scale ReRAM-Based Sparse LU Factorization

Weiran Chen, *Graduate Student Member, IEEE*, Zaitian Chen, Bei Yu, *Senior Member, IEEE*, Song Chen, *Member, IEEE*, Yi Kang, *Member, IEEE*, and Qi Xu, *Member, IEEE*

***Abstract*—Recently, resistive switching random access memory (ReRAM)-based hardware accelerators have demonstrated unprecedented performance compared to digital accelerators. However, due to limitations in the manufacturing process and large-scale integration, several significant nonideal effects, including IR-Drop, stuck-at-fault, and device noises in real ReRAM-based crossbar arrays, are typically incurred. These nonideal effects degrade signal integrity and performance, particularly in crossbar structures used for building high-density ReRAMs. Therefore, finding a fast and efficient software solution that can predict the effects of IR-drop without involving expensive hardware is highly desirable. In this work, addressing the main limitations of existing simulation methods, such as slow speed and high-resource costs, we propose an efficient analysis of large-scale ReRAM crossbar arrays and the corresponding nonideal factors based on sparse matrix modeling. We classify nonideal factors into linear (e.g., IR-drop) and nonlinear categories (e.g., shot noise). For linear factors, super-nodal sparse LU factorizations are used to solve. The array-level results show that compared to SPICE simulation, our method achieves a numerical solution accuracy of 10^{-15} with 506.8 ∼ 1253.3× faster and 17.46 ∼ 42934.3× reduced memory usage. For nonlinear factors, we propose two solutions based on different requirements. In one method, we obtain an approximate initial solution by solving a linear system while disregarding the nonlinear contributions and subsequently apply an extended Anderson acceleration method to solve the nonlinear equation, which is suitable for high-precision solutions. Another method simplifies the nonlinear equation into an equivalent linear form. Theoretical validation confirms the effectiveness of this method, significantly enhancing simulation speed while maintaining accuracy. Moreover, we build a high-precision ReRAM accelerator architecture with real-time compensation. Experimental results demonstrate that the proposed architecture effectively mitigates accuracy loss caused by nonideal factors.**

***Index Terms*—Analog computing, current compensation, IR-drop, resistive random access memory, sparse LU factorization.**

Received 7 December 2024; revised 3 April 2025; accepted 28 May 2025. Date of publication 4 June 2025; date of current version 6 January 2026. This work was supported in part by the National Natural Science Foundation of China under Grant 92473113; in part by the Anhui Provincial Natural Science Foundation under Grant 2408085MF173; and in part by the Fundamental Research Funds for the Central Universities under Grant WK2100000060. This article was recommended by Associate Editor Q. Qiu. *(Weiran Chen and Zaitian Chen contributed equally to this work.) (Corresponding author: Qi Xu.)*

Weiran Chen, Zaitian Chen, Song Chen, Yi Kang, and Qi Xu are with the School of Microelectronics, University of Science and Technology of China, Hefei 230026, China (e-mail: chenweiran2001@mail.ustc.edu.cn; czt2020@mail.ustc.edu.cn; xuqi@ustc.edu.cn).

Bei Yu is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong (e-mail: byu@cse.cuhk.edu.hk).

Digital Object Identifier 10.1109/TCAD.2025.3576332

I. Introduction

ReRAM crossbar arrays (RCAs) technologies offer the potential for highly efficient neural network hardware [1], which critically depends on the assumption that RCAs can function as both computing and storage units, eliminating the von Neumann architecture bottleneck. Under ideal conditions, an RCA can perform parallel matrix-vector multiplication (MVM) [2] between a weight conductance matrix and an input voltage vector, achieving a time complexity of $O(1)$ instead of $O(n^2)$.

Despite these tremendous advantages, several nonideal effects are encountered, such as IR-Drop at the array level, stuck-at-fault (SAF) and device noise at the cell level. These nonideal effects lead to uneven voltage and current distributions in resistive switching random access memory (ReRAM) [3], which limit the computational accuracy of ReRAM-based deep neural network (DNN) accelerators, particularly in large-scale array integrations [4]. As a result, directly mapping an ideal DNN into real RCAs is not appropriate at the inference phase. Several recent studies have proposed techniques to address these issues as summarized follows.

Software Approach: Lee et al. [5] proposed a novel approach to mitigate accuracy degradation by incorporating the IR-Drop problem into binarized neural network (BNN) training. He et al. [6] introduced a framework called PytorX, which evaluates digital SAF error correction and develops a noise injection adaptation training method for IR-Drop and fault compensation. Jain and Raghunathan [7] proposed a CxDNN flow, which calculates the average proportional coefficient between the actual and ideal output of the RCAs and applies a multiplier for compensation.

Hardware Approach: Huang et al. [8] proposed two effective hardware methods to compensate for current deviation: adding adjustable lines or using ReRAM-TIA with adjustable gain. Soudry et al. [9] performed the neural network training process directly on-chip and further refined the iterations by observing the differences between the ideal and actual on-chip output.

However, the main disadvantage of the above methods is that the parameters must be readjusted for different tasks, and the compensation effect is greatly reduced when encountering data with large feature differences. In addition, using SPICE or PytorX to predict the effects of undesirable factors often involves significant runtime and hardware resource consumption. Fundamentally different from these approaches, we propose a real-time current compensation method for ReRAM utilizing sparse LU factorization. This methodology addresses accuracy degradation caused by IR-Drop, SAF and device noises without retraining the neural network. Key technical contributions of this work are enumerated as follows:

1) We model various nonideal effects of ReRAM, including IR-Drop, SAF, and device noises, as either linear or nonlinear noises. Through sparse matrix modeling, an efficient compensation method for ReRAM is achieved.
2) We propose a fast sparse matrix-based calculation technique to overcome the linear nonideal factors (e.g., SAF and IR-Drop). Considering the consistency of the coefficient matrix, we divide the solution process into symbolic and numerical factorization, and only perform symbolic factorization once throughout the process, thereby minimizing redundant calculations and significantly accelerating the linear equations solving.
3) To further address the challenges of slow convergence and computational speed in analyzing nonlinear systems, we first provide the initial solution by solving linear equations and then propose an extension method based on Anderson acceleration to enhance the solution process. At the same time, we present two efficient approximation algorithms that approximate the nonlinear problem as a linear problem, significantly accelerate computation, and maintain accuracy for large-scale array solutions.
4) We propose a real-time current compensation methodology integrating software simulation with hardware compensation to mitigate the inherent errors in ReRAM-based computing systems. Experiments on various neural network architectures with different datasets are conducted to validate the effectiveness of the proposed approach. The results demonstrate that our approach provides a fast, high-precision solution while maintaining memory efficiency comparable to SPICE and other SOTA methods.

II. Background

A. ReRAM Crossbar Array System

Fig. 1 depicts a representative resistive crossbar-based accelerator, specifically a Resistive crossbar processing unit (PRC) connected to a Central PRC (CPU) and main memory via a system bus. The resistive crossbar PRC is composed of RCAs that store weights and compute vector-matrix multiplications, input and output buffer memories for storing activations, a local bus, a controller to receive instructions and signals from the CPU and provides signals to peripheral circuits, and a Pooling and ReLU Circuits PRC to execute additional DNN operations, including ReLU and max/average pooling. This article focuses on RCAs as the fundamental computing unit of the resistive crossbar system.

As illustrated on the right side of Fig. 1, a RCA consists of a 2-D array of synaptic elements, along with Digital-to-Analog Converters (DACs), Analog-to-Digital Converters (ADCs), and Trans-impedance amplifiers (TIA). The synaptic elements, which are programmable resistors, store DNN weights as conductances. The precision of each synaptic element is determined by the number of available conductance levels. RCA enables two primary functions: 1) vector-matrix multiplication and 2) programming. During vector-matrix multiplication, analog voltages are applied to all rows (word lines) via DACs, and the resulting currents are sensed through columns (bit lines) by ADCs. Conversely, programming (or writing) operations set the conductance of synaptic elements row-by-row, where the write circuitry applies currents to adjust each element to its target conductance. The write time for each ReRAM cell is primarily determined by the resistance switching time, which typically ranges from microseconds to milliseconds. In contrast, the read time is significantly shorter, generally falling within the nanosecond to microsecond range.

B. ReRAM Crossbar Array Modeling

To effectively address current losses due to nonideal factors, a robust simulation method is crucial. IR-Drop is considered one of the most complex and critical factors among all nonidealities in circuit simulations. As it significantly affects the performance and accuracy of simulations, we primarily investigate the current methods that specifically address IR-Drop in RCA and similar system. These methods can generally be divided into two categories from circuit modeling: accurate and approximate models.

Accurate Models: These models aim to capture all the detailed physical behaviors and nonidealities of the system, prioritizing high precision. By employing general nodal analysis, the circuit is decomposed into separate nodes, with the interconnections and device behaviors precisely represented through SPICE syntax or mathematical equations. However, such accurate models typically incur high-computational costs, requiring more time and resources. An example of an accurate model is SPICE simulation, which considers every element and behavior of the circuit in detail. The simulation can produce precise results for any set of parameters and demonstrate strong applicability and versatility. However, when used for neural network inference, each RCA must simulate every input, rendering the aforementioned accurate model impractical due to the unacceptable simulation speed. In addition, certain custom iterative algorithms, such as CIter1 [10] and CIter2 [11], have considerably enhanced simulation speed. However, it has been observed that convergence is assured only when the array parameters lie within a specific, often narrow, range.

Approximate Models: Approaches, such as CCCS [12] and NRAMA [13] simplify the system by making certain assumptions or approximations, which help decrease the computational burden. While this method may sacrifice some accuracy, it results in faster simulation times, making it more appropriate for large-scale systems. These models are particularly advantageous when exact precision is less critical or when there are strict time constraints. Furthermore, specialized simulators, such as MNSIM [14], [15] and NeuroSim [16], [17], are designed specifically for RCAs systems. These simulators focus on architecture-level simulations for neural network applications. To expedite simulations, RCAs are often used as

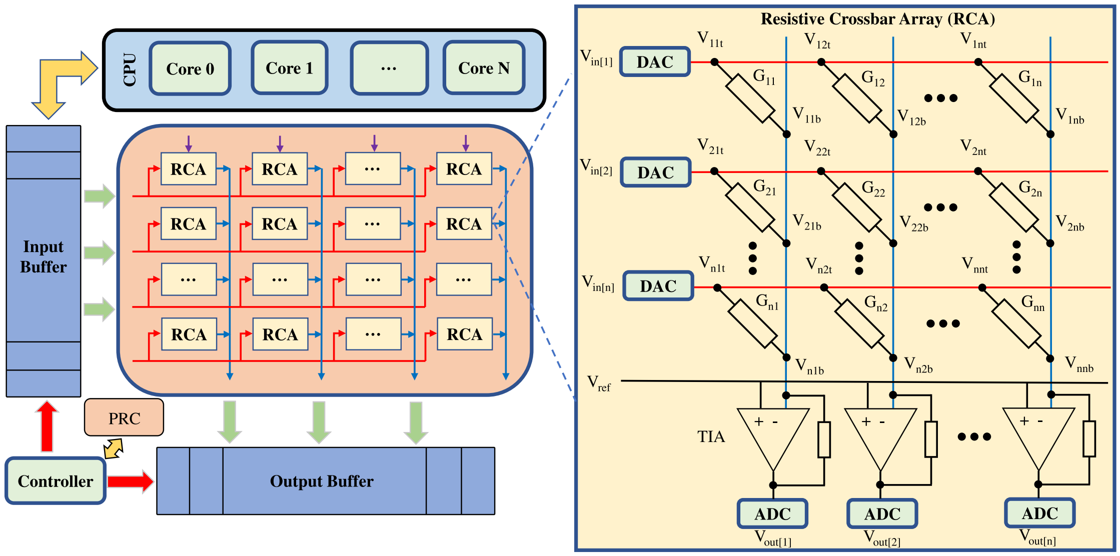


Fig. 1. Architecture for RCAs system. PRC: Pooling and ReLU circuits PRC. Controller: Orchestrate various operations.

ideal MVM engines or simulated with simplified circuits that overlook the effects of coupled wire resistance between rows and columns.

