RESEARCH ARTICLE | JUNE 18 2025

Simulation and benchmarking of crossbar parasitic resistance models: Accuracy and performance comparison

Alessandro Lambertini ; Tommaso Zanotti ; Paolo Pavan ; Andrea Padovani ; Francesco Maria Puglisi

APL Mach. Learn. 3, 026116 (2025)
https://doi.org/10.1063/5.0274233

Simulation and benchmarking of crossbar parasitic resistance models: Accuracy and performance comparison

Cite as: APL Mach. Learn. 3, 026116 (2025); doi: 10.1063/5.0274233
Submitted: 4 April 2025 • Accepted: 9 June 2025 •
Published Online: 18 June 2025

Alessandro Lambertini,[1] Tommaso Zanotti,[1,a)] Paolo Pavan,[1] Andrea Padovani,[2] and Francesco Maria Puglisi[1,a)]

AFFILIATIONS

[1] Dipartimento di Ingegneria "Enzo Ferrari," Universitá di Modena e Reggio Emilia, Via P. Vivarelli, 10/1, 41125 Modena (MO), Italy
[2] Dipartimento di Scienze e Metodi dell'Ingegneria, Universitá di Modena e Reggio Emilia, Via Amendola 2, 42122 Reggio Emilia (RE), Italy

[a)] **Authors to whom correspondence should be addressed:** tommaso.zanotti@unimore.it and francescomaria.puglisi@unimore.it

ABSTRACT

Resistive crossbar arrays have been shown to enable the implementation of energy-efficient in-memory computing accelerators suitable for the diffusion of artificial neural networks (ANNs) at the edge. However, the effect of line parasitic resistances can degrade the performance of ANNs if not appropriately considered during the design of the accelerator and each time a new task is targeted. Poised by these limitations, several crossbar line parasitic resistance models have been proposed in the literature. However, a comparative study of the performance of these models is still missing. In this work, we compare and benchmark over a broad range of operating conditions several crossbar line parasitic resistance models from the literature. The results emphasize a practical trade-off: compact analytical models are fast but provide reliable estimates only for small (i.e., 32 × 32) array sizes; conversely, iterative numerical models require more computing time but are more accurate under all conditions. Thus, iterative methods remain indispensable for larger or more complex designs. In addition, the implications of using these models in line parasitic-aware training of the ANN are analyzed on an MNIST classification task. The results further indicate that despite providing a considerable speedup of the ANN training, analytical models preserve accuracy only when small crossbar tiles are used, highlighting the need for more dependable yet fast compact models.

I. INTRODUCTION

Artificial intelligence (AI) technology is gaining increasing popularity and is influencing nearly every aspect of modern life. In fact, its range of applications is vast, from deep neural networks (DNNs) running on edge devices to large language models (LLMs) running within cloud clusters. The exponential growth rate of these technologies is raising questions regarding their sustainability,[1–3] indicating the need for more energy-efficient hardware solutions. The main culprit for energy inefficiency is currently represented by the separation of the memory and processing resources that leads to continuous and highly inefficient data transfer.[4,5] As the technology progresses, the model size increases, thus worsening this efficiency bottleneck. Moreover, these issues are further exacerbated when considering edge computing applications, which typically have strict constraints on energy and computing resources.[6] Promising solutions to these problems include the development of in-memory computing hardware accelerators[7–10] and unconventional computing approaches, such as neuromorphic systems.[11,12] These approaches are enabled by emerging nonvolatile memory technologies, which encompass a class of 2- and 3-terminal nonvolatile memory devices such as resistive random access memory (RRAM),[13] phase change memory (PCM),[14] ferroelectric tunnel junction (FTJ),[14] ferroelectric field-effect transistor (FeFET),[15] and magnetic tunnel junction (MTJ)[14] devices. In particular, these nonvolatile memory devices are commonly assembled to form 2D matrix-like structures, i.e., crossbar arrays (see Fig. 1), which can be effectively used to accelerate in hardware the computation of vector–matrix multiplication (VMM) operations, providing considerable performance improvements[16] and enhancing energy efficiency by several orders of magnitude compared to classical architectures.[17] Such VMM accelerators are essential to reduce the

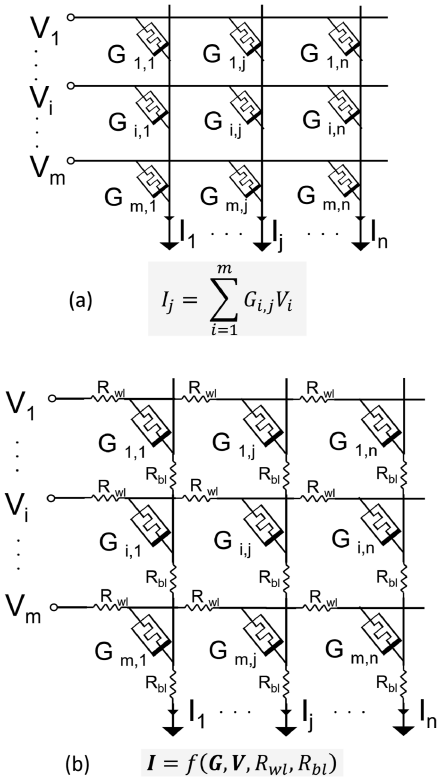


FIG. 1. (a) Ideal crossbar array. The BLs output currents add up linearly based on each cell's conductance (i.e., G_{ij}) and the applied voltages (V_i). (b) Crossbar array including line parasitic resistances, which result in a nonlinear relation between the inputs, the device conductances, and BLs output currents.

energy footprint of current AI technologies, as the VMM operation is extensively used in artificial neural network (ANN) applications.[18] The weights of each layer of an ANN are encoded and programmed into the nonvolatile resistance of the memory devices of a crossbar array. By encoding the layer's input activations to analog voltages applied to the wordlines (WLs) of the crossbar, the result of the VMM is computed in a single step as the current flowing in each bitline (BL) of the crossbar, thanks to Ohm's and Kirchhoff's laws.[7–9,16]

However, several challenges exist for the advancement and widespread adoption of this technology, with the parasitic resistances of the line interconnections in crossbar arrays being particularly critical.[19–21] In fact, as the technology scales, the cross section of the interconnections reduces, leading to significant line resistance contributions between elements in the crossbar. These parasitic resistance elements cause the IR-drop (i.e., voltage drop due to resistance) along the WLs and BLs of the crossbar, resulting in deviations from the ideal voltage drop across each memory device. Such deviations, which increase with larger crossbars, cause unwanted perturbations in the output currents, leading to a considerable drop in the accuracy of the accelerator.[19–23] Although restricting the size of the crossbar would limit the impact of line parasitic resistances, it would considerably reduce the effectiveness of these VMM accelerators due to the substantial hardware overheads introduced by DACs and ADCs in the periphery of the array. A more promising approach is the implementation of hardware-aware mapping, adaptation, or retraining of the neural network weights and/or biases.[19,24,25] However, this requires solving the complex resistive network shown in Fig. 1(b). In particular, the estimation of output currents is non-trivial because they are formulated as a nonlinear function of the device conductances, input potentials, and parasitic resistances, making their accurate computation challenging. While SPICE circuit simulations of the crossbar can reliably model the effect of line parasitic resistances, their use is not suitable for the implementation of these approaches due to the excessive computing time required. Thus, the development of compact models accurately capturing the effect of line parasitic resistances, at a limited computational cost, is essential for the effective implementation of hardware-aware strategies. Indeed, several crossbar parasitic resistance compact models have been developed.[21–23,26,27] These include analytical,[21–23] iterative[26] compact models and Kirchhoff's equation solvers.[27] However, a thorough comparative study between such models, identifying their strengths and limitations, has not yet been conducted.

In this work, we analyze and compare, in terms of accuracy, performance, and robustness, six crossbar parasitic resistance compact models from the literature. The analysis is performed over a broad range of operating conditions by varying several circuit parameters (e.g., size of the crossbar, resistance levels, and memory window). The results of the analysis indicate a clear trade-off between computing time and accuracy, providing help in the compact model selection depending on the simulated conditions and application. Furthermore, we benchmark the models in a parasitic-aware training of a simple DNN performing an MNIST[28] classification task. Although all the compact models, through the parasitic-aware training, can compensate for most of the accuracy loss due to parasitics in small array sizes, the results indicate clear limitations of analytical compact models when dealing with larger crossbars. In contrast, iterative approaches are more robust for larger crossbars but come with increased computational costs. This highlights the need for more robust and efficient compact models.

This paper is organized as follows: Sec. II reviews the state-of-the-art of crossbar parasitic resistance models. Section III presents the simulation setup, the adopted benchmarking metrics, and comparison results of the different compact models. Section IV discusses the implementation and results of the comparison between the models when adopted in the parasitic-aware training of a simple NN classifier. This is followed by conclusions in Sec V.

