Formal Verification of Error Bounds for Resistive-Switching-based Multilevel Matrix-Vector Multipliers

Kemal Çağlar Coşkun[ш] Chandan Kumar Jha[ш] Muhammad Hassan[ш,*] Rolf Drechsler[ш,*]
[ш]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[*]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
{kcoskun,chajha,hassan,drechsler}@uni-bremen.de

***Abstract*—The need for performance and energy efficiency by expanding technologies such as internet-of-things devices and *artificial neural networks* (ANNs) has led to the exploration of in-memory computing paradigms, specifically utilizing *resistive-switching memory* (RSM) based analog and multilevel *matrix-vector multipliers* (MVMs). However, nonidealities in these MVMs cause larger-than-expected deterioration in the output quality and introduce errors with potentially catastrophic consequences in safety-critical applications such as autonomous vehicles, medical diagnosis, and control systems. Therefore, to enable the use of MVMs in such applications, the error bounds of the MVMs must be formally verified, which, to our knowledge, remains unaddressed. In this paper, we aim to address this gap with a formal verification methodology for finding the maximum possible error in resistive-switching-based multilevel MVMs. We introduce three approaches to compute the maximum error and provide a polynomial-time solution as our primary contribution, which reduces the computation time by up to 2181 times. Additionally, we provide a tracing feature for error source identification and debugging, enabling targeted enhancements of the design. We demonstrate the methodology's efficiency with a timing analysis and its effectiveness through a case study using the metrics of a fabricated design from the literature. We made the source code of our software implementation publicly available to promote further research in the use of multilevel MVMs in safety-critical applications.**

***Index Terms*—memristors, neuromorphics, resistive RAM, formal verification, analog computing.**

I. Introduction

Multilevel *matrix-vector multipliers* (MVMs) represent the intersection of two emerging computing paradigms: *in-memory computing* (IMC) and analog computing. IMC aims to reduce the energy consumption and latency of data-intensive applications by performing computations in the memory units themselves, thereby reducing the data movement between the memory and the processing units [1]. Analog computing, on the other hand, aims to perform computations in a single step, using all of the available voltage range, thereby reducing the energy consumption and latency of the computation [2].

The main structural component of multilevel MVMs is a *resistive-switching memory* (RSM) device [3]. Arranged as a crossbar, these devices store the entries of the multiplication matrix and take simultaneously part in the multiplication operation, thereby avoiding the bottleneck of data movement between memory and processing units. Furthermore, this operation is performed with an analog circuit in a single step, further reducing the energy consumption, latency, and area requirement of the computation [4].

By combining IMC and analog computing paradigms, multilevel MVMs provide a promising solution for the challenges faced by *artificial neural networks* (ANNs) and internet-of-things (IoT) devices.

ANNs play an essential role in the era of emerging AI applications such as large language models, visual recognition, image generation, and autonomy. In light of this expansion and the need for ever-increasing capabilities, the parameters of ANNs are increasing dramatically. However, the separation of processing and data units in the von Neumann architecture creates a bottleneck for these data-intensive ANNs, which causes significant latency and excess energy consumption.

The IMC paradigm has been used to alleviate this bottleneck, resulting in lower latency and power consumption [1]. However, analog and multilevel MVMs have the advantage of being both IMC and using analog computing for the matrix-vector multiplications that ANNs heavily rely on [2].

Similarly, IoT devices are becoming increasingly integral to our daily lives, with applications ranging from smart homes to smart cities. These devices must be energy-efficient, as they are often battery-powered and are expected to operate for long periods without human intervention.

Matrix-vector multiplications are a common operation in many IoT applications, such as signal processing and linear algebra operations. Furthermore, data-intensive applications such as inferencing on ANNs are embedded in IoT devices due to the increasing demand for edge computing. Multilevel MVMs provide a significant reduction in energy consumption and latency for these applications.

Offsetting these advantages, a major challenge in multilevel MVMs is errors caused by nonidealities including process variations, parasitic resistances, device nonlinearity, imprecise conductance adjustments, device aging, and inaccuracies in *digital-to-analog converters* (DACs) and *analog-to-digital converters* (ADCs) [3], [4]. While ANNs can tolerate a degree

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project ECXL (grant no. 01IW22002) as well as by the German Research Foundation (DFG) within the Projects PLiM (DR 287/35-1, DR 287/35-2) and Reinhart Koselleck Project PolyVer (DR 287/36-1)

of imprecision [5], this tolerance has its limits. To make these systems practical, several attempts have been made to reduce the error of multilevel MVMs [6], including optimization of weight-to-conductance mappings [7]–[9], retraining of ANN weights [10], [11], usage of ANN architecture with error suppression and compensation [12], usage of committee machines [13], and the use of redundant neurons and synapses [14].