C. Sparse LU Factorization

Assuming that the given matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is nonsingular and can be factorized as $\boldsymbol{A} = \boldsymbol{LU}$, where $\boldsymbol{L}$ is a lower triangular matrix with unit diagonal elements and $\boldsymbol{U}$ is an upper triangular matrix [18]. Generally, sparse LU factorization involves three main steps: 1) preprocessing; 2) symbolic factorization; and 3) numerical factorization [19]. Before symbolic factorization, it is often necessary to reorder the matrix to minimize the number of fill-in elements. Symbolic and numeric factorizations are employed to determine the sparsity structure of $\boldsymbol{L}$ and $\boldsymbol{U}$ and the values of nonzero entries, respectively.

III. ReRAM Crossbar Arrays Nonidealities

Mathematically, the nonideal factors in ReRAM can generally be classified into linear and nonlinear factors. In this work, SAF, IR-Drop, Johnson–Nyquist noise, and Flicker noise are considered as linear influencing factors, while Shot noise is regarded as a nonlinear influencing factor.

A. Linear Nonideal Effects

IR-Drop: IR-Drop in RCAs is primarily caused by nonnegligible wire resistance as circuit feature sizes shrink, resulting in uneven voltage distribution and decreased read/write accuracy [20]. This phenomenon becomes particularly problematic in large-scale RCAs, where cumulative IR-Drop potentially alter resistive states, thereby limiting the scalability of the RCAs.

SAF: SAF is another reliability issue in ReRAM [21], where memory cells become "stuck" in a particular resistive state (either Stuck-At-One or Stuck-At-Zero), regardless of the applied programming voltage. The Stuck-At-Zero defect typically arises from short defects, resulting in specific ReRAM cells being permanently maintained in a high conductance state. Accordingly, the Stuck-At-One defect results from manufacturing defects, causing certain ReRAM cells to be stuck in a low-conductance state. Note that SAF cells can be identified using testing methods, such as March [22], [23] before deployment, allowing the fault locations to be directly set to SA0 or SA1 during simulation. Since the SA0/SA1 fault testing is conducted before deployment as a one-time overhead to verify the initial functionality of the array. Once the array is deployed, there is no need to repeat testing processes unless hardware failures occur or special maintenance is required [22], [23]. Therefore, the testing time is considered a one-time overhead before the system enters its operational phase.

Johnson–Nyquist Noise and Flicker Noise: Johnson–Nyquist noise [24] originates from the random thermal motion of electrons within a conductor, leading to small, random voltage fluctuations. In contrast, Flicker noise [25] is caused by material defects or low-frequency carrier fluctuations. Both can be modeled as random current sources following a zero-mean Gaussian distribution, and combined in parallel with the ReRAM to form a single current source, expressed by the root mean square (RMS) current

$$I_{\text{linear}} = \sqrt{\frac{K}{C_{\text{ox}}\, WL\Delta f} \cdot G^2 + 4K_B T \Delta f G} \tag{1}$$

where G represents the ReRAM conductance, f denotes the operating frequency of the crossbar array, C_{ox} signifies the oxide capacitance, K_B is the Boltzmann constant, and W and L are channel width and length, respectively. For Flicker noise, K is the process parameter, and T is the temperature in Kelvin. We also consider the impacts of Johnson–Nyquist and Flicker noise on interconnection wire, which can be equivalent to the current source using the above modeling method.

B. Nonlinear Nonideal Effect

Shot Noise: Shot noise is a type of electronic noise that arises from the quantized nature of electric charge [26]. Since shot noise follows a Poisson process due to the discrete charge of an electron, the RMS current fluctuations can be calculated as

$$I_{\text{nolinear}} = \sqrt{2G\Delta f q \Delta V} \tag{2}$$

where ΔV is the voltage drop across the two terminals of the ReRAM cell and q is the charge of an electron. Given that linear and nonlinear noise sources are independent, we can compute the total power spectral density (PSD) by integrating them throughout the entire modeling process.

C. Problem Formulation

Definition 1 (Maximum Relative Error (MRE)): In this article, MRE is a critical metric for evaluating the precision of ReRAM-based systems, which is defined as

$$\text{MRE} = \max_i \left| \left(I_i - I_i^{\text{ref}} \right) / I_i^{\text{ref}} \right| \times 100\% \tag{3}$$

where I_i is the output current of the ith ReRAM cell in the array, and I_i^{ref} denotes SPICE simulation. In general, high precision ensures that minor numerical errors do not accumulate significantly during iterative computations, particularly important in the simulation of neural networks deployed on RCAs.

In this work, we attempt to solve the nonideality problem encountered when deploying neural network models onto RCAs in inference tasks. Our problem is formulated as follows.

Problem 1 (ReRAM Nonidealities Compensation): The inputs to our methodology include the mathematical model of the RCA, incorporating its nonideal characteristics, and the neural network models to be deployed. The output is a real-time compensation scheme based on LU factorization, which dynamically corrects errors caused by nonidealities in the RCAs. Our ultimate objective is to minimize MRE to improve inference accuracy, accelerate the simulation speed of RCA, and reduce the hardware resource overhead required for compensation.

IV. Methodology

A. RCA Modeling and Compensation

In this work, we assume that ReRAM exhibits ideal resistor behavior and consider only its static characteristics. This means that factors, such as device I-V nonlinearity, disturbances, parasitic capacitance, and other nonideal effects are not taken into account. Under these simplifications, the resistive memristor array can be regarded as a linear system. Thus, any linear input can be linearly superimposed, and the first-order term in $\boldsymbol{I} = f(\boldsymbol{V}, \boldsymbol{G})$ is essentially the only term in f. Although this reasoning relies solely on the superposition principle and may seem elementary, it serves as the underlying mechanism for the success of previous IR-Drop modeling efforts.

Fig. 2 illustrates the equivalent circuit schematic of the RCA that accounts for wire resistance. Kirchhoff's Current Law

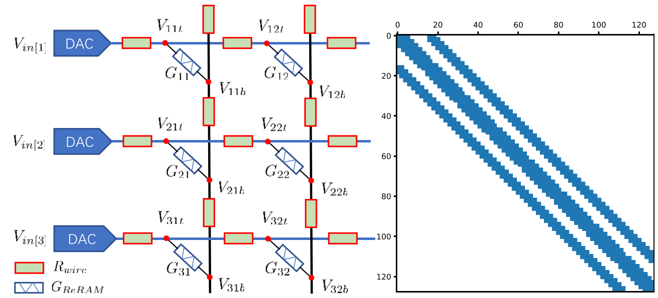


Fig. 2. Left: The circuit schematic of RCA with IR-Drop, Right: The coefficient matrix of RCA physical modeling (Array size: 8 × 8).

(KCL) defines the continuity of current flow at every junction. For each node (i, j) in the RCAs, the following applies:

$$\begin{cases} (2 + G_{i,j}R_w)V_{i,j,t} - G_{i,j}R_w V_{i,j,b} - V_{i,j+1,b} - V_{i,j-1,b} = 0 \\ G_{i,j}R_w V_{i,j,t} - (2 + G_{i,j}R_w)V_{i,j,b} - V_{i+1,j,b} - V_{i-1,j,b} = 0 \end{cases} \tag{4}$$

where $V_{i,j,t}$ and $V_{i,j,b}$ denote the respective top and bottom voltages of each ReRAM cell, with $i = 1, \ldots, n-1$ and $j = 1, \ldots, n-1$. $G_{i,j}$ represents the ReRAM conductance at in the ith row and the jth column, and R_w represents wire resistance. Regarding boundary conditions, specifically, for the first and last nodes in each row ($j = 1$ or $j = n$), since the input voltage is applied to the external node on the left side while the external node on the right side is left floating, the following applies:

$$\begin{cases} (2 + G_{i,j}R_w)V_{i,j,t} - G_{i,j}R_w V_{i,j,b} - V_{i,j+1,b} = V_{\text{in}[i]}, j = 1 \\ (2 + G_{i,j}R_w)V_{i,j,t} - G_{i,j}R_w V_{i,j,b} - V_{i,j-1,b} = 0, \quad j = n \end{cases} \tag{5}$$

where $V_{\text{in}[i]}$ is the input voltage and is therefore placed on the right side of the KCL equation. Thus, KCL equations for all nodes can be organized as a linear system in a compact form:

$$\boldsymbol{AX} = \boldsymbol{B} \tag{6}$$

where $\boldsymbol{A} \in \mathbb{R}^{2n^2 \times 2n^2}$ is the coefficient matrix, while $\boldsymbol{B}$ represents the vector of input voltages applied to the circuit. $\boldsymbol{X}$ is the requested nodal voltage matrix, as expressed

$$\boldsymbol{X} = \left[V_{1,1,t}, V_{1,1,b}, V_{1,2,t}, V_{1,2,b}, \ldots, V_{n,n,t}, V_{n,n,b}\right]^\top \tag{7}$$

where $V_{i,j,t}$ and $V_{i,j,b}$ denote the potentials at the top and bottom nodes at the intersection of the ith row and the jth column, respectively. Solving this system provides the nodal potentials across the network. At the same time, we need to adjust the equation according to the different working states of RCAs. Generally, when RCAs are weight-writing, the outermost part of the RCAs has voltage access. When RCAs are employed for neural network acceleration, the outermost part can be regarded as being in a floating state, nearly grounded, with a voltage of zero.