II. CROSSBAR PARASITIC RESISTANCE MODELS

Different modeling approaches for reproducing the effect of crossbar parasitic resistances have been presented in the literature. These models can be divided into three main categories: analytical models, iterative models, and Kirchhoff's equation solvers. The former are constructed as a set of equations that can be solved analytically, enabling an estimate of the voltages across each element of the crossbar and of the BLs output currents. However, analytical models typically employ approximations for faster computation at the cost of lower accuracy under certain conditions. On the other hand, iterative models iteratively minimize the error in the prediction of node voltages until it converges to a value below a defined target.

Thus, such models can achieve higher accuracy under all conditions at the cost of longer simulation times. Finally, Kirchhoff's equation solvers include standard SPICE simulators, which can provide a reliable and well-established solution by solving the full circuit network. Although computationally intensive, SPICE simulators remain a benchmark for accurate modeling of crossbar parasitic resistances. In addition, solutions that employ optimized numerical methods to speed up the computation of the solution of the whole resistive networks have also been proposed in the literature.[27,29]

The following sub-sections discuss the key characteristics of the different crossbar parasitic resistance models, considering as a reference the $m \times n$ crossbar array shown in Fig. 1(b), with m input voltages and n output currents. In addition, in the discussion, parasitic resistances are assumed uniform for each segment of the WLs and BLs, between adjacent memory devices, exhibiting resistances R_{wl} and R_{bl}, respectively.

A. Analytical compact models

In this subsection, three analytical models are examined in the order of their publication date.

1. Jeong's model

Jeong's model[22] is the less computationally demanding among the three models. To compute the output current at each BL, a different weight is assigned to each parasitic resistance element based on its position within the crossbar array. In particular, where the currents are high, e.g., near the inputs, the voltage drop across the parasitic resistance element will be higher. Thus, a higher weight is assigned to the parasitic resistance elements in that position. This allows the computation of two equivalent parasitic resistors (one for the BL and one for the WL), and the currents can subsequently be derived by solving a simple resistive network [see Fig. 2(c)]. The derivation of this compact model is based on two assumptions: (i) all inputs share the same voltage (i.e., V_a in Fig. 2) and (ii) every device in the crossbar has the same resistance R_{avg}. In particular, R_{avg} is a resistance value in between the maximum and minimum resistance of devices in the array that can be empirically estimated with Eqs. (1) and (2), where p depends on several crossbar parameters, e.g., programmed patterns and memory window. Typically, R_{avg} is considerably higher than the parasitic resistance of the interconnections; thus, currents flowing through the memory devices are approximated to be all equal to $I_{avg} = V_a/R_{avg}$,

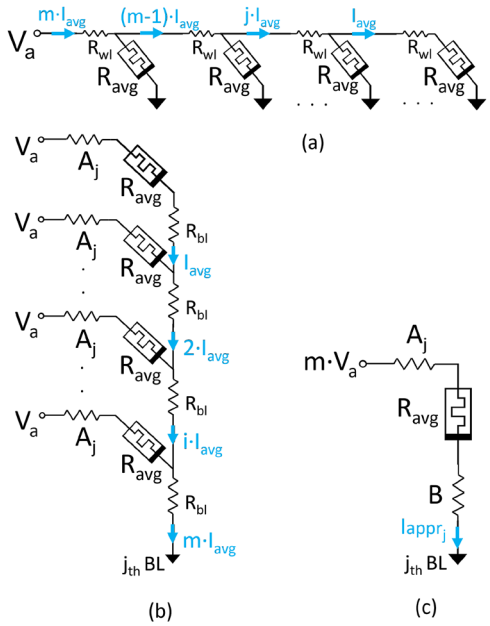


FIG. 2. Sketch of Jeong's model approximations. Panels (a) and (b): WL and BL approximations used to estimate the parameters A_j and B, respectively. (c) Approximate circuit used to estimate the output current (I_{appr_j}) from the jth BL, adapted from Ref. 22.

$$R_{\text{avg}} = \frac{a\,R_{\min} + b\,R_{\max}}{a+b}, \tag{1}$$

$$a = (R_{\min})^{-p}, \quad b = (R_{\max})^{-p}. \tag{2}$$

As sketched in Fig. 2(a), with these approximations, it is easy to see that currents on the WLs accumulate, increasing in value when approaching the input terminals. In particular, integer multiples of I_{avg} build up, from right to left, at each segment of the WLs, as shown in Fig. 2(a). An equivalent parasitic resistance A_j is computed for the j_{th} BL using Eq. (3). Similarly, an equivalent resistor B can be computed for the BLs with Eq. (4). In this case, only a single value of B is needed to estimate the output currents [i.e., the current flowing in the BL after the last row m in Fig. 2(b)],

$$A_j = \sum_{k=1}^{n} (j+1-k)\,R_{\text{wl}} \quad (\text{for the jth BL}), \tag{3}$$

$$B = \sum_{k=1}^{m} k\,R_{\text{bl}}. \tag{4}$$

The approximated average output current at each j BL can be computed with Eq. (5), which considers the circuit shown in Fig. 2(c), where a bias voltage of $m \cdot V_a$ is applied to the series of R_{avg} and the two equivalent parasitic resistors (i.e., A_j and B),

$$I_{\text{appr,avg}_j} = m\,\frac{V_a}{A_j + R_{\text{avg}} + B}. \tag{5}$$

Eventually, this current can be compared to the ideal average output current [i.e., the output current from the ideal crossbar; see Fig. 1(a)] given by the following equation:

$$I_{\text{ideal,avg}} = \frac{mV_a}{R_{\text{avg}}}. \tag{6}$$

The ratio between $I_{\text{ideal,avg}}$ and $I_{\text{appr,avg}_j}$ Eq. (7) represents a scaling factor for the jth BL that can be used to compensate for the effect of parasitics when computing VMM operations. In particular, the scaling factor multiplied by the ideal currents results in an estimate of the output currents including the effect of parasitic resistances,

$$\textit{scaling factor}_j = \frac{I_{\text{ideal,avg}}}{I_{\text{appr,avg}_j}} = \frac{A_j + R_{\text{avg}} + B}{R_{\text{avg}}}. \tag{7}$$

The key advantage of this model is its simplicity and broad applicability. In fact, its use only requires knowing a few technology-related parameters, i.e., the value of parasitic resistances, the array size, and the range of programmed resistances (i.e., $R_{\min}$ and $R_{\max}$). In essence, it is unnecessary to know the exact numerical values associated with the matrix of programmed devices.

2. Diagonal matrix regression model

The diagonal matrix regression (DRM) model is another analytical model in which two diagonal matrices (i.e., A and B) are computed and used to perform a linear transformation on the crossbar conductance matrix (i.e., W) to account for the effect of parasitic resistances. The applicability of the model requires g_{bl} and g_{wl} to be larger than the average conductance in each of the BLs and WLs, respectively. Matrix A is extracted considering the approximated circuit shown in Fig. 3(a), in which $g_{wl} = 0$ and $G_{i,avg}$ is the average conductance in the ith row. Kirchhoff's matrix formulation of the circuit is reported in Eq. (8). This system of equations is solved to estimate the elements a_i of the diagonal matrix, as reported in Eqs. (9) and (10),

$$\begin{bmatrix} G_1 V_1 \\ G_i V_i \\ G_m V_m \end{bmatrix} = \begin{bmatrix} G_1 + g_{bl} & -g_{bl} & 0 \\ -g_{bl} & G_i + 2g_{bl} & -g_{bl} \\ 0 & -g_{bl} & G_m + 2g_{bl} \end{bmatrix} \begin{bmatrix} x_1 \\ x_i \\ x_m \end{bmatrix}, \tag{8}$$

$$a_i = \frac{V_i - x_i}{V_i}, \tag{9}$$

$$a_i \approx \frac{1 + R_{bl} \sum_{k=1}^{i} (i-k)\, G_k}{1 + R_{bl} \sum_{k=1}^{m} (m+1-k)\, G_k}, \tag{10}$$

$$A = \mathrm{diag}\,(a_1,\ a_2,\ \ldots,\ a_n). \tag{11}$$

A similar analysis can be formulated for the WLs; this time considering $R_{bl} = 0$ and taking averages of device conductances along the columns. The approximated circuit is shown in Fig. 3(b). As before, Kirchhoff's equations can be formulated in matrix form, from which a set of coefficients b_j and the corresponding diagonal matrix B can be derived, as reported in Eq. (12),

$$b_j \approx \frac{1 + R_{wl} \sum_{k=j}^{n} (k-j)\, G_k}{1 + R_{wl} \sum_{k=1}^{n} k\, G_k}, \tag{12}$$

$$B = \mathrm{diag}\,(b_1,\ b_2,\ \ldots,\ b_n). \tag{13}$$

The effective conductance matrix W, which accounts for the effect of line parasitic resistances, is then computed using Eq. (14). Finally, W can be used to estimate the output currents from each BL, as described by Eq. (15), resulting in a time complexity of $O(mn)$,

$$W = (A \times G \times B), \tag{14}$$

$$I_{\text{out}} = V_{\text{in}} \times W. \tag{15}$$

3. αβ-matrix model

The $\alpha\beta$-matrix model[23] is similar to the DMR model, however, with some key differences. In this compact model, two coefficient matrices, α and β, multiply the crossbar conductance matrix G to approximately account for the effect of parasitic resistances. Compared to the DMR model, the devices' conductance is not averaged