Due to the inherent errors in ReRAM-based computing system, the actual output current ($\boldsymbol{I}_{\text{actual}}$) is different from the ideal output current ($\boldsymbol{I}_{\text{ideal}}$) without considering device-level imperfections, which can be calculated as

$$I^j_{\text{ideal}} = \sum_{i=1}^{n} \left(V_{i,j,t} - V_{i,j,b}\right) \cdot G_{i,j}$$
$$I^j_{\text{actual}} = \sum_{i=1}^{n} \left(V_{i,j,t} - V_{i,j,b}\right) \cdot \left(G_{i,j} + G_{\text{noise}}\right) + I^j_{\text{wire}} \quad (8)$$

where I^j_{ideal} and I^j_{actual} refer to the current values of the jth element in $\boldsymbol{I}_{\text{actual}}$ and $\boldsymbol{I}_{\text{ideal}}$. G_{noise} denotes equivalent noise conductance. I^j_{wire} represents the equivalent noise current of the interconnection wire resistance caused by the linear noise. To realize the hardware compensation, we calculate the compensation coefficient C_j as follows.

$$C_j = 1 + \frac{I^j_{\text{ideal}} - I^j_{\text{actual}}}{I^j_{\text{actual}}}. \quad (9)$$

Based on the compensation coefficient, we mitigate errors in vector-matrix multiplications to obtain compensated outputs.

$$I^j_{\text{comp}} = I^j_{\text{actual}} \times C_j. \quad (10)$$

B. Property of Coefficient Matrix $\boldsymbol{A}$

Sparse: Currently, common methods for solving large-scale systems of linear equations include Gaussian elimination [27], LU factorization [28], and Cholesky factorization [29]. However, the time complexity of these methods is $O(m^3)$, where m is the dimensions of the matrix. Since the size of the RCAs is proportional to the square of the size of matrix $\boldsymbol{A}$, the overall time complexity for solving the equations becomes $O(n^6)$. In PytorX [6], just training the LeNet-5 model consumes all 64GB of main memory and takes several days to solve, making it computationally prohibitive. Considering an RCA array of size $n \times n$, the linear system holds $2n^2$ variables, and a $2n^2 \times 2n^2$ coefficient matrix $\boldsymbol{A}$ must be constructed. According to the KCL equation, each row or column has no more than 4 nonzero terms, and the sparsity of the matrix is approximately $(2/n^2)$. As shown in Fig. 2, the blue areas indicate nonzero positions, while the blank areas represent zero positions. Clearly, a significant amount of memory is wasted in storing and calculating these zeros, which does not contribute to the efficiency of LU factorization.

Symmetric: Note that when $i \neq j$ & $j \in N(i)$, we always have $\boldsymbol{A}(i, j) = -G_{i,j} = -G_{j,i} = \boldsymbol{A}(j, i)$. Here, $N(i)$ represents the set of adjacent nodes connected to node i, not including the external nodes. Thus, $\boldsymbol{A}$ is a symmetric matrix. The LU factorization of symmetric matrices offers distinct advantages in numerical computation, primarily by lowering both computational load and memory requirements. Since symmetric matrices require only half of the matrix (either the upper or lower triangular part) for calculation and storage, LU factorization effectively reduces operational complexity and conserves memory resources.

Positive Definite: Since $\boldsymbol{A}$ is symmetric, for any current distribution in the circuit, the power $\boldsymbol{P_w}$ in the resistance network is given by

$$\boldsymbol{P_w} = \boldsymbol{V}^\top \boldsymbol{A} \boldsymbol{V} \quad (11)$$

Algorithm 1 Fill-in and Identification of supernodes

Input: Symmetric and positive definite $\boldsymbol{A} \in \mathbb{R}^{n\times n}$.

Output: Sparse strict lower triangular pattern $\boldsymbol{P} \in \mathbb{R}^{n\times n}$ with same pattern as $\boldsymbol{L}, \boldsymbol{U}^\top$ as well as column size $\boldsymbol{s} \in \mathbb{R}^m$ of each supernode.

Initialize the variable $m = 0$;
Derive the parent array $\boldsymbol{p}$ from the elimination tree $T(A)$;
for $j = 1$ to n **do**
  For each column j, supplement the nonzeros of column j of $\boldsymbol{P}$ with $i > j$ and $A_{ij} \neq 0$;
  Calculate the number of non-zero elements in column j of $\boldsymbol{P}$ and denote it as r;
  if $j > 1$, $j = p_{j-1}$, and $s_m + r = l$ **then**
    Continue the current supernode: $s_m = s_m + 1$;
  else
    Start a new supernode: $m = m + 1$, $s_m = 1$, and $l = r$;
  end if
  Obtain the parent node index $k = p_j$;
  if $k > 0$ **then**
    Supplement the nonzeros of column k of $\boldsymbol{P}$ with the nonzero elements of column j of $\boldsymbol{P}$ greater than $\boldsymbol{p}$;
  end if
end for

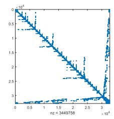
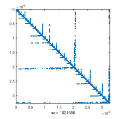
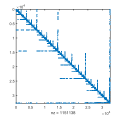
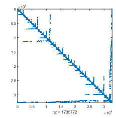
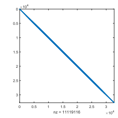
where $\boldsymbol{V}$ is the voltage vector. In a resistance network, the power consumption $\boldsymbol{P_w}$ is always greater than zero under any nonzero voltage distribution $\boldsymbol{V} \neq \boldsymbol{0}$. This characteristic demonstrates that the resistance matrix $\boldsymbol{A}$ is positive definite because the positive definite matrix $\boldsymbol{A}$ guarantees that the quadratic form $\boldsymbol{V}^\top \boldsymbol{A} \boldsymbol{V}$. Positive definite matrices play a crucial role in maintaining numerical stability in operations, such as solving linear equations or performing matrix factorization.

C. Sparse LU Factorization for RCAs

In the previous section, we demonstrated that the coefficient matrix $\boldsymbol{A}$ is large, symmetric, positive definite, and sparse. Since the RCA dimension is fixed throughout the process, the nonzero element positions remain constant in the matrix during simulation, with only the values being updated at each iteration. Therefore, reordering and symbolic factorization during preprocessing in RCAs simulations are typically performed only once, whereas numerical factorization must be repeated multiple times. Hence, in circuit simulation, the primary computational bottleneck lies in the speed of numerical factorization.

Reordering: In sparse LU factorization, the speed of numerical factorization is closely related to the number of fill-in elements. The quantity of fill-in directly impacts the computational complexity of LU factorization. That is because an increase in fill-in results in a higher number of nonzero elements, thereby increasing computation time. Moreover, as the number of fill-in elements increases, the required storage space likewise expands. This can result in reduced memory efficiency, further impacting the factorization speed.

TABLE I
COMPARISON BETWEEN DIFFERENT REORDERING METHODS FOR THE RCA COEFFICIENT MATRIX

Reordering Method	COLAMD	AMD	Nested Dissection	Symamd	Symrcm
Reordering time (s)	0.1006	0.2158	0.2133	1.5392	0.6447
Numeric Factorization					
Factorization (nnz)	3449758	1621658	1151138	1735772	11119116

[1] The benchmark was conducted on a laptop equipped with an Intel Core i7-13700 processor, 8 GB of RAM, and MATLAB 2023b.
[2] In the subsequent experiment, the Nested Dissection method provided by the METIS package will be used for reordering to further reduce fill-in.
[3] nnz denotes the number of nonzero elements.

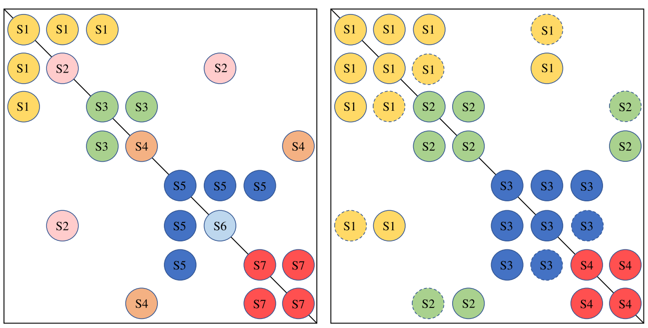


Fig. 3. Left: Original supernodes, Right: Supernodes after merging. Dotted border round: additional zero element.

Table I shows the reordering effects of several common reordering algorithms on the matrix $\boldsymbol{A}$ (Array size: 128×128). The results indicate that, compared with other sorting algorithms (such as COLAMD [30], AMD [31], Symamd [32], and Symrcm [32]), Nested Dissection [33] demonstrates significant advantages in terms of reduced fill-in and shorter sorting times. Thus, this article employs the nested dissection algorithm from the METIS [34] package for reordering to minimize the number of fill-in. Besides, this approach is suitable for parallel computing.

Symbolic Factorization: Given the symmetric positive definite property of the coefficient matrix, we apply the supernode[35] method to partition the sparse matrix into supernode, thereby reducing computational complexity.

As shown in the left subfigure of Fig. 3, the process of supernodal sparse LU factorization involves grouping columns with identical nonzero structures in $\boldsymbol{L}$ into symmetric supernodes. This grouping facilitates storage and computation by treating these columns as dense submatrices. On the right subfigure of Fig. 3, from left to Right, columns with different row structures (e.g., column 1 and column 2) are merged to form new supernodes, which results in filling some additional zero elements. In simple terms, a supernode allows consecutive columns to be grouped together into a single dense block, which includes a triangular diagonal block and a dense subdiagonal block that fit together in a trapezoidal shape. This block structure enables parts of the matrix to be processed together using efficient dense matrix operations, such as those provided by level-3 BLAS [36] and LAPACK libraries. By employing this symbolic factorization method, the RCAs can be efficiently solved while accounting for IR-Drop nonidealities. The specific process is shown in Algorithm 1.

In simulating the influence of the linear noise on the ReRAM device, the input voltage in (6) needs to be refined to incorporate Johnson–Nyquist and Flicker noise as equivalent current sources. The updated formulation is given as

$$\boldsymbol{B} = \left[V_{in[1]} - I_n R_w, -I_n R_w \cdots V_{in[n]} - I_n R_w, -I_n R_w \cdots\right]^{\top}$$

where I_n is the equivalent noise current in ReRAM cell. Since the coefficient matrix $\boldsymbol{A}$ remains unchanged, performing linear noise simulation has minimal impact on the simulation speed.

D. Nonlinear Noise Simulation

In this article, we develop two techniques to solve the nonlinear noise, such as shot noise. One solution uses an iterative method to achieve high precision, while the other employs a Maclaurin series and the Least squares method to approximate shot noise, transforming the nonlinear problem into a linear one. This approach greatly enhances solving speed while maintaining high numerical accuracy.

Iterative Method: As shown in (2), shot noise can essentially be modeled as a controlled current source, whose magnitude is primarily governed by the ReRAM voltage. Given that the noises above are independent, the following expression can be derived using the superposition principle of PSD:

$$\begin{aligned} I_{\text{noise}} &= \sqrt{I_{\text{linear}}^2 + I_{\text{nolinear}}^2} = I_{\text{linear}}\sqrt{1 + \frac{I_{\text{nolinear}}^2}{I_{\text{linear}}^2}} \\ &= \sqrt{4k_B T G f + \frac{KG^2}{C_{ox}WLf}}\sqrt{1 + \frac{2qf}{4k_B T f + \frac{KG}{C_{ox}WLf}}\Delta V} \\ &= a\sqrt{1 + b\Delta V}. \end{aligned} \tag{12}$$

where $I_{1\text{linear}}$ and I_{nolinear} represent the linear noise and nolinear noise, which are described in Section III-A.

Based on the noise current I_{noise}, the equivalent noise conductance G_{noise} can be derived as

$$G_{\text{noise}} = I_{\text{noise}}/\left(V_{i,j,t} - V_{i,j,b}\right) = I_{\text{noise}}/\Delta V \tag{13}$$

where ΔV denotes the voltage drop in the memristor cell. Considering the noise effect, the actual memristor's conductance and the corresponding I-V function are derived as:

$$\begin{aligned}\hat{G}_{i,j} &= G_{i,j} + G_{\text{noise}}, \mathbf{f}\left(V_{i,j}\right)\\ &= \Big[\left(2+\hat{G}_{i,j}R_w\right)V_{i,j,t} - \hat{G}_{i,j}R_w V_{i,j,b} - V_{i,j+1,b} - V_{i,j-1,b}\\ &\qquad \hat{G}_{i,j}R_w V_{i,j,t} - \left(2+\hat{G}_{i,j}R_w\right)V_{i,j,b} - V_{i+1,j,b} - V_{i-1,j,b}\Big] \quad (14)\\ &= \mathbf{0} \quad (15)\end{aligned}$$

where $\mathbf{f}$: $\mathbb{R}^n \to \mathbb{R}^n$ is the I-V function. We want to find a solution $\boldsymbol{x}^* \in \mathbb{R}^n$ such that $\mathbf{f}(\boldsymbol{x}^*) = \mathbf{0}$. Since the noise is relatively small compared to the ideal value, we can obtain an appropriate initial solution by solving the linear equation and then apply the iterative method. Compared with the random initialization, the proposed linear approximation-based initialization ensures convergence and speed. Thus, we reformulate the nonlinear equation as a fixed-point iteration. The iteration formula is given by

$$\mathbf{x}_{k+1} = \mathbf{g}(\mathbf{x}_k) \tag{16}$$

where the function $\mathbf{g}(\cdot)$ is chosen such that a fixed point of $\mathbf{g}$ corresponds to a solution of $\mathbf{f}(\mathbf{x}^*) = \mathbf{0}$, i.e., $\mathbf{g}(\mathbf{x}^*) = \mathbf{x}^*$. In order to further improve the convergence speed and stability, we introduce the Anderson acceleration method. In Anderson acceleration, the kth iteration's point $\mathbf{x}_{k+1}$ is updated using a linear combination of the previous iteration points and residuals

$$\mathbf{x}_{k+1} = \sum_{i=0}^{m} \alpha_i \mathbf{x}_{k-i} - \sum_{i=0}^{m} \alpha_i \boldsymbol{r}_{k-i} \tag{17}$$

where α_i is the combination coefficients determined by minimizing the residual's squared norm, m is the number of previous iterations used, and $\boldsymbol{r}_k = \mathbf{x}_k - \mathbf{g}_k$ is the residual, indicating the iteration error. By solving the below constrained least squares problem, the combination coefficients α_i are achieved.

$$\min_{\alpha_i} \left| \sum_{i=0}^{m} \alpha_i \boldsymbol{r}_{k-i} \right|_2, \quad \text{s.t.} \sum_{i=0}^{m} \alpha_i = 1. \tag{18}$$

Note that the coefficients α_i are determined such that the current residuals are minimized within the subspace spanned by the vectors of previous residuals. We introduce a relaxation factor to the Successive Over-Relaxation (SOR) [37] relaxation algorithm, denoted as ω, to adjust the weighted proportion of the new and old solutions. Using the relaxation factor ω, the new iteration point is obtained as

$$\mathbf{x}_{k+1} = (1-\omega)\mathbf{x}_k + \omega\left(\sum_{i=0}^{m} \alpha_i \mathbf{x}_{k-i} - \sum_{i=0}^{m} \alpha_i \boldsymbol{r}_{k-i}\right) \tag{19}$$

Algorithm 2 RCAs Simulation Considering Nonlinear Noise

Input: $\boldsymbol{A}$: Coefficient matrix, $\boldsymbol{B}$: Input source voltage, ω: Controls the weight of the new update, M: Maximum iterations, *tol*: Convergence threshold for the residual norm, m: Number of previous vectors to retain, n: ReRAM array size.
Output: $\boldsymbol{I}_{actual}$: Output current.

Construct $\boldsymbol{A}$ using KCLs;
Initialize $\boldsymbol{V}_{\text{Initial}}$ by solving $\boldsymbol{AV} = \boldsymbol{B}$;
Initialize $k = 0$, $\boldsymbol{x}_0 = \boldsymbol{V}_{Initial}$, $\boldsymbol{F} = \boldsymbol{x}_0 - \boldsymbol{g}(\boldsymbol{x}_0)$, where $\boldsymbol{F}$ stores the residuals;
for $k = 1, 2, \ldots M$ **do**
  Compute residual: $\boldsymbol{r}_k = \boldsymbol{x}_k - \boldsymbol{g}(\boldsymbol{x}_k)$;
  if $\|\boldsymbol{r}_k\| < tol$ **then**
    break;
  end if
  $\boldsymbol{X} = [\boldsymbol{x}_{k-m}, \ldots, \boldsymbol{x}_k]$, $\boldsymbol{F} = [\boldsymbol{r}_{k-m}, \ldots, \boldsymbol{r}_k]$;
  Solve α_i; ▷ Equation (18)
  Update $\boldsymbol{x}_{k+1}$; ▷ Equation (19)
end for
Assign $\boldsymbol{x}_k$ to node voltage;
for $i = 1, 2, \ldots n$ **do**
  for $j = 1, 2, \ldots n$ **do**
    $I_{i,j} = (V_{i,j,t} - V_{i,j,b}) \cdot \hat{G}_{i,j}$;
  end for
end for
for $j = 1, 2, \ldots n$ **do**
  $I^j_{actual} = \sum_{i=1}^{n} I_{i,j}$; ▷ Output current in j-th column of RCA
end for
return $\boldsymbol{I}_{actual} = [I^1_{actual}, \ldots, I^n_{actual}]$.

where ω and m are set to 0.01 and 10, respectively. The above methods can effectively solve small-scale nonlinear simulation problems, but as the size of RCAs increases further, the time complexity grows exponentially. When $n \geq 512$, the convergence difficulty increases significantly, leading to a substantial rise in the time required for computation. Algorithm 2 provides a detailed description of the complete process for solving IR-Drop using the iterative method. Since the current $\boldsymbol{I}_{\text{actual}}$ is calculated with considering nonideal effects, current deviations need to be compensated for.

Approximate Method: The aforementioned method can effectively accelerate large-scale RCAs simulations. However, it is difficult for iterative methods to significantly improve simulation speed by reusing symbol Factorization like sparse LU Factorization. To address this problem, we propose a segmented approximation algorithm based on the Maclaurin series and the least squares method, which ensures solution accuracy while minimizing simulation time. When $n > 128$, ΔV is on the order of 10^{-2}. In this case, the Maclaurin series expansion of the above equation can be directly applied, yielding the following expression:

$$I_{\text{noise}} = a\left(1 + \frac{b}{2}\Delta V - \frac{b^2}{8}\Delta V^2 + \cdots\right) \approx a\left(1 + \frac{b}{2}\Delta V\right). \tag{20}$$

TABLE II
MAXIMUM AND AVERAGE MRE OF TWO APPROXIMATE METHODS ACROSS VARYING FREQUENCIES AND ARRAY SIZES

Methods	Maclaurin series						Least square approximation					
Frequency (Hz)	$1.0e^7$	$1.0e^9$	$1.0e^7$	$1.0e^9$	$1.0e^7$	$1.0e^9$	$1.0e^7$	$1.0e^9$	$1.0e^7$	$1.0e^9$	$1.0e^7$	$1.0e^9$
Array size	32	32	64	64	128	128	32	32	64	64	128	128
Max. MRE [%]	$1.83e^{-3}$	$1.84e^{-2}$	$1.56e^{-3}$	$1.57e^{-2}$	$1.06e^{-3}$	$1.06e^{-2}$	$1.26e^{-4}$	$1.25e^{-3}$	$4.56e^{-4}$	$4.52e^{-3}$	$1.45e^{-3}$	$1.42e^{-2}$
Avg. MRE [%]	$1.72e^{-3}$	$1.73e^{-2}$	$1.27e^{-3}$	$1.27e^{-2}$	$3.89e^{-4}$	$3.81e^{-3}$	$9.54e^{-5}$	$9.50e^{-4}$	$3.57e^{-4}$	$3.54e^{-3}$	$1.09e^{-3}$	$1.07e^{-2}$
Speed-up	25.97	28.71	56.72	54.65	102.31	99.84	22.73	23.46	51.49	49.32	97.36	99.41

For smaller array sizes, the value of ΔV increases, leading to a greater error in the approximation. To address this, we employ an alternative method of linear approximation: least squares fitting. By determining the approximate range of ΔV, this fitting method minimizes the overall error within that range. Below, ΔV is represented by the variable x, and the equivalent current i_{noise} is described by y. Through the linear fitting, $y = c_1 x + c_2$. Besides, the range of x is in (x_1, x_2). Then, the error function is depicted as

$$E = \int_{x_1}^{x_2} \left(a\sqrt{1+bx} - (c_1 x + c_2) \right)^2 dx. \tag{21}$$

To minimize the error function, we take the partial derivatives with respect to c_1 and c_2, respectively, and set them equal to zero.

$$\begin{aligned} \frac{\partial E}{\partial c_1} &= \int_{x_1}^{x_2} -2x\left(a\sqrt{1+bx} - (c_1 x + c_2) \right) dx = 0 \\ \frac{\partial E}{\partial c_2} &= \int_{x_1}^{x_2} -2\left(a\sqrt{1+bx} - (c_1 x + c_2) \right) dx = 0. \end{aligned} \tag{22}$$

Note that, $I_1 = \int_{x_1}^{x_2} ax\sqrt{1+bx}dx, I_2 = \int_{x_1}^{x_2} a\sqrt{1+bx}dx$ are constants. The above formula can be expressed in a simplified form as follows:

$$\begin{aligned} &\in t_{x_1}^{x_2}\left(c_1 x^2 + c_2 x \right) dx = I_1 \\ &\in t_{x_1}^{x_2}(c_1 x + c_2) dx = I_2. \end{aligned} \tag{23}$$

Integrating the left-hand side of the above system results in the following linear system of equations for c_1 and c_2:

$$\begin{aligned} &\frac{x_2^3 - x_1^3}{3} c_1 + \frac{x_2^2 - x_1^2}{2} c_2 = I_1 \\ &\frac{x_2^2 - x_1^2}{2} c_1 + (x_2 - x_1) c_2 = I_2. \end{aligned} \tag{24}$$

By solving the above equations, the values of c_1 and c_2 are obtained. For different x-ranges, the corresponding parameters can be calculated as described above, allowing the linear expression to be fitted to minimize the overall error. Algorithm 3 presents a detailed procedure for approximating nonlinear noise using the Least squares method.