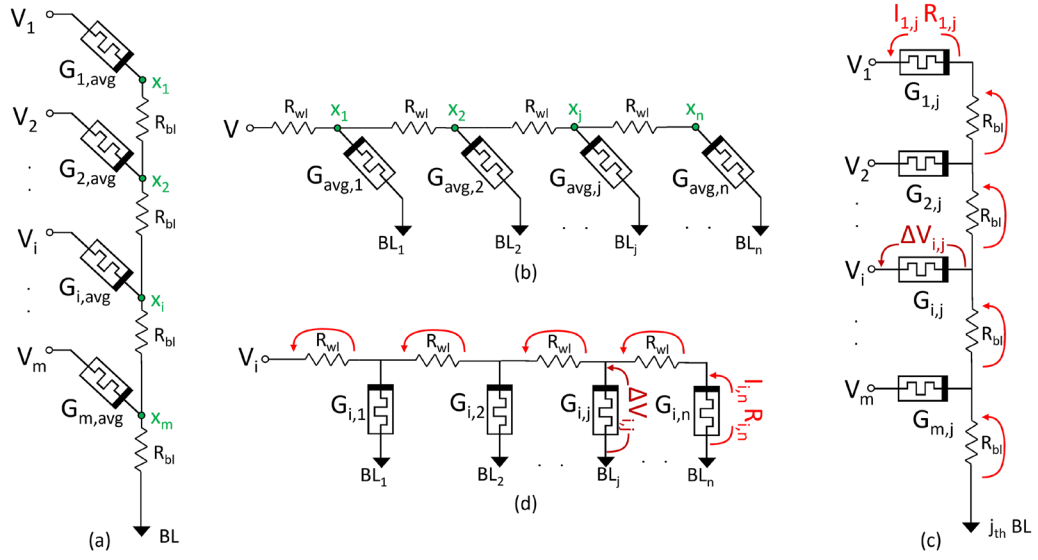


FIG. 3. Schematic of the approximations of the crossbar array over the BLs and WLs, according to DMR in panels (a) and (b) and the $\alpha\beta$-matrix model in panels (c) and (d). DMR considers just one equivalent BL (WL) and uses a matricial approach, while the $\alpha\beta$-matrix model considers generic BLs (WLs) and uses a standard equation approach. The figure is adapted from Refs. 21 and 23.

over the WLs and BLs, resulting in a more accurate estimation of the BLs output currents and of IR drops across the whole crossbar array. The two $m \times n$ matrices α and β account for the effect of the BLs and WLs parasitics, respectively. To derive the model parameters first, the currents ideally flowing in the array are computed using the approximation shown in Eq. (16), where V_{in} is the WLs input voltage and $G_{i,j}$ is the conductance of the device in the ith WL and jth BL,

$$I_{i,j} = V_{in} \cdot G_{ij}. \tag{16}$$

The coefficients of α and β are determined considering the normalized IR-drop on each device in the crossbar (i.e., $\Delta V_{i,j}/V_{IN_i}$, where $\Delta V_{i,j}$ is the voltage across the device G_{ij}). In particular, the elements of the α and β matrices are particular cases of the normalized IR drop, when the effect of parasitic resistance is neglected on the WLs and BLs, respectively. To compute the elements of α, the equivalent circuit in Fig. 3(c) is considered for each BL. The voltage across $G_{i,j}$ can be described by using Kirchhoff's voltage law reported in Eq. (17), where the sum of the voltage across $G_{1,j}$ and the voltages across all the interconnection resistances along the path to ground [i.e., red arrows in Fig. 3(c)] is considered. Similarly, V_{IN_i} can also be derived as shown in Eq. (18). In Eqs. (17) and (18), g indicates the parasitic conductance elements on the BLs or the WLs when computing the α and β coefficients, respectively,

$$\Delta V_{i,j} = I_{1,j} \cdot R_{1,j} + [(i-1)I_{1,j} + (i-2)I_{2,j} + \cdots + I_{i-1,j}]/g, \tag{17}$$

$$V_{IN_i} = I_{1,j}/G_{1,j} + [mI_{1,j} + (m-1)I_{2,j} + \cdots + I_{m,j}]/g. \tag{18}$$

The coefficients of α can be computed as in the following equation:

$$\alpha_{i,j} = \frac{\Delta V_{i,j}}{V_{IN_i}} = \frac{I_{1,j} \cdot g_{bl} + \left[\sum_{k=1}^{i} (i-k) I_{k,j}\right] \cdot G_{1,j}}{I_{1,j} \cdot g_{bl} + \left[\sum_{k=1}^{m} (m-k+1) I_{k,j}\right] \cdot G_{1,j}}. \tag{19}$$

The same approach can be used for the WLs, however, considering the approximated circuit shown in Fig. 3(d). The elements of β can be computed as in the following equation:

$$\beta_{i,j} = \frac{I_{i,n} \cdot g_{wl} + \left[\sum_{k=1}^{n-j+1} (k-1) I_{i,j+k-1}\right] \cdot G_{i,n}}{I_{i,n} \cdot g_{wl} + \left[\sum_{k=1}^{n} k I_{i,k}\right] \cdot G_{i,n}}. \tag{20}$$

Finally, the effective conductance matrix is obtained by multiplying the coefficient matrices and G, and the BLs output currents are given by the following equation:

$$I_o = V_{in} \times (\alpha \cdot G \cdot \beta). \tag{21}$$

As for the previous methods, this model is proved to be significantly faster than solving Kirchhoff's system of equations, with a computation complexity $o(mn)$.

B. Iterative models

Iterative methods include the solution implemented within the *CrossSim*[26] simulator and a similar approach proposed by Lepri *et al.*[30] In particular, CrossSim employs an iterative algorithm, depicted in Fig. 4, to efficiently compute the currents and voltages for crossbar arrays affected by parasitic resistances in the interconnects. This numerical method is specifically optimized for handling IR-drop effects in large-scale crossbars, providing a fast and reliable estimation of the voltage drop across each element of the crossbar. The algorithm initializes the system assuming ideal conditions: no initial voltage drops due to interconnect resistances (i.e., $R_{wl} = R_{bl} = 0$) and uniform input voltages across the BLs. Then, an iterative process refines the voltage drops estimation accuracy by first computing the current matrix via element-wise multiplication of the conductance and voltage matrices. Then, the voltage drops on WLs and BLs are estimated independently by multiplying cumulative currents by the corresponding lumped parasitic resistance element (i.e., R_{wl} or R_{bl}), as detailed in Eqs. (22) and (23). The total parasitic voltage drop across each device is the sum of these individual drops. Subsequently, ΔV, i.e., the deviation of the newly estimated voltage matrix with respect to the one from the previous iteration, is computed. Finally, ΔV together with a relaxation parameter, γ, are used to update the voltage matrix. This cycle is iteratively repeated until the estimation error between successive iterations (i.e., ΔV) falls below a predefined accuracy threshold, at which point the final updated voltage matrix is returned,

$$V_{par\_wl,i,j} = R_{wl} \sum_{j'=1}^{j} \sum_{k=j'}^{n} I_{i,k}, \tag{22}$$

$$V_{par\_bl,i,j} = R_{bl} \sum_{k=m+1-i}^{m} \sum_{l=1}^{k} I_{l,j}. \tag{23}$$

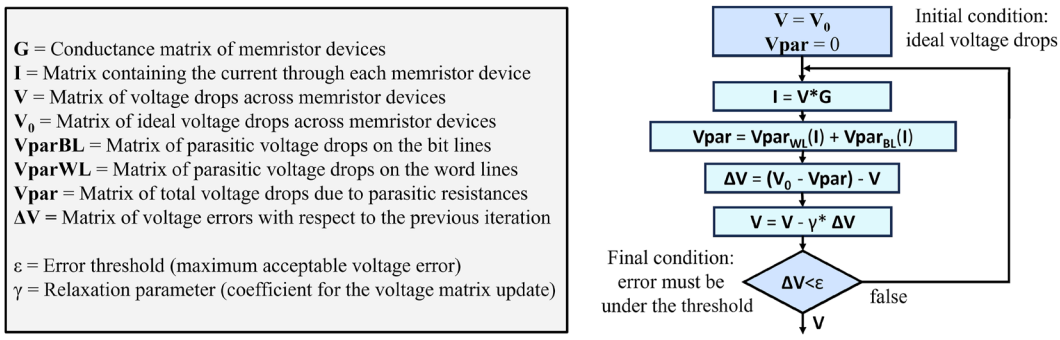


FIG. 4. Flow chart of the CrossSim iterative algorithm for voltage drop estimation.