E. Scalability Analysis

In this section, we first compare the relative MRE of two approximate methods across varying frequencies and array sizes. As shown in Table II, we present the relative MRE at array sizes where $n < 128$, demonstrating that the least square method improves accuracy by one to two orders of magnitude compared to the Maclaurin expansion in both average and maximum values. Compared to the iterative method, the approximation method significantly accelerates the solution time, and it is expected that the acceleration ratio will be further improved as the size increases. According to (12), the shot noise current is proportional to $\sqrt{\Delta f \Delta V}$. Therefore, increasing the frequency f amplifies the nonlinear noise component, leading to larger errors, as illustrated in Fig. 4. As a result, the choice between iterative and approximation methods in simulation should be based on specific requirements.

Algorithm 3 Least Squares Approximation for Nonlinear Noise Simulation

Input: $\boldsymbol{A}$: Coefficient matrix, $\boldsymbol{B}$: Input source voltage, n: ReRAM array size, a and b: System parameters.
Output: I_{noise}: Approximate noise current.

```
Construct A using KCLs;
Calculate node voltage by solving AV = B;
ΔV_min ← min_{i,j}(V_{i,j,t} − V_{i,j,b});
ΔV_max ← max_{i,j}(V_{i,j,t} − V_{i,j,b});
Set x_1 = ΔV_min, x_2 = ΔV_max;
I_1 ← ∫_{x_1}^{x_2} ax√(1 + bx) dx;
I_2 ← ∫_{x_1}^{x_2} a√(1 + bx) dx;
Solve the linear system to obtain c_1 and c_2;
                                    ▷ Equation (24)
I_noise ← c_1 ΔV + c_2;
return I_noise.
```

To further evaluate the impact of temperature on approximation errors, we compare the MRE of the Maclaurin series and least squares methods for different array sizes ($n = 32, 64, 128$) and temperatures ($T = 300 \sim 350\ K$). As shown in Fig. 5, temperature has a negligible effect on the accuracy of both methods.

In the end, we provide the validity analysis of the approximate method. Table II shows that the truncation error in the first-order Maclaurin series approximation decreases as the array size increases. The reason is that as the array size grows, the current is distributed across more paths, reducing the actual drive current for each unit. Additionally, the increased wire resistance with large array sizes leads to a substantial voltage reduction at the remote units. As depicted in Table III, the number of ReRAM cells with voltages below a certain threshold increases as the array size grows.

F. Real-Time Compensation

The above fast sparse matrix-based calculation technique can be considered as an array-level computation. However,

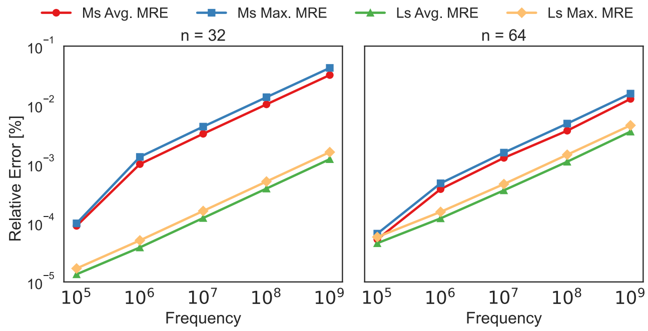


Fig. 4. Relative error of Ms (Maclaurin series) and Ls (Least squares) approximations under varying array sizes and operating frequency.

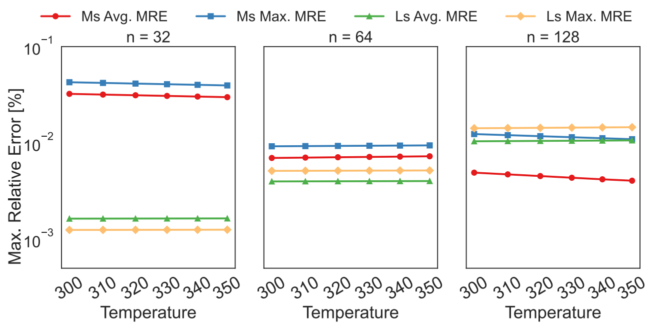


Fig. 5. Relative error of Ms (Maclaurin series) and Ls (Least squares) approximations under varying array sizes and temperatures.

TABLE III
VOLTAGE DROP DISTRIBUTION OF RERAM CELLS ACROSS VARIOUS ARRAY SIZES. THE PERCENTAGE VALUE IS CALCULATED AS THE RATIO OF THE NUMBER OF RERAM WITH VOLTAGES BELOW A CERTAIN THRESHOLD TO THE TOTAL NUMBER OF RERAM CELLS

Array size	$< 0.1V$	$< 0.01V$	$< 0.001V$
256	92.14%	42.69%	16.24%
512	97.16%	85.05%	16.45%
1024	98.78%	93.45%	69.78%
2048	99.43%	97.16%	91.16%

when deploying neural network models onto ReRAM, further hardware deployment design is required. To address this, the proposed real-time compensation methodology integrates software simulation with hardware compensation to effectively mitigate the inherent errors in ReRAM-based computing systems. The detailed architecture is illustrated in Fig. 6. This integrated solution not only enhances computational accuracy but also ensures minimal impact on system throughput, making it suitable for large-scale neural network applications.

Software Simulation: In the software simulation phase, discrepancies between the ideal and actual system behaviors are identified, and the required compensation coefficients are calculated to mitigate these errors effectively. This phase mainly comprises the following steps.

1 *DNN Model Mapping:* The trained DNN model are first mapped to ReRAM system for computation.

2 *Computation of Ideal and Actual Output Currents:* The ideal output current ($\boldsymbol{I}_{\text{ideal}}$) is obtained by simulating the network under ideal conditions without considering device-level imperfections. The actual output current ($\boldsymbol{I}_{\text{actual}}$) is influenced by various hardware-specific nonideal effects, including IR-Drop, SAF, and different noise sources. Our proposed sparse LU factorization method accounts for and calculates these effects, ensuring accurate modeling of the system.

3 *Compensation Coefficient Derivation:* The difference between $\boldsymbol{I}_{\text{ideal}}$ and $\boldsymbol{I}_{\text{actual}}$ is calculated to determine hardware-induced errors. The compensation coefficient $\boldsymbol{C}$ is calculated by (9) and then is used in the hardware compensation phase. We leverage the multicore computing capabilities of the CPU, assigning one core to each RCA to enable parallel processing. This approach can minimize the impact on throughput as much as possible.

Hardware Compensation: The right side of Fig. 6 shows an RCA with the compensation logic, which includes registers for storing the compensation factor and a multiplier. We store the compensation factor (C_1, C_2, ... ,C_n) in a register. Using the compensation logic, we mitigate errors in vector-matrix multiplications to obtain compensated outputs, as calculated in (10). Based on the ReRAM crossbar system specifications summarized in Table IV, we give the detailed circuit overhead and system throughout analysis of the proposed hardware compensation logic as follows.

1) *Circuit Area:* Our calculations indicate that the compensation logic increases the circuit area by approximately 8% at the RCA level and around 4% at the system level.
2) *Power Consumption:* The dynamic power of registers and multipliers dominates compensation logic power. Compared to the conventional ReRAM array circuit, the power consumption increase is less than 7% [38].
3) *Latency:* The critical path includes multiplier delay (1.2ns) and register access (0.5ns), totaling (1.7ns). This is negligible compared to DAC/ADC latency (30-100 μs).
4) *System Throughput:* The slowest stage that determines RCA throughput is the crossbar-connected DAC and ADC. In our evaluation, the latency of the 8-bit ADC and 8-bit DAC operating at 0.1 MHz is between 30–50 μs and 80–100 μs, respectively. Considering the time delay, the proposed method can generally be applied to arrays of sizes 32, 64, and 128, where the real-time compensation process does not significantly affect the throughput. Moreover, integrating multicore parallelism during the software simulation phase ensures efficient processing of large-scale RCAs, further enhancing scalability.

V. EXPERIMENTAL RESULTS

A. Experimental Setup

We categorize the experimental setup into hardware and software components to clarify the toolchain and models employed in this study.

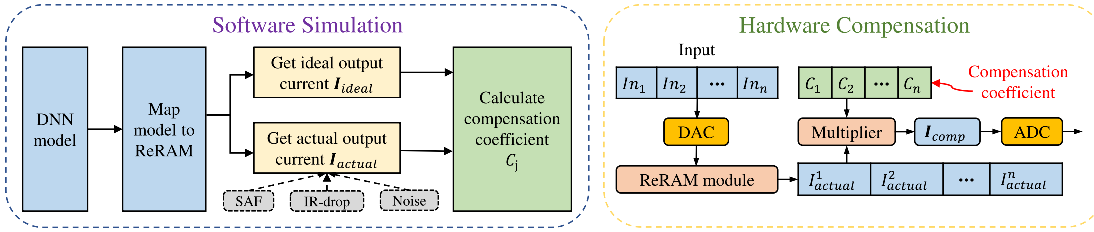


Fig. 6. Entire process of the real-time compensation method combined with software and hardware designs.

TABLE IV
PARAMETERS OF RERAM CROSSBAR SYSTEM [39]

Symbol	Description	Value
G_{min}/G_{max}	ReRAM min/max conductance	100 / 1μS
V_{dd}/V_{ref}	Supply/referencevoltage	0.8/0.4 V
f	Operating frequency	100KHz $\sim$ 1GHz
T	Temperature	300 $\sim$ 350 K
W	Width of ReRAM cell	100 nm
L	Length of ReRAM cell	28 nm
C_{ox}	Oxide capacitance	1×10^{-2} μF/cm^2
K	Flicker noise figure	1×10^{-21}
q	Elementary charge	1.6×10^{-19}C
k_B	Boltzmann constant	1.38×10^{-23} J/K

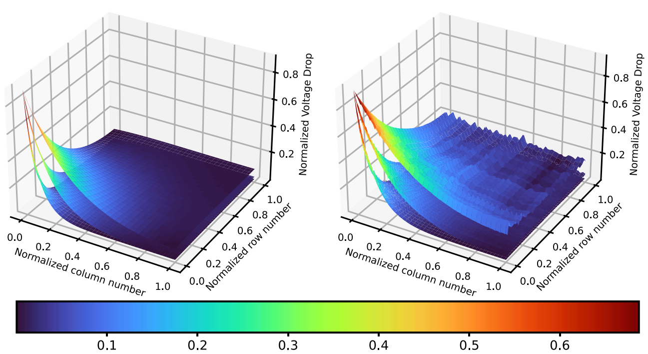


Fig. 7. Distribution of voltage drop of RCAs with different dimensions. Surfaces from top to down are the arrays with sizes of 64 × 64, 128 × 128, and 256 × 256, respectively. Normalized voltage denotes the ratio with and without IR-Drop consideration. Left: worst case, Right: typical case.