It is important to mention that the choice of the relaxation parameter, γ, significantly impacts the algorithm's efficiency. Higher values of γ can accelerate convergence; however, they may also lead to instability due to aggressive updates of the voltage matrix. Therefore, optimizing this parameter is crucial for simulations. While the original CrossSim simulator implementation keeps γ fixed, this approach is not always optimal, particularly in our simulations where crossbar parameters vary. To ensure a fair comparison across different crossbar conditions, we optimized the relaxation parameter by making it inversely proportional to the two most critical parameters, i.e., parasitic resistance and crossbar size.

C. Kirchhoff's equation solvers

The complete system of equations modeling crossbar arrays, including parasitic resistors, can be solved using conventional SPICE simulators. However, such implementations result in excessively long computing times. However, they offer a reliable solution, essential for the sake of benchmarking, and can also account for the nonlinear characteristic of memory devices.

The *MemTorch*[29] simulator uses Chen's[27] model, which also solves Kirchhoff's equations in an exact and efficient way by leveraging optimized sparse matrix operations for fast linear algebra computations. In particular, MemTorch implements the sparse supernodal LU factorization with partial pivoting to solve the system of equations.[29] This method is parallelizable and can exploit CUDA acceleration. Consequently, the model is capable of providing a highly precise solution and a considerably lower computational cost compared to SPICE simulators, leading to faster computing times.

III. BENCHMARKING OF THE MODELS UNDER DIFFERENT OPERATING CONDITIONS

A. Benchmark metrics definition

A comparison between the different parasitic-aware crossbar models is essential to identify which model provides the optimal balance between accuracy, execution speed, and robustness across a range of different parameter configurations. Thus, the different models introduced in Sec. II were compared considering the following metrics: (i) the mean relative error (MRE) on the BLs output currents, (ii) normalized simulation time, and (iii) robustness against crossbar's parameter variations. In particular, the MRE of the BLs output currents is computed using Eq. (24) for an $m \times n$ crossbar and quantifies the deviation of the results of the model (i.e., I_{model}) with respect to the reference value obtained from SPICE simulations (i.e., I_{ref}), providing a clear indication of the model's accuracy. Finding a model with good accuracy is the main goal for many applications, as it can faithfully simulate the behavior of the real hardware,

$$\mathrm{MRE}\,(\%) = \frac{1}{n}\sum_{j=1}^{n}\frac{\left|I_{\mathrm{model},j} - I_{\mathrm{ref},j}\right|}{I_{\mathrm{ref},j}} \times 100\%. \tag{24}$$

The normalized simulation time was computed as the ratio between the simulation time of the model and the execution time of the ideal VMM (i.e., $I = V_{in} \cdot G$ computed in software). All simulations were performed on the same computing machine mounting an AMD Ryzen 5 5500u, 2.1 GHz CPU. Indeed, the simulation speed is another key metric of performance for parasitic-aware crossbar models, especially when considering ANN hardware-aware training applications, which require fast models to alleviate the computational effort required to complete the task. Finally, the robustness metric, by quantifying how much the model is immune against the change of a crossbar's parameter, provides an indication about the usability and flexibility of the model. To compute the robustness metric, a set of N parametric sweep simulations were performed, and the MRE of the BLs output currents was computed for each instance. The robustness to a parameter (p) variation was then computed as the inverse of the standard deviation of the MRE, as described by the following equation:

$$\mathrm{Robustness}_p = \frac{1}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\mathrm{MRE}_i - \overline{\mathrm{MRE}}\right)^2}}. \tag{25}$$

The robustness was computed for the following key parameters: array size, parasitic resistance value, memory window (MW, i.e., the ratio between the HRS and LRS resistances), variability (i.e., the effect of device programming variations), and sparsity (i.e., the percentage of devices in the crossbar programmed in HRS). These parameters were selected to provide information regarding the robustness of the different models to specific characteristics of the crossbar, the memory devices, and of application-related features.

B. Simulation setup

All models were implemented using optimization techniques, such as vectorization and parallelization, and simulated using Python 3.11. The memory devices were modeled as ideal linear resistors. This approximation is typically valid for eNVM technologies, such as RRAMs, which exhibit negligible nonlinearity under small read bias voltages. This is the case for the simulations performed in this analysis, where a V_{READ} of 0.3 V was applied to all WLs of the crossbar. In addition, for the scope of this analysis, only binary values were mapped into the devices' resistances. To estimate each benchmark metric, each condition was simulated repeatedly, considering randomized patterns. In experiments considering the effect of variability due to device programming, the HRS and LRS resistances were modeled considering the physics-based RRAM compact model described in Ref. 31, calibrated on a TiN/Ti/HfO_x/TiN RRAM technology from SEMATECH. In particular, the compact model, which reproduces the behavior of filamentary RRAM devices, considers the device resistance as the sum of a metallic-like conductive filament component with cross section S and a dielectric barrier component of thickness x when the device is in HRS. Device programming variations are reproduced by the inclusion of zero-mean Gaussian noise sources on S and x, with standard deviations of 0.35 nm^2 and 0.7 nm, respectively. As a result, the model reproduces the experimentally observed normal LRS and log-normal HRS resistance distributions.[31] Finally, simulations of the crossbar including parasitic resistances performed either with Ngspice or an accurate implementation of CrossSim model (i.e., with a target error $<10^{-4}$ V on the potential matrix of the crossbar) were used as a reference for accuracy evaluation.

C. Results of the comparative analysis

First, the accuracy (i.e., low MRE of the estimated output currents) of the analytical models was compared for changing MW, parasitic resistance value, and array size. In particular, these parameters were swept over a range of values consistent with the literature,[30] with parasitic resistance ranging from 0.1 to 5 Ω, MW ranging from 2 up to 50, and considering three typical array sizes (i.e., 32 × 32, 64 × 64, and 128 × 128). As shown in Figs. 5(a)–5(c), the DMR model is always outperformed by the other two models. The DMR and $\alpha\beta$-matrix models follow similar model approaches; however, the former averages the conductance over the WLs and BLs, which results in slightly less accurate estimates, as highlighted in the upper panels in Fig. 5. For sufficiently large values of MW, the accuracy of the models is primarily influenced by the parasitic resistance R_P, or more precisely by the R_{LRS}/R_P ratio, with the MRE increasing as R_P increases. However, when the MW is reduced below 10, a non-negligible dependence of the MRE on MW is observed. In addition, it can be observed that the MRE increases with the array size, with the $\alpha\beta$-matrix model being more affected than Jeong's model. In particular, the $\alpha\beta$-matrix model is more accurate for small arrays and at low values of parasitic resistance, while Jeong's model, instead, shows increased robustness for larger arrays and higher parasitic resistance values. It is important to note that in all the simulations, the optimal value minimizing MRE of the parameter p, which is used in Jeong's model to compute R_{avg}, was used. Although Jeong's model is more accurate under these worst-case conditions, its estimates suffer from high MRE (i.e., $MRE > 10\%$). These results are also evident by looking at the output current estimates shown in Figs. 6(a)–6(c) for three specific crossbar configurations. In fact, DMR and $\alpha\beta$-matrix models estimates are nearly always overlapped, with the latter achieving a slightly higher accuracy. Moreover, these images provide details on the cause of high MRE: especially in Fig. 6(c), Jeong's model systematically underestimates the currents, while DMR and $\alpha\beta$-matrix models predict higher current values with respect to the reference (i.e., SPICE simulation). These trends suggest that a more accurate analytical model could be achieved by combining the results of multiple models; however, this analysis is outside the scope of this work and will be explored in the future.

A clearer overview of the different trade-offs existing in the analyzed analytical models is provided by the radar charts reported in Figs. 7(a)–7(c). From these charts, the performance of the models across all metrics can be assessed concurrently for different array sizes. In terms of simulation speed, Jeong's model is the fastest one, thanks to its compact formulation. In addition, it achieves better accuracy and overall robustness to parameter variations for larger array sizes, with the exception for MW variations. The increased