Hardware Configuration: In this work, we select moderate resolutions for the ADC, DAC, and ReRAM (i.e., 8-8-7 bits) for easy comparison with other studies. The remaining parameters of the RCAs system are listed in Table IV.

Software Configuration: To comprehensively evaluate the performance of a ReRAM-based neural network accelerator under various nonideal effects, we develop a comprehensive simulation framework based on sparse LU decomposition, implemented in Python and C++, and built on top of PyTorch. The simulations are performed on an NVIDIA RTX 3090 GPU and an Intel Core i7-14700K CPU.

B. Array-Level IR-Drop Effect Analysis

A comprehensive quantitative analysis of the principal physical parameters within RCAs, with a particular emphasis on the impact of IR-Drop, is conducted to assess its influence on performance. As illustrated in Fig. 7, IR-Drop considerably alters the internal voltage of RCAs. The worst-case scenario

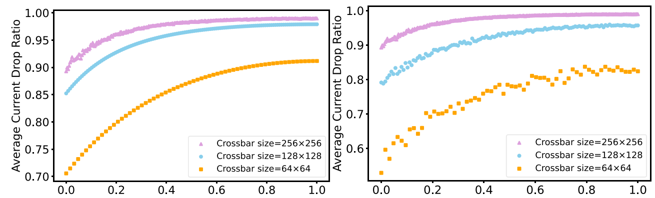


Fig. 8. Average current drop ratio = $(I_{\text{actual}} - I_{\text{ideal}})/I_{\text{ideal}}$. Left: Worst case, Right: Typical case. The horizontal coordinate represents the normalized column coordinate of the ReRAM.

depicted in Fig. 7 assumes that the resistance of every ReRAM cell in the array is set uniformly to 10KΩ. Under typical conditions, the resistance of memory cell in the array randomly varies within the range of 10KΩ and 1MΩ. It can be seen that the normalized value between the node voltage and the ideal voltage at the nth row and the nth column can reach 90%, indicating near-total distortion. In addition, as the size of the RCAs increases, the voltage drops substantially impact the performance of the accelerator. Fig. 8 further presents the relative error between the output current of RCAs considering IR-Drop and the ideal current. The larger the array, the greater the observed relative error.

C. Comparison With SOTA Works

We perform 100 ReRAM array-level simulations, where input voltages and ReRAM conductance parameters are randomly assigned, to evaluate the performance of our proposed approach thoroughly. This extensive testing ensures the robustness and reliability of our method across a wide range of configurations and scenarios. Table V shows our evaluation results on arrays of different sizes. The results demonstrate that the time complexity of our method scales nearly linearly with array size. Considering that RCAs circuit simulation will reuse the result of symbolic factorization multiple times, the actual simulation time accounts only for the numerical factorization. We complete the calculation in just 0.53ms and 0.98ms, respectively, for RCAs with typical dimensions (n = 64, 128). For large-scale array size accelerator design exploration (n = 2048), the numeric factorization requires just 0.25s. We drastically reduce the numerical factorization time to microsecond levels using only a single CPU core, providing an efficient solution for the simulation of ReRAM-based neural network accelerators and the online training of diverse fault-tolerant algorithms.

The resulting MRE distribution is shown in comparison with other methods in Fig. 9, including Au [40], EM [41],

TABLE V
PERFORMANCE EVALUATION OF RCAs WITH VARYING SIZES

Array size	nnz	Symbolic factorization	Numeric factorization	Total time
16	1984	0.00132	0.00032	0.00172
32	8064	0.00469	0.00041	0.00501
64	32512	0.01746	0.00053	0.01799
128	130560	0.07444	0.00098	0.07542
256	523264	0.31443	0.00446	0.31889
512	2095104	1.11956	0.01438	1.13394
1024	8384512	4.99321	0.07105	5.06426
2048	33546240	23.23645	0.25670	23.49315

[1] The benchmark is conducted on a desktop computer equipped with an Intel Core i7-14700K processor and 32 GB of RAM.
[2] In the above experiment, we use a single-core configuration.
[3] Operating system: Windows 11.
[4] nnz is the number of nonzero elements in the array and other performance unit is in seconds.

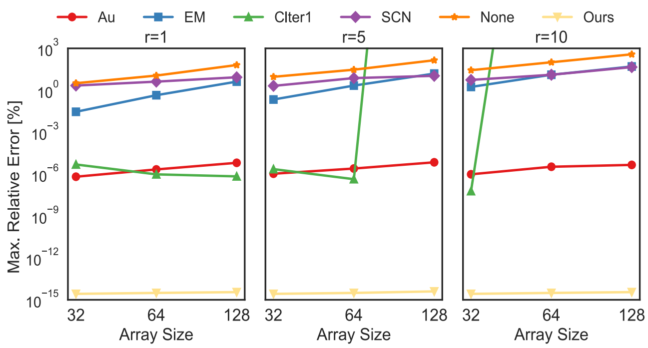


Fig. 9. Average maximum relative error under different array size and wire resistance (r).

CIter1 [10], and SCN [42]. In addition, "None" refers to no optimization for IR-Drop. It can be noticed that most model approximation and iteration methods exhibit high sensitivity to simulation parameters. As array size increases, the MRE shows a gradual upward trend, to the extent that CIter1 fails to converge as wire resistance increases. Compared with EM and other iterative algorithms, the numerical solution accuracy of our proposed method is more than seven orders of magnitude greater. In addition, our proposed method differs from the above methods by guaranteeing high numerical accuracy for large arrays ($n > 1024$) across varying parameters. For example, the MRE of our approach remains on the order of 10^{-15}, even for large-scale arrays with the array size $n = 2048$.

We also illustrate the computation time and memory usage of different methods, as shown in Fig. 10. For SPICE simulation, we adopt two widely used SPICE programs (NGSPICE and HSPICE) to simulate the RCAs. As the array size increases, our method is expected to achieve greater acceleration and reduced resource utilization compared to NGSPICE. In summary, our method achieves a speedup ranging from 54.7 to 660099.2 and a reduction in memory usage from 35.1 to 1882.9 compared with NGSPICE. Similarly, compared to HSPICE, our approach significantly improves simulation time and reduces memory footprint. Our method uses just 5.7% of the memory and 0.08% of the time required by HSPICE to achieve the same numerical accuracy. Compared with the exact

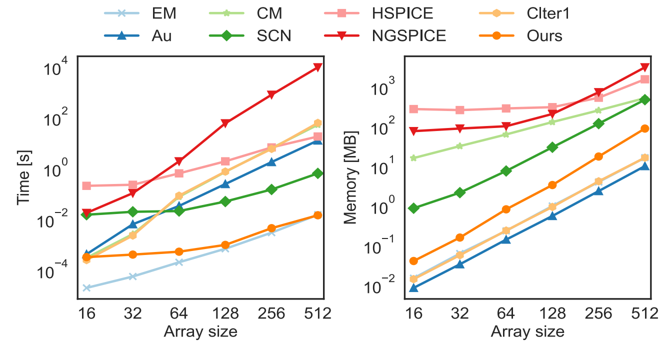


Fig. 10. Left: Computational time (s) used to solve a linear system. Right: Computational memory (MB) used to compute a linear system.

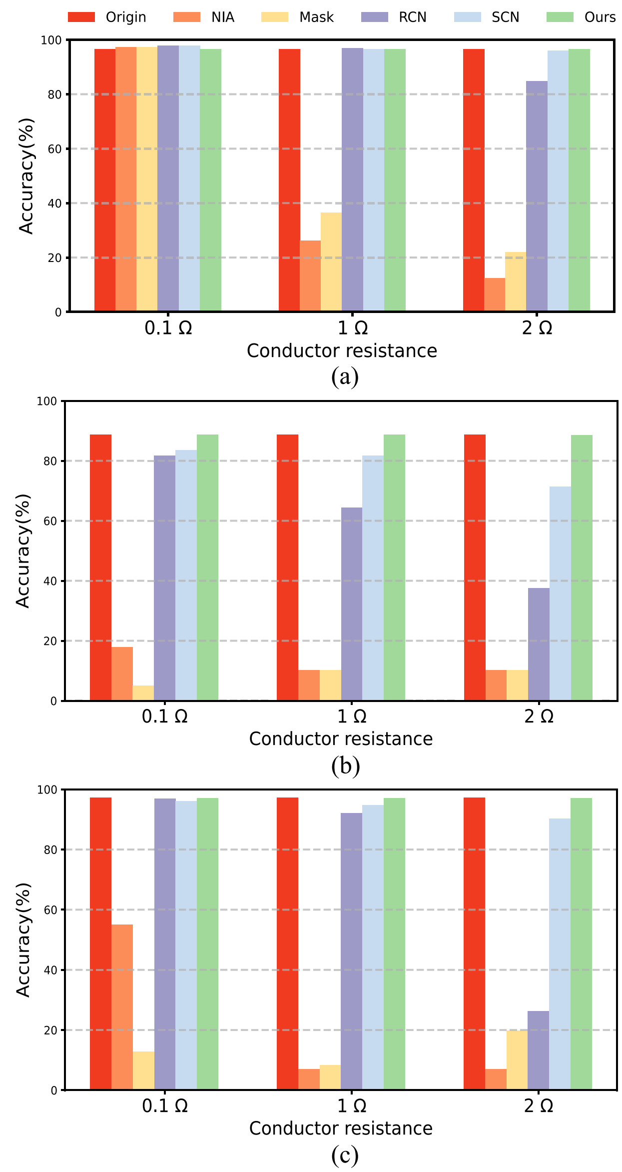


Fig. 11. Accuracy under different R_{wire} parameters. The RCA size is 128 × 128 (a) MNIST dataset (b) CIFAR-10 dataset (c) SVHN dataset.

model CM [43], we exploit the properties of sparse matrices and achieve a speedup of 3623.5 in RCAs simulation while utilizing less than 2% of the memory required by the accurate

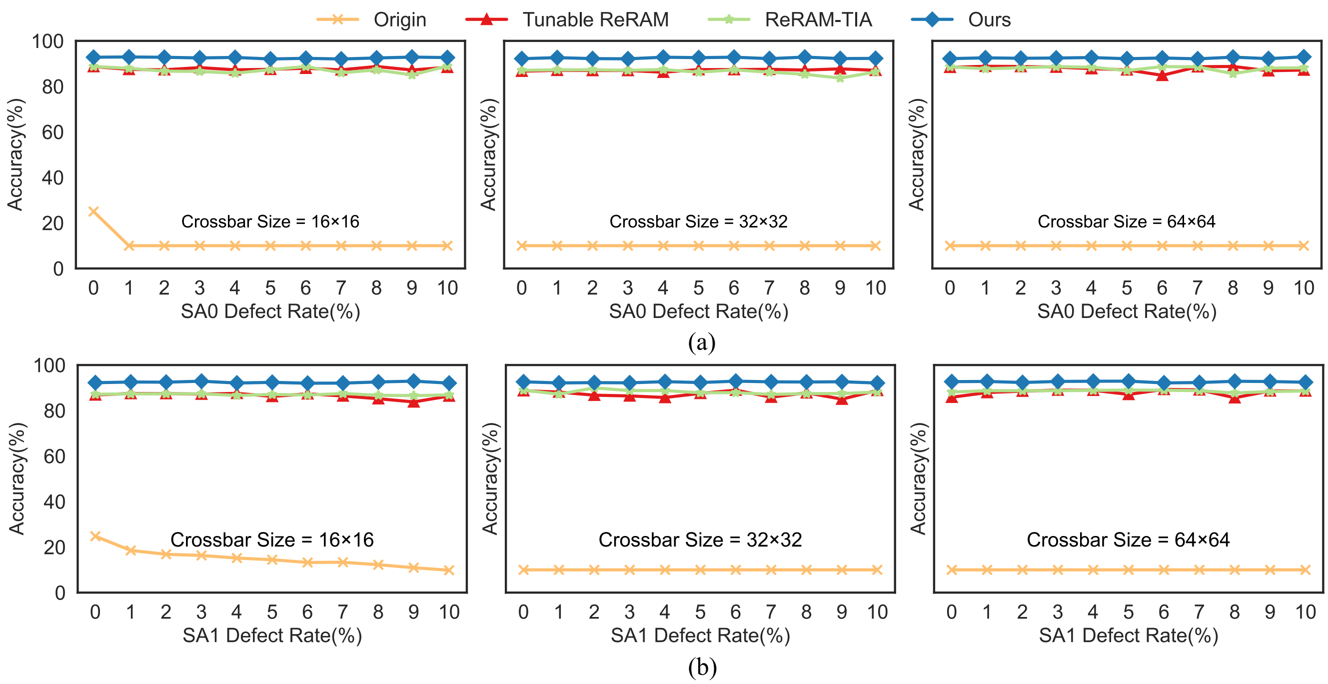


Fig. 12. Simulation results with considering IR-Drop and SA0 or SA1 on different size of RCAs on CIFAR-10 dataset. "Origin" presents the results conducted without applying any optimization method (a) IR-Drop + SA0 defect (b) IR-Drop + SA1 defect.

model for $n = 512$. As shown in the right subfigure of Fig. 10, our memory usage is slightly larger than that of EM, Au, and CIter1. However, we achieve superior simulation speed and accuracy, with our method enhancing simulation speed by an average of 3 to 4 orders of magnitude and achieving an accuracy level of 10^{-15}.

D. Ablation Study

To further verify the effectiveness of the proposed real-time current compensation methodology, we explore the deployment performance of BNN [44] and a convolutional neural network on RCAs. Three datasets MNIST [45], CIFAR-10 [46], and SVHN [45] are performed.

We first compare the proposed methodology with four related approaches on BNN: NIA [6], Mask [47], RCN [42], and SCN [42]. The evaluation is conducted on the array size of 128 × 128. The wire resistance R_{wire} of each unit is varied from 0.1 to 2Ω. Fig. 11 presents the validation accuracy results on all three datasets. "Origin" represents the baseline accuracy on GPU. Our findings reveal the significant impact of IR-Drop in ReRAM-based neural networks. Even at a wire resistance as low as 0.1Ω, MVM computations in a 128 × 128 RCA exhibit distortions so severe that the network becomes unsuitable for BNN applications. Moreover, the proposed method consistently surpasses previous approaches, exhibiting strong robustness even when $R_{\text{wire}} = 2\Omega$ and the array size is 128 × 128. While the improvements in MNIST-BNN are subtle except in particularly challenging cases, they become notably more significant when applied to more complex datasets, such as CIFAR-10 and SVHN. As shown in Fig. 11, our method achieves accuracies 17.17% and 6.88% higher than SCN on the CIFAR-10 and SVHN datasets, respectively, under the most challenging condition ($R_{\text{wire}} = 2\Omega$).

We further conduct a comprehensive assessment of our proposed approach by simultaneously considering the effects of IR-Drop and SAF effects. This dual-consideration facilitates a more rigorous evaluation of our method's performance and robustness under conditions reflecting real-world challenges in electronic systems. The CIFAR-10 dataset is evaluated on the VGG16 model, with SA0 and SA1 percentages varied from 0 to 10%, respectively. As shown in Fig. 12, the accuracy of baseline decreases significantly as the SA0/SA1 ratio increases, but extremely high accuracy can still be maintained using our method. Compared to Tunable ReRAM and RERAM-TIA methods, our method has an average accuracy improvement of 4%.

Although our proposed method does not significantly reduce IR-Drop and SAF effects compared to tunable ReRAM and ReRAM-TIA [8], it exhibits superior stability across various SAF ratios. The tunable ReRAM and ReRAM-TIA approaches require recalculating the mean and variance of the compensated current for each new task and rewriting weights for new rows. This process significantly increases circuit area, thereby limiting the method's applicability. In contrast, our framework offers a stable and adaptable solution with fewer constraints, making it applicable to a broader range of tasks.

The 7-bit ReRAM used in this work is constrained by the current fabrication process. To assess the impact of this limitation, we conduct random tests on 25 groups using both 7-bit and 8-bit ReRAM across different array sizes. As shown in Fig. 13, as the array size increases, both absolute and relative errors decrease significantly, indicating that the effect of mismatched quantization bits becomes less pronounced in large-scale RCAs.

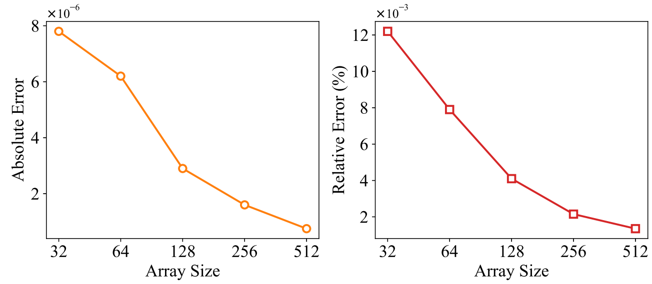


Fig. 13. 7-bit and 8-bit ReRAM comparison: absolute and relative errors of output current across different array sizes (n). Absolute Error Difference $= (1/n)\sum_{i=1}^{n}|I^i_{7\mathrm{bit}} - I^i_{8\mathrm{bit}}|$; Relative Error Difference $= (1/n)\sum_{i=1}^{n}|I^i_{7\mathrm{bit}} - I^i_{8\mathrm{bit}}|/I^i_{7\mathrm{bit}} \times 100\%$.

VI. Conclusion and Discussion

In this work, we introduce a hardware-software methodology that incorporates a sparse LU factorization algorithm and a low-overhead hardware compensation scheme, aimed at achieving high-accuracy DNNs on resistive crossbars. The software algorithm addresses the challenges of simulating large-scale RCAs during online training and inference, significantly accelerating the speed of RCAs simulation while reducing hardware resource costs. In addition, we propose an approximate algorithm for nonlinear noise simulation, which converts nonlinear problems into linear problems, and greatly accelerates the solving speed of large-scale arrays while ensuring accuracy. Based on the above methods, we propose a real-time current hardware compensation method, which addresses the need for traditional approaches to modify compensation factors for different task scenarios, ensuring that DNN inference precision remains largely unaffected.

Our proposed sparse LU factorization method is applicable to a broad range of ReRAM technologies, including Al-doped Hafnium Oxide (HfO_2) [48] and Copper-doped $Pt/KNbO_3/KN/TiN/Si$ [49]. As long as these devices operate within a voltage range that ensures sufficiently linear I-V behavior, our approach remains valid. However, for ReRAM devices with strongly nonlinear characteristics (e.g., $TiN/TiO_2/Ti_2/Pt$ [50] and $Pt/TaO_x/TiO_{2-x}/P$ [51] ReRAM), the linear approximations used in our method may reduce accuracy, especially in large-scale arrays. Extending the model to capture nonlinearity is an interesting direction for future work. Besides, the proposed framework does not incorporate the imperfections of DAC/ADC components, which might introduce additional computational errors in hardware implementations. Future research should systematically integrate these factors to enhance the model's generality across diverse scenarios.