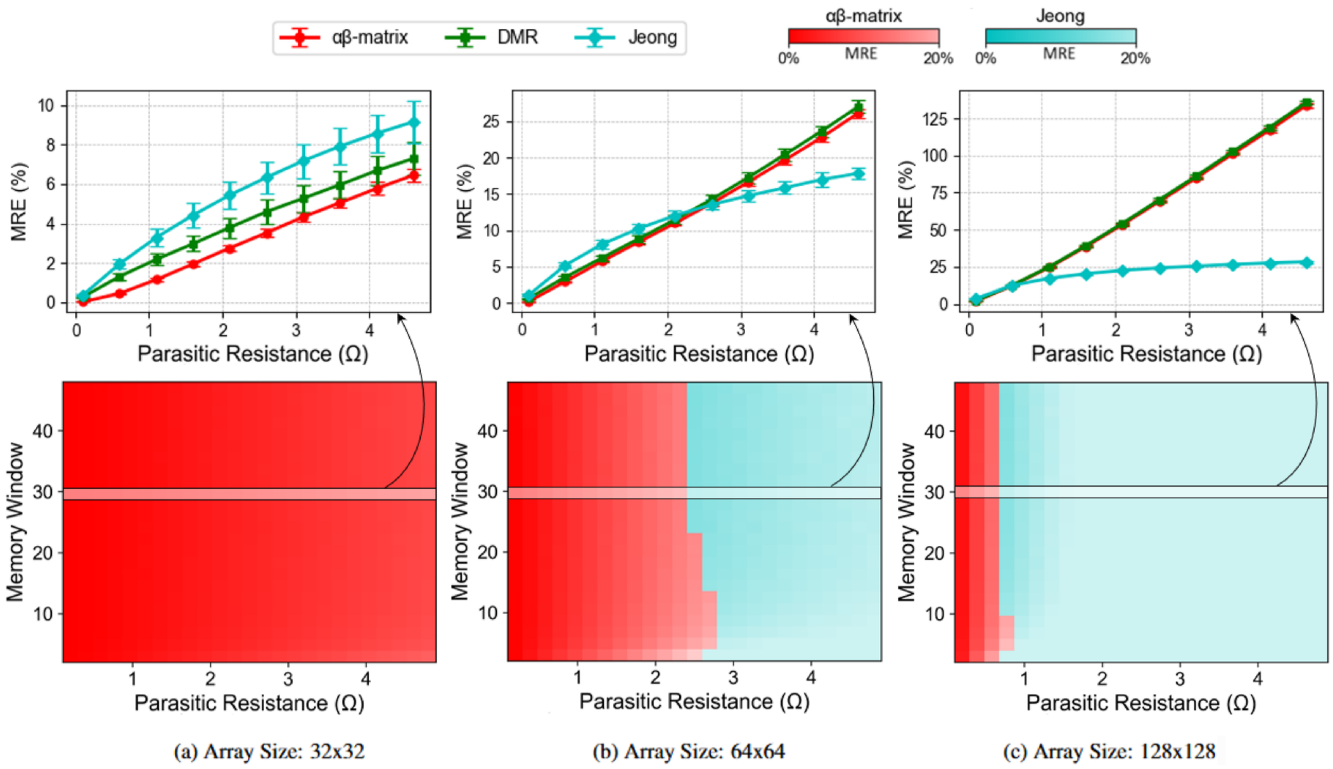
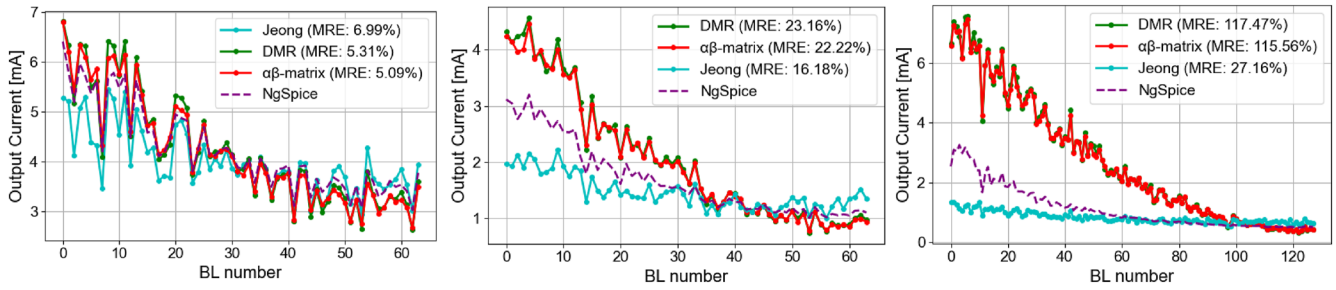


FIG. 5. MRE (%) on the estimated BLs output currents for varying parasitic resistance and MW (bottom panels) and for a fixed MW of 30 (upper panels). A value of $R_{LRS} = 1$ kΩ and 50% of devices programmed in HRS were considered. In the bottom panels, only the model achieving the lowest error is shown for each pair of parasitic resistance and MW values and the luminance indicates its corresponding MRE. The DMR model is not visible as it is always outperformed by the other models, as shown in the graphs at fixed MW (upper panels).

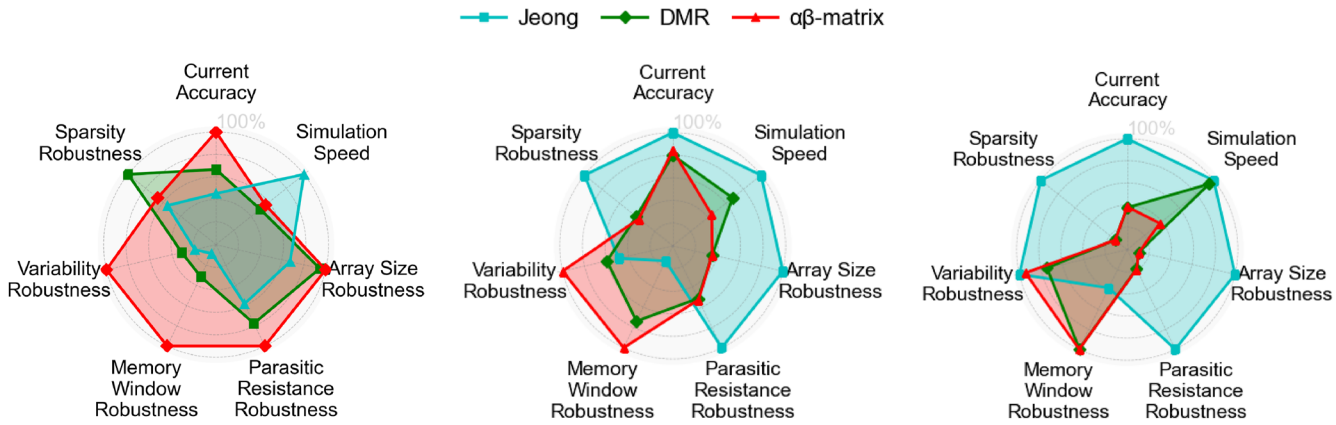


(a) Parasitic Resistance: 1 Ω, Array Size: 64x64 (b) Parasitic Resistance: 4 Ω, Array Size: 64x64 (c) Parasitic Resistance: 4 Ω, Array Size: 128x128

FIG. 6. Example of the BL output currents estimated by analytical models, along with the reference obtained from a SPICE simulator. Different values of parasitic resistance and array sizes were considered: (a) 1 Ω, 64 × 64 array; (b) 4 Ω, 64 × 64 array; and (c) 4 Ω, 128 × 128 array. The LRS resistance is 1 kΩ; the MW is 30%; and 50% of devices are programmed in HRS.

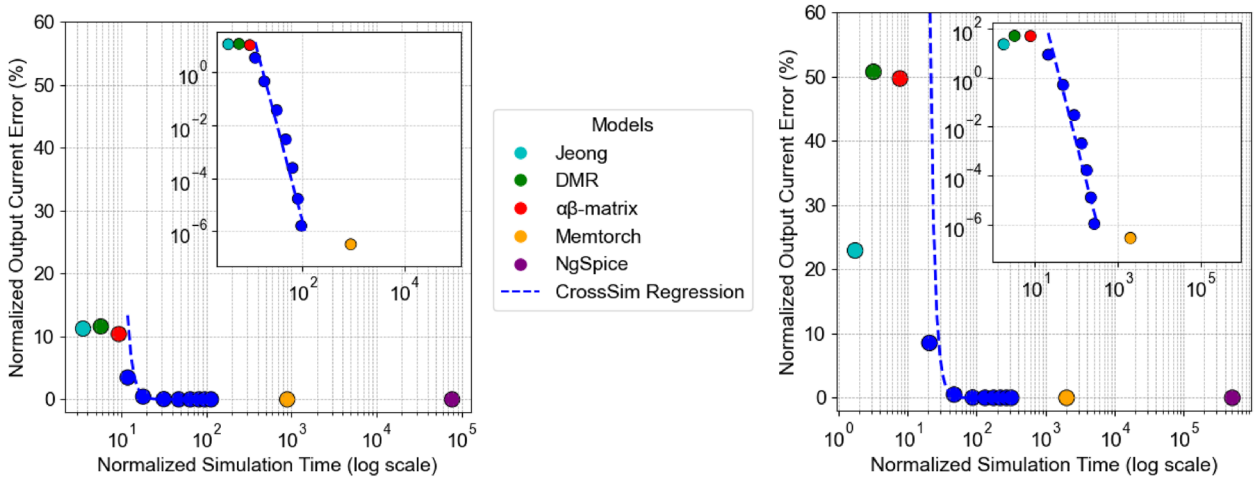
robustness to parameter variations of Jeong's model is likely linked to the averaging operation performed within the model. For smaller array sizes, the $\alpha\beta$-matrix model achieves higher accuracy and robustness to parameter variations with respect to the other models. However, its performance quickly degrades for medium and large array sizes, as shown in Figs. 7(b) and 7(c), respectively. However, with respect to the DMR model, the $\alpha\beta$-matrix model achieves better performance at the cost of increased execution time. The longer execution times and the higher performance of the $\alpha\beta$-matrix model compared to the DMR model are linked to the model formulation, in which the conductance of all devices is used to estimate the crossbar output currents, thus capturing the details of the programmed devices.

Eventually, it is useful to compare the performance of analytical models with respect to Kirchhoff's equation solvers and the iterative ones. Figure 8 shows the accuracy vs simulation time performance of the different models for a 64 × 64 and a 128 × 128 crossbar. Ideally, a model should be both accurate and fast (i.e., located in the lower-left corner of the plot in Fig. 8). However, it is clear that none of the models can achieve such performance and an accuracy vs simulation time trade-off exists. On the one hand, optimized Kirchhoff's equation solvers (such as Memtorch and Ngspice, shown in Fig. 8) offer the highest accuracy. However, this comes at the cost of significantly longer execution times (e.g., from 10 to 10^5 times slower compared to other models reported in Fig. 8), which increase rapidly with larger array sizes. Conversely, analytical models enable computing results in a shorter amount of time, however, at much lower accuracy. Iterative models, such as the one implemented in CrossSim, fill the gap between the two other approaches, enabling to achieve high estimation accuracy, with simulation times that are only 6–15 times longer with respect to analytical models. In addition, the CrossSim model offers high flexibility since the accuracy vs simulation time



Small Array Size [10x10, 50x50] Medium Array Size [50x50, 100x100] Large Array Size [100x100, 150x150]

FIG. 7. Radar charts summarizing model performance for three different array size ranges, which are reported below each chart. Simulations also varied the parasitic resistance [0.1 Ω, 5 Ω], memory window [10, 100], and sparsity [10%, 100%]. Each condition was evaluated over multiple variability seeds, considering different programming patterns and conductance fluctuations.



(a) Array Size: 64x64 (b) Array Size: 128x128

FIG. 8. Scatterplot: each dot represents a different model (multiple instances of the CrossSim model are set with a different accuracy level). A logarithmic regression line is shown in the case of the CrossSim model to better show its flexibility. The scale is semi-logarithmic for the main plot and logarithmic for the inset. Simulations were performed with two crossbars of size 64 × 64 (a) and 128 × 128 (b), 2 Ω of parasitic resistance and a memory window of 40. In both cases, 50% of devices are programmed in HRS.

trade-off can be selected by the user by changing the target estimate error. As shown in the inset in Fig. 8, this trade-off follows a power law. However, compared to analytical models, the execution time of iterative models increases more rapidly for increasing array size. Despite the influence of the employed computing hardware, it is also useful to examine the absolute simulation times required by the different models. In order to compute the solution of a 64 × 64 and a 128 × 128 crossbar array with NgSpice 0.4 and 4.5 s are required, respectively. Conversely, 20 and 70 μs are needed when using Jeong's model for the two array sizes, respectively. Although SPICE simulation times may not seem excessive, they can easily become a limiting factor when considering realistic use cases. In fact, the programmed weight distribution varies with the application and performed task, meaning that multiple simulations are required during the design phase of the crossbar architecture. In addition, intrinsic device variability further increases the total number of required simulations. Moreover, as larger crossbar arrays are targeted, SPICE simulation times increase exponentially with the array size, whereas analytical models are less affected. Furthermore, the importance of compact crossbar parasitic resistance models is emphasized in the context of ANNs hardware accelerators. These models require parasitic-aware training strategies involving numerous iterations, which would lead to excessive training times when using SPICE simulators to solve the crossbar network.

Thus, although the results of this analysis still suggest that iterative models seem to be the best choice for applications such as parasitic-aware training of neural networks, the intrinsic tolerance to noise of ANNs may still enable the use of the analytical models in such applications, with a considerable time saving. In light of these considerations, all the models are benchmarked over a use case application consisting of a parasitic-aware training of a multi-layer perceptron (MLP) classifying the MNIST dataset.[28]

IV. MODELS' COMPARISON IN A PARASITIC-AWARE OFFLINE TRAINING TASK OF A CLASSIFIER

Training the parameters of an ANN typically involves the evaluation of the classification error of the network over several samples from a training dataset and a subsequent backpropagation of the error and adjustment of the network parameters. This procedure is repeated multiple times until the error, estimated over a different portion of the dataset, is below a defined threshold. When employing novel memristor-based VMM analog hardware accelerators, additional steps are required in order to retain the accuracy achieved in software. In particular, when mapping the software-trained network parameters, hardware nonidealities, such as the crossbar array nonlinearity and parasitics, must be considered. This is typically done by performing a final fine-tuning of the software-trained network parameters by means of either online[32,33] or offline[21,34,35] learning steps. In online learning, parameters are optimized by implementing backpropagation on the final hardware that is supposed to run the network. Despite its promising effectiveness,[32,33] this approach leads to increased hardware complexity and cost and a nonguaranteed portability of the retrained network parameters to other replicas of the same device. Conversely, offline learning aims to adjust the network parameters to account for the hardware nonidealities that are introduced in the training algorithm using compact models of the hardware components. The key advantages of this approach are the simpler hardware implementation and the portability of the trained network parameters to multiple replicas of the same device. However, to enable its implementation, it is clear that accurate and time-efficient compact models are needed.

Thus, to analyze the suitability of the different parasitic-aware crossbar models for online parasitic-aware training of ANNs, we trained multiple networks on PyTorch to perform an MNIST

classification task using the different models. The MNIST classification problem can be solved using a variety of neural networks, such as MLPs and convolutional neural networks (CNNs). For this comparative study, we used the relatively simple network made of two fully connected layers shown in Fig. 9. In the parasitic-aware training algorithm, we modified the layers to integrate the parasitic resistance models within the network. These custom layers use a user-selected crossbar model to account for line parasitics, providing a more realistic estimate of the crossbar output currents compared to directly performing the VMM between the inputs and the weights. In particular, in the forward pass, the software-trained weight matrix (W) is mapped to RRAM devices' conductances (G). For this purpose, a simple linear transformation is used, where the maximum and minimum values of the weights are determined and then mapped to the corresponding extremes of the conductance range (i.e., G_{min} and G_{max}, respectively). Subsequently, the corresponding conductance values are input to the analyzed crossbar model, which is used to estimate the crossbar output currents. These output currents are then scaled back to the original output domain and propagated to the next layer. This parasitic-aware simulation is executed at each training step, before the backward pass, ensuring that the gradient is computed based on realistic outputs, affected by IR-drop. Regarding backpropagation, the automatic differentiation functionality of PyTorch[36] was used, as it provides a flexible backward pass that adapts to different models and accounts for cases where analytical implementation solutions are unavailable (e.g., iterative models). Moreover, it ensures a fair comparison across different models without excessively increasing the computational cost. During the training phase, conductance values are considered with full precision, while 5-bit quantization is used for inference. As the goal of this analysis is to compare different parasitic-aware crossbar models, the effects introduced by other circuit components (e.g., ADC quantization) were not considered. Nevertheless, embedding the crossbar simulation within the network layers inevitably increases training time. While some of this overhead comes from mapping weights to conductances, most of the slowdown is caused by the execution of the parasitic-aware models, which is an important factor to compare.

First, the impact of the line parasitic resistances on the software-trained network (i.e., the network trained without considering parasitic resistances) accuracy was analyzed. In particular, the trained network's weights were mapped onto the two ANN layers implemented on crossbars arrays and then the inference task was simulated using CrossSim as the reference model for accounting for parasitic resistances. CrossSim was used as it ensures accurate results at a shorter simulation time compared to SPICE simulations, as discussed in Sec. III C. As shown in Fig. 10, a huge loss in accuracy is observed even for very small values of parasitic resistance. This is likely due to the large number of rows (i.e., 784) of the crossbar encoding the weights of the first layer, which leads to considerable distortion of the BLs output currents. This is confirmed by the following analysis, in which, by using the tiling technique,[37] the network layer is mapped not onto a single large crossbar (e.g., 784×128), but onto multiple smaller crossbar arrays (i.e., tiles), each handling a portion of the VMM operation. In particular, slices of the input vector are shared across the tiles on the same row, while tiles within the same column generate partial sum outputs, which are then combined. This approach is designed to reveal the network performance dependence on the crossbar array size, given that this is the main source of parasitic effects. As expected, Fig. 10 shows that the accuracy is better preserved with smaller tiles; however, even in those cases, the accuracy of the network is poor, highlighting the need for hardware-aware training.