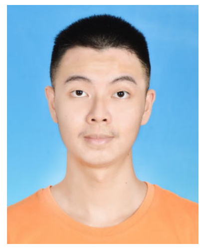
References

[1] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *Proc. HPCA*, 2017, pp. 541–552.
[2] H. Akinaga and H. Shima, "Resistive random access memory (ReRAM) based on metal oxides," *Proc. IEEE*, vol. 98, no. 12, pp. 2237–2251, Dec. 2010.
[3] K. Smagulova, M. E. Fouda, F. Kurdahi, K. N. Salama, and A. Eltawil, "Resistive neural hardware accelerators," *Proc. IEEE*, vol. 111, no. 5, pp. 500–527, May 2023.
[4] C. Quan et al., "Training-free stuck-at fault mitigation for ReRAM-based deep learning accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 7, pp. 2174–2186, Jul. 2023.
[5] S. Lee, M. E. Fouda, J. Lee, A. M. Eltawil, and F. Kurdahi, "Offline training-based mitigation of IR drop for ReRAM-based deep neural network accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 2, pp. 521–532, Feb. 2023.
[6] Z. He, J. Lin, R. Ewetz, J.-S. Yuan, and D. Fan, "Noise injection adaption: End-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping," in *Proc. 56th ACM/IEEE DAC*, 2019, pp. 1–6.
[7] S. Jain and A. Raghunathan, "CxDNN: Hardware-software compensation methods for deep neural networks on resistive crossbar systems," *ACM Trans. Embedd. Comput. Syst.*, vol. 18, no. 6, pp. 1–23, 2019.
[8] C. Huang, N. Xu, K. Qiu, Y. Zhu, D. Ma, and L. Fang, "Efficient and optimized methods for alleviating the impacts of IR-drop and fault in RRAM based neural computing systems," *IEEE J. Electron Devices Soc.*, vol. 9, pp. 645–652, 2021.
[9] D. Soudry, D. Di Castro, A. Gal, A. Kolodny, and S. Kvatinsky, "Memristor-based multilayer neural networks with online gradient descent training," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 10, pp. 2408–2421, Oct. 2015.
[10] Y. Luo, S. Wang, P. Zuo, Z. Sun, and R. Huang, "Modeling and mitigating the interconnect resistance issue in analog RRAM matrix computing circuits," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 11, pp. 4367–4380, Nov. 2022.
[11] N. Lepri, M. Baldo, P. Mannocci, A. Glukhov, V. Milo, and D. Ielmini, "Modeling and compensation of IR drop in crosspoint accelerators of neural networks," *IEEE Trans. Electron Devices*, vol. 69, no. 3, pp. 1575–1581, Mar. 2022.
[12] F. Zhang and M. Hu, "CCCS: Customized SPICE-level crossbar-array circuit simulator for in-memory computing," in *Proc. IEEE/ACM ICCAD*, 2020, pp. 1–8.
[13] S. Gao, F. Yang, C. Lu, and Y. Zhao, "Fast algorithms for exact IR drop de-embedding in analog multiply–accumulate computing," *IEEE Trans. Electron Devices*, vol. 69, no. 11, pp. 6376–6383, Nov. 2022.
[14] L. Xia et al., "MNSIM: Simulation platform for memristor-based neuromorphic computing system," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 5, pp. 1009–1022, May 2018.
[15] Z. Zhu et al., "MNSIM 2.0: A behavior-level modeling tool for memristor-based neuromorphic computing systems," in *Proc. GLSVLSI*, 2020, pp. 83–88.
[16] P.-Y. Chen, X. Peng, and S. Yu, "NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 12, pp. 3067–3080, Dec. 2018.
[17] X. Peng, S. Huang, H. Jiang, A. Lu, and S. Yu, "DNN+NeuroSim V2.0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 11, pp. 2306–2319, Nov. 2021.
[18] T. A. Davis and I. S. Duff, "An unsymmetric-pattern multifrontal method for sparse LU factorization," *SIAM J. Matrix Anal. Appl.*, vol. 18, no. 1, pp. 140–158, 1997.
[19] O. Schenk and K. Gärtner, "Solving unsymmetric sparse systems of linear equations with PARDISO," *Future Gener. Comput. Syst.*, vol. 20, no. 3, pp. 475–487, 2004.
[20] A. Chen, "A comprehensive crossbar array model with solutions for line resistance and nonlinear device characteristics," *IEEE Trans. Electron Devices*, vol. 60, no. 4, pp. 1318–1326, Apr. 2013.
[21] L. Xia et al., "Stuck-at fault tolerance in RRAM computing systems," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 1, pp. 102–115, Mar. 2018.
[22] E. I. Vatajelu, P. Prinetto, M. Taouil, and S. Hamdioui, "Challenges and solutions in emerging memory testing," *IEEE Trans. Emerg. Topics Comput.*, vol. 7, no. 3, pp. 493–506, Jul.–Sep. 2019.
[23] Y.-X. Chen and J.-F. Li, "Fault modeling and testing of 1T1R memristor memories," in *Proc. IEEE 33rd VLSI*, 2015, pp. 1–6.
[24] Z. Wei et al., "True random number generator using current difference based on a fractional stochastic model in 40-nm embedded ReRAM," in *Proc. IEEE IEDM*, 2016, pp. 4.8.1–4.8.4.
[25] W. Feng, H. Shima, K. Ohmori, and H. Akinaga, "Investigation of switching mechanism in HfOx-ReRAM under low power and conventional operation modes," *Sci. Rep.*, vol. 6, no. 1, 2016, Art. no. 39510.
[26] X. Yang et al., "Multi-objective optimization of ReRAM crossbars for robust DNN inferencing under stochastic noise," in *Proc. IEEE/ACM ICCAD*, 2021, pp. 1–9.

[27] R. Schreiber, "A new implementation of sparse Gaussian elimination," *ACM Trans. Math. Softw.*, vol. 8, no. 3, pp. 256–276, 1982.
[28] J. R. Gilbert and R. Schreiber, "Highly parallel sparse Cholesky factorization," *SIAM J. Sci. Stat. Comput.*, vol. 13, no. 5, pp. 1151–1172, 1992.
[29] N. J. Higham, "Cholesky factorization," *Wiley Interdiscip. Rev., Comput. Stati.*, vol. 1, no. 2, pp. 251–254, 2009.
[30] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, "Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 377–380, 2004.
[31] P. R. Amestoy, T. A. Davis, and I. S. Duff, "Algorithm 837: AMD, an approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 381–388, 2004.
[32] W.-H. Liu and A. H. Sherman, "Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices," *SIAM J. Numer. Anal.*, vol. 13, no. 2, pp. 198–213, 1976.
[33] R. J. Lipton, D. J. Rose, and R. E. Tarjan, "Generalized nested dissection," *SIAM J. Numer. Anal.*, vol. 16, no. 2, pp. 346–358, 1979.
[34] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1998.
[35] T. Wang, W. Li, H. Pei, Y. Sun, Z. Jin, and W. Liu, "Accelerating sparse LU factorization with density-aware adaptive matrix multiplication for circuit simulation," in *Proc. 60th ACM/IEEE DAC*, 2023, pp. 1–6.
[36] L. S. Blackford et al., "An updated set of basic linear algebra subprograms (BLAS)," *ACM Trans. Math. Softw.*, vol. 28, no. 2, pp. 135–151, 2002.
[37] Z. Wen, W. Yin, and Y. Zhang, "Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm," *Math. Program. Comput.*, vol. 4, no. 4, pp. 333–361, 2012.
[38] M. Mao, P.-Y. Chen, S. Yu, and C. Chakrabarti, "A multilayer approach to designing energy-efficient and reliable ReRAM cross-point array system," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 5, pp. 1611–1621, May 2017.
[39] C.-X. Xue et al., "16.1 A 22nm 4mb 8b-precision ReRAM computing-in-memory macro with 11.91 to 195.7TOPS/W for tiny AI edge devices," in *Proc. IEEE ISSCC*, 2021, pp. 245–247.
[40] B. Wu et al., "FADESIM: Enable fast and accurate design exploration for memristive accelerators considering nonidealities," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 44, no. 4, pp. 1529–1543, Apr. 2025.
[41] R. Han, P. Huang, Y. Zhao, X. Cui, X. Liu, and J. Kang, "Efficient evaluation model including interconnect resistance effect for large scale RRAM crossbar array matrix computing," *Sci. China Inf. Sci.*, vol. 62, pp. 1–11, Feb. 2019.
[42] S. Lee, M. E. Fouda, J. Lee, A. Eltawil, and F. Kurdahi, "Accurate prediction of ReRAM crossbar performance under I-V nonlinearity and IR drop," in *Proc. IEEE 40th ICCD*, 2022, pp. 9–16.
[43] M. E. Fouda, A. M. Eltawil, and F. Kurdahi, "Modeling and analysis of passive switching crossbar arrays," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 1, pp. 270–282, Jan. 2018.
[44] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. 30th NIPS*, 2016, pp. 4114–4122.
[45] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet, "Multi-digit number recognition from street view imagery using deep convolutional neural networks," 2014, *arXiv:1312.6082*.
[46] M. Liang and X. Hu, "Recurrent convolutional neural network for object recognition," in *Proc. IEEE CVPR*, 2015, pp. 3367–3375.
[47] M. E. Fouda, S. Lee, J. Lee, A. Eltawil, and F. Kurdahi, "Mask technique for fast and efficient training of binary resistive crossbar arrays," *IEEE Trans. Nanotechnol.*, vol. 18, pp. 704–716, Jul. 2019.
[48] S. Chandrasekaran, F. M. Simanjuntak, R. Saminathan, D. Panda, and T.-Y. Tseng, "Improving linearity by introducing Al in HfO_2 as a memristor synapse device," *Nanotechnology*, vol. 30, no. 44, 2019, Art. no. 445205.
[49] J. H. Yoon, Y.-W. Song, W. Ham, J.-M. Park, and J.-Y. Kwon, "A review on device requirements of resistive random access memory (RRAM)-based neuromorphic computing," *APL Mater.*, vol. 11, no. 9, 2023, Art. no. 90701.
[50] F. Lentz, B. Roesgen, V. Rana, D. J. Wouters, and R. Waser, "Current compliance-dependent nonlinearity in TiO_2 ReRAM," *IEEE Electron Device Lett.*, vol. 34, no. 8, pp. 996–998, Aug. 2013.
[51] J. J. Yang et al., "Engineering nonlinearity into memristors for passive crossbar applications," *Appl. Phys. Lett.*, vol. 100, no. 11, 2012, Art. no. 113501.

Weiran Chen (Graduate Student Member, IEEE) received the undergraduate degree from the Hefei University of Technology, Hefei, China, in 2023. He is currently pursuing the master's degree with the School of Microelectronics, University of Science and Technology of China, Hefei.

His research interests include neuromorphic computing, large-scale integrated circuit simulation, and computer vision.



Zaitian Chen received the undergraduate degree in applied physics from the University of Science and Technology of China, Hefei, China, in 2024, where he is currently pursuing the master's degree with the School of Microelectronics.

His research interests include neuromorphic computing and multiphysics analysis.

Bei Yu (Senior Member, IEEE) received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Associate Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

Dr. Yu received the eleven Best Paper Awards from ICCAD 2024, 2021, and 2013, the IEEE TSM 2022, the DATE 2022, the ASPDAC 2021 and 2012, the ICTAI 2019, the Integration, the VLSI Journal in 2018, the ISPD 2017, the SPIE Advanced Lithography Conference 2016, the six ICCAD/ISPD contest awards, the IEEE CEDA Ernest S. Kuh Early Career Award in 2021, DAC the Under-40 Innovator Award in 2024, and the Hong Kong RGC Research Fellowship Scheme Award in 2024.

Song Chen (Member, IEEE) received the Ph.D. degree in computer science from Tsinghua University, Beijing, China, in 2005.

He is currently an Associate Professor with the School of Microelectronics, University of Science and Technology of China, Hefei, China. His research interests include several aspects of VLSI design automation, on-chip communication system, in-memory computing, and computer-aided design for emerging technologies.

Dr. Chen is a member of ACM, and IEICE.

Yi Kang (Member, IEEE) received the Ph.D. degree in computer science from University of Illinois at Urbana–Champaign, Champaign, IL, USA, in 1999.

He is currently a Professor with the School of Microelectronics, University of Science and Technology of China (USTC), Hefei, China. Before went to USTC, he worked as Chief Scientist and SVP with Spreadtrum Communications Inc. in Shanghai, China. His current research area includes new computing and memory architecture and implementation of neural networks.

Qi Xu (Member, IEEE) received the Ph.D. degree in electronic science and technology from the University of Science and Technology of China, Hefei, China, in 2018. He is currently an Associate Professor in the School of Microelectronics, USTC. His research interests include physical design automation and machine learning in EDA.