Thus, we evaluated the performance of the different parasitic-aware models described in Sec. II when used to train the weights and biases of the same network (see Fig. 9). To achieve a broad overview of the performance of the different models under different conditions, the training procedure was repeated multiple times, varying the tile size starting from 16×16 up to 784×128 while considering

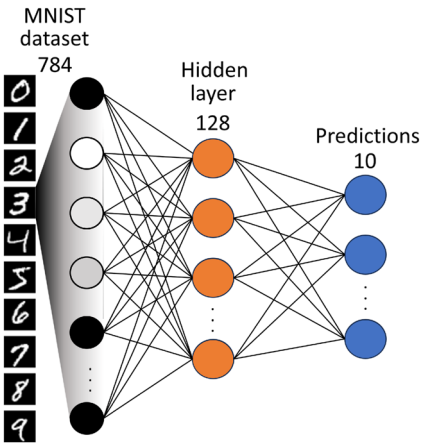


FIG. 9. Representation of the network used for the MNIST classification task. The 784 pixels in the image of the hand-written number are given as inputs of the network and ten output neurons are used to classify the digit. A hidden layer of 128 neurons with ReLU activation function is involved in the classification process.

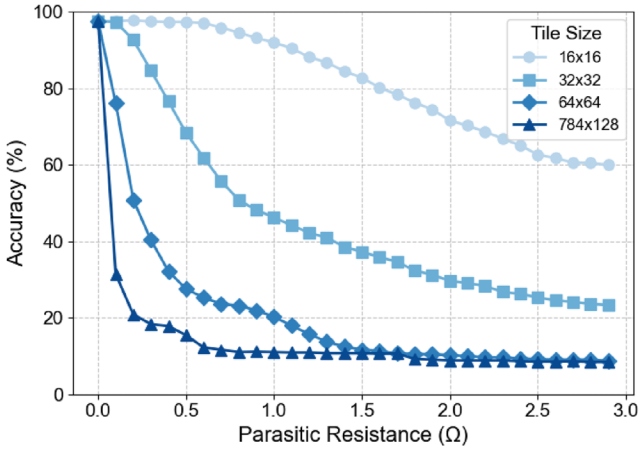


FIG. 10. Inference accuracy of the software-trained network (i.e., trained without considering parasitic resistances) on 1000 MNIST evaluation samples. Multiple simulations of the hardware are performed for different tile sizes and parasitic line resistances. The weights are mapped to devices with resistance ranging from 1 to 40 kΩ with 32 levels of quantization.

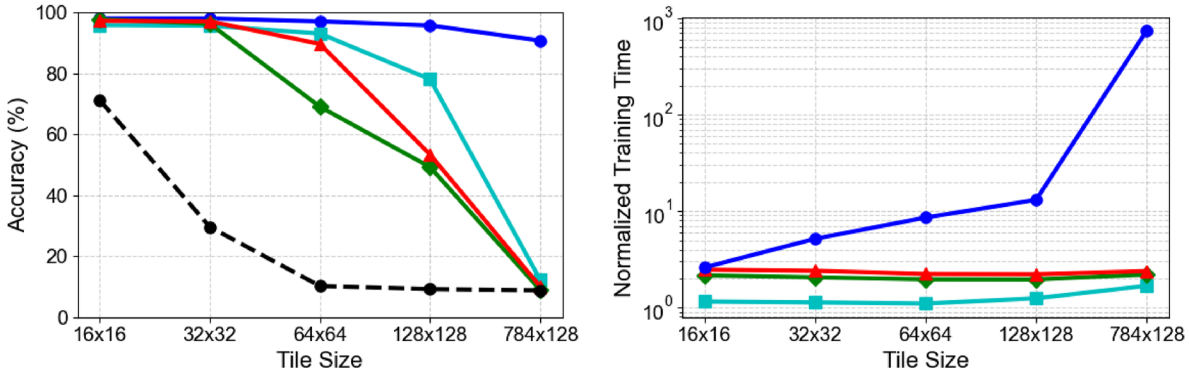


FIG. 11. (a) Inference accuracy of the hardware-aware trained networks on the MNIST evaluation dataset, tested with a high-precision iterative model. (b) Normalized training time per epoch, relative to the ideal software (trained w/o parasitic resistances) network model epoch duration [i.e., dashed black line in panel (a)]. In both cases, each point represents a network trained with a different model and tile size; all networks were trained considering a 2 Ω line resistance and a MW of 40.

a parasitic resistance of 2 Ω and a MW of 40. The training of each network was stopped when an accuracy of at least 97% was reached on the test dataset. To verify the effectiveness of crossbar models in mitigating the effect of parasitics, each trained network was compared to the results obtained when using the CrossSim model with very low target error (i.e., $10^{-4}V$). As shown in Fig. 11(a), analytical models can recover part of the lost accuracy; however, as the tile size increases, the effectiveness of the parasitic compensation drops rapidly. In fact, all three analytical models can retain most of the accuracy up to 32 × 32 tile size, with Jeong's and $\alpha\beta$-matrix models being robust up to 64 × 64 tile sizes. This is consistent with the results reported in Sec. III C, which showed that the MRE computed on the BLs output currents introduced by analytical models in large arrays ranged from 10% to 20%, leading unavoidably to increased accuracy loss. In this scenario, the only dependable option is to train the network with an iterative model, however, at the cost of increased simulation times. In fact, using the CrossSim model with a target error of 10^{-4} V in the parasitic-aware training of the network enables to achieve a robust parameter set even when using 784 × 128 crossbar arrays [see Fig. 11(a)], but with a longer training time. This last factor is important to keep into consideration and is better examined in Fig. 11(b), where the time required to train one epoch, normalized with respect to the ideal software implementation, is evaluated for each model and array size. The simulation times for the analytical models show negligible dependence on array size, while CrossSim experiences a significant increase in computational time of up to two orders of magnitude for the largest size. Unfortunately, while analytical models outperform CrossSim in terms of computational speed, their accuracy drops in the very situations where they would be most beneficial (i.e., large array sizes).

A closer inspection of the trained weights is portrayed in Fig. 12, revealing a general strategy determined by the parasitic-aware training algorithm. For small tile sizes (i.e., 16 × 16 in Fig. 12), the trained weights follow a normal distribution centered around the middle of the defined conductance range, as expected. However, as the tile size increases, together with the effect of line parasitic resistances, the weight distribution drifts from the normal one and moves toward lower conductances. In fact, with the majority of devices programmed in HRS, the crossbar is less affected by the parasitic. In addition, the parasitic-aware training implements a topological compensation strategy, assigning lower resistance value to devices that are located further from the inputs and the outputs (i.e., the top right region of the crossbar in Fig. 1), thus balancing the IR-drop where most critical. In addition, the network learned to compensate for the output current distortion by reducing the biases progressively across the columns.

Another important aspect to consider is the robustness of the trained parameter set to the variation of critical circuit-related parameters, such as the values of parasitic resistances and of the MW, which may be affected by device-to-device variability. As shown in Fig. 13, the trained network suffers from a strong sensitivity to variations, especially related to the value of parasitic resistances, or more precisely, to the ratio of R_{LRS}/R_P[38] as R_{LRS} is fixed to 1$k\Omega$. In fact, during the training, specific values of parasitic resistance and

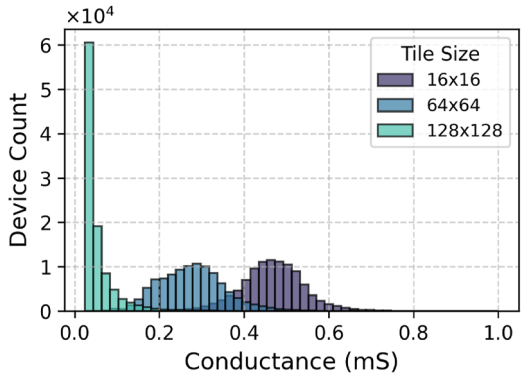


FIG. 12. Distributions of the conductance associated with the weights of neural networks parasitic-aware trained with the CrossSim model using three different tile dimensions. Weights are mapped to conductances in the range of 0.025–1 mS, with device count shown across 32 bins.

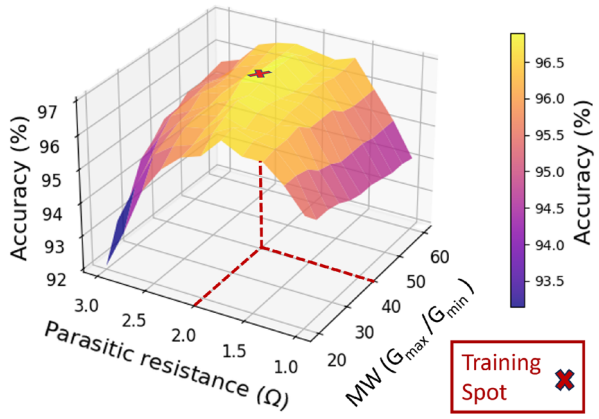


FIG. 13. Classification accuracy of the CrossSim parasitic-aware trained network with 32 × 32 tile size. The parasitic resistance and MW used during the parasitic-aware training are highlighted (i.e., training spot). The accuracy is shown as a function of parasitic resistance and MW, highlighting its sensitivity to parameter variations.

MW are set and the weights of the network adapt differently based on those values. Even a slight change in these parameters leads to additional degradation of the inference accuracy. Indeed, as illustrated in Fig. 13, a deviation from the trained parameters (i.e., $R_P = 2\ \Omega$, MW = 40) results in decreased accuracy, with a significant dependence on the values of parasitic resistance and, for sufficiently large MW values, a less pronounced dependence on the MW, which is consistent with other results reported in the literature.[38] Thus, solutions aimed at increasing the robustness to parasitic resistance variations should be implemented in the training algorithm.

V. CONCLUSION

In this work, we compared the main parasitic-aware crossbar models available in the literature over a broad range of operating conditions. The results of the comparison highlighted clear trade-offs between model accuracy, execution time, and robustness. In particular, DMR and α–β models are better suited for the simulation of small crossbar arrays as their estimation error rapidly grows with increasing array size. Conversely, Jeong's model, despite being the most compact among the ones we analyzed, and thus faster at the cost of lower accuracy, provides more robust estimates for larger array sizes. In addition, we analyzed the performance of the models when used in a hardware-aware training task of a MLP NN classifying the MNIST dataset. The results suggest that although analytical models can speed up the parasitic-aware training of the network, they are only applicable for the simulation of relatively small crossbars (i.e., up to 32 × 32 arrays), thus restricting their usability. When the NN layers are mapped to larger crossbar arrays, iterative models should be preferred, as they currently provide the best trade-off between accuracy and execution speed.

ACKNOWLEDGMENTS

T. Zanotti and F. M. Puglisi acknowledge financial support from PNRR MUR under Project No. ECS_00000033_ECOSISTER.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Alessandro Lambertini: Conceptualization (equal); Data curation (lead); Formal analysis (equal); Investigation (lead); Methodology (equal); Software (lead); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Tommaso Zanotti**: Conceptualization (equal); Data curation (supporting); Formal analysis (equal); Funding acquisition (supporting); Investigation (supporting); Methodology (equal); Project administration (supporting); Resources (equal); Software (supporting); Supervision (lead); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Paolo Pavan**: Formal analysis (supporting); Funding acquisition (supporting); Methodology (supporting); Project administration (supporting); Resources (supporting); Supervision (supporting); Visualization (supporting); Writing – review & editing (equal). **Andrea Padovani**: Formal analysis (supporting); Funding acquisition (supporting); Methodology (supporting); Project administration (supporting); Visualization (supporting); Writing – review & editing (equal). **Francesco Maria Puglisi**: Conceptualization (supporting); Formal analysis (supporting); Funding acquisition (lead); Investigation (supporting); Methodology (supporting); Project administration (lead); Resources (equal); Supervision (supporting); Visualization (supporting); Writing – original draft (supporting); Writing – review & editing (equal).

DATA AVAILABILITY

The data that support the findings of this study are openly available in GitHub at https://github.com/alambertini01/Crossbar_Models_Comparison.

REFERENCES

[1] S. Samsi, D. Zhao, J. McDonald, B. Li, A. Michaleas, M. Jones, W. Bergeron, J. Kepner, D. Tiwari, and V. Gadepally, in *2023 IEEE High Performance Extreme Computing Conference (HPEC)* (IEEE, 2023), pp. 1–9.
[2] S. Luccioni, Y. Jernite, and E. Strubell, in *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency, FAccT'24* (Association for Computing Machinery, New York, NY, 2024), pp. 85–99.
[3] Decadal plan for semiconductors—SRC.
[4] G. Indiveri and S.-C. Liu, Proc. IEEE **103**, 1379 (2015).
[5] P. Delestrac, J. Miquel, D. Bhattacharjee, D. Moolchandani, F. Catthoor, L. Torres, and D. Novo, in *2024 IEEE 35th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)* (IEEE, 2024), pp. 143–151.
[6] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)* (IEEE, 2016), pp. 27–39.
[7] Y. Xi, B. Gao, J. Tang, A. Chen, M.-F. Chang, X. S. Hu, J. V. D. Spiegel, H. Qian, and H. Wu, Proc. IEEE **109**, 14 (2021).
[8] T. P. Xiao, C. H. Bennett, B. Feinberg, S. Agarwal, and M. J. Marinella, Appl. Phys. Rev. **7**, 031301 (2020).
[9] I. Chakraborty, M. Ali, A. Ankit, S. Jain, S. Roy, S. Sridharan, A. Agrawal, A. Raghunathan, and K. Roy, Proc. IEEE **108**, 2276 (2020).

[10]W. Wan, R. Kubendran, C. Schaefer, S. B. Eryilmaz, W. Zhang, D. Wu, S. Deiss, P. Raina, H. Qian, B. Gao, S. Joshi, H. Wu, H.-S. P. Wong, and G. Cauwenberghs, Nature **608**, 504 (2022).
[11]A. Serb, A. Corna, R. George, A. Khiat, F. Rocchi, M. Reato, M. Maschietto, C. Mayr, G. Indiveri, S. Vassanelli, and T. Prodromakis, Sci. Rep. **10**, 2590 (2020).
[12]L. Benatti, T. Zanotti, D. Gandolfi, J. Mapelli, and F. M. Puglisi, Nano Futures **7**, 025003 (2023).
[13]H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, Proc. IEEE **100**, 1951 (2012).
[14]S. Slesazeck and T. Mikolajick, Nanotechnology **30**, 352003 (2019).
[15]N. Zagni, F. M. Puglisi, P. Pavan, and M. A. Alam, Proc. IEEE **111**, 158 (2023).
[16]D. Ielmini and H.-S. P. Wong, Nat. Electron. **1**, 333 (2018).
[17]M. J. Marinella, S. Agarwal, A. Hsia, I. Richter, R. Jacobs-Gedrim, J. Niroula, S. J. Plimpton, E. Ipek, and C. D. James, IEEE J. Emerg. Sel. Top. Circuits Syst. **8**, 86 (2018).
[18]A. Amirsoleimani, F. Alibart, V. Yon, J. Xu, M. R. Pazhouhandeh, S. Ecoffey, Y. Beilliard, R. Genov, and D. Drouin, Adv. Intell. Syst. **2**, 2000115 (2020).
[19]M. E. Fouda, S. Lee, J. Lee, A. Eltawil, and F. Kurdahi, IEEE Trans. Nanotechnol. **18**, 704 (2019).
[20]M. E. Fouda, S. Lee, J. Lee, G. H. Kim, F. Kurdahi, and A. M. Eltawi, IEEE Access **8**, 228392 (2020).
[21]Y. Liao, B. Gao, P. Yao, W. Zhang, J. Tang, H. Wu, and H. Qian, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **40**, 1662 (2021).
[22]Y. Jeong, M. A. Zidan, and W. D. Lu, IEEE Trans. Nanotechnol. **17**, 184 (2018).
[23]T. Cao, C. Liu, Y. Gao, and W. L. Goh, in *2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)* (IEEE, Singapore, Singapore, 2021), pp. 122–126.
[24]X. Zhao, L. Liu, L. Si, K. Pan, H. Sun, and N. Zheng, in *2021 IEEE 14th International Conference on ASIC (ASICON)* (IEEE, 2021), pp. 1–4.
[25]D. Song, F. Yang, C. Wang, N. Li, P. Jiang, B. Gao, X. Miao, and X. Wang, IEEE Electron Device Lett. **44**, 1280 (2023).
[26]T. Xiao, C. Bennett, B. Feinberg, M. Marinella, and S. Agarwal, "CrossSim inference manual v2.0," Technical Report SAND2022-7074R, 1869509, 706724 (Sandia National Laboratories, 2022).
[27]A. Chen, IEEE Trans. Electron Devices **60**, 1318 (2013).
[28]Y. LeCun, C. Cortes, and C. Burges, MNIST handwritten digit database.
[29]C. Lammie and M. R. Azghadi, in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)* (IEEE, Seville, Spain, 2020), pp. 1–5.
[30]N. Lepri, M. Baldo, P. Mannocci, A. Glukhov, V. Milo, and D. Ielmini, IEEE Trans. Electron Devices **69**, 1575 (2022).
[31]T. Zanotti, P. Pavan, and F. m. Puglisi, Microelectron. Eng. **266**, 111886 (2022).
[32]P.-Y. Chen, X. Peng, and S. Yu, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **37**, 3067 (2018).
[33]Q. Zhang, H. Wu, P. Yao, W. Zhang, B. Gao, N. Deng, and H. Qian, Neural Networks **108**, 217 (2018).
[34]S. Lee, M. E. Fouda, J. Lee, A. M. Eltawil, and F. Kurdahi, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **42**, 521 (2023).
[35]T. Cao, C. Liu, Y. Gao, and W. L. Goh, IEEE J. Emerg. Sel. Top. Circuits Syst. **12**, 436 (2022).
[36]A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)* (Curran Associates, 2019).
[37]D. J. Mountain, M. R. McLean, and C. D. Krieger, IEEE J. Emerg. Sel. Top. Circuits Syst. **8**, 137 (2018).
[38]F. L. Aguirre, S. M. Pazos, F. Palumbo, J. Suñé, and E. Miranda, IEEE Access **8**, 202174 (2020).