Similarly, measuring the error of multilevel MVMs has also been a subject of focus, with approaches based on simulations [15], emulations [16], and testing [17]. However, finding averages, stochastic distributions of errors, or equivalence errors [18] is not sufficient for safety-critical applications such as autonomous vehicles, medical diagnosis, and control systems, where MVM errors could lead to catastrophic consequences. For such applications, the error of utilized MVM devices must be guaranteed to stay below the safety margin of the application. Formally verifying the error bounds could provide the necessary guarantees for these applications, as evidenced by the increased interest in formally verified approximation error (called *relaxed equivalence checking*) in the field of approximate computing [19], [20].

Contribution: In this paper, we aim to address this need for guaranteed error bounds in resistive-switching-based multilevel MVMs, with what we believe to be the first formal verification methodology in this direction. We present three approaches to compute the maximum error and introduce a polynomial-time solution as our primary contribution. Furthermore, we also provide a tracing feature to identify the sources of errors. We demonstrate the efficiency of our methodology with a timing analysis and its effectiveness with a case study using the metrics of a fabricated multilevel MVM from the literature [21].

The next section summarizes the necessary background information relevant to this work and introduces notation. Section III presents three formal verification approaches, the tracing capability, as well as an overview of our software implementation. In Section IV we present the results of our experiments and conclude the paper with Section V.

II. Preliminaries

In this section we give some background information, define the problem, and introduce notation. We first conceptually describe our formal verification approach, then introduce multilevel MVMs, and finally describe the source of errors in these devices.

A. Analog Formal Verification

Before delving into the intricacies of the methodology, it is useful to mention the differences between the formal verification in this study and the more familiar digital formal verification. Digital formal verification [22], [23], which becomes increasingly applicable thanks to advances in polynomial methods [24], [25], focuses on verifying the structure of hardware designs, such as logic gates at the gate level and hardware description code at the register-transfer level. In contrast, analyzing the circuit structure of the crossbar in Figure 1, as discussed in the final paragraph of Section II-B, easily reveals that it performs a matrix-vector multiplication.

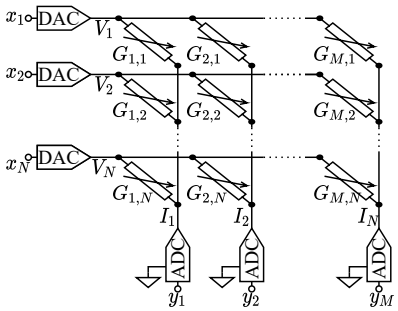

Fig. 1: Schematic of a Multilevel Matrix-Vector Multiplier

However, the analog nature of the computation introduces unwanted inaccuracies due to variances and rigid design parameters. These inaccuracies lead to erroneous computation results.

The methodology presented here uses the analog computation's properties to formally verify that errors are either zero or remain within a certain bound.

B. Multilevel Matrix-Vector Multipliers

Multilevel MVMs using resistive-switching devices have the structure in Figure 1. The variable resistors represent 2- and 3-terminal RSM devices such as *resistive random access memory* (RRAM), *phase change memory* (PCM), *ferroelectric field-effect transistor* (FeFET), and *floating-gate field-effect transistor* (FGFET). When 3-terminal devices are used, extra row or column connections are required to operate the resistances. However, for the purpose of assessing functional correctness, these connections can be ignored, and 2-terminal devices can be assumed.

The purpose of this structure is to execute the operation $\vec{y} = W\vec{x}$, where W is an $M \times N$ sized matrix, with M number of *multiply-accumulate* (MAC) operations of the form

$$y_i = \vec{w_i} \cdot \vec{x} = \sum_{j=1}^{N} w_{ij} x_j \tag{1}$$

where $\vec{w_i} = [w_{i1}, w_{i2}, \ldots, w_{iN}]$ and $i = 1..M$.

This MAC operation (hereinafter called the symbolic MAC operation) is realized as an analog computation by loading the weights w_{ij} into the variable conductances G_{ij} and the input signals x_j into the DACs to produce the analog voltages V_j. The analog counterpart to the symbolic MAC operation, $I_i = \sum_{j=1}^{N} G_{ij} V_j$, is then realized with the columns in Figure 1 using Ohm's law ($I_{G_{ij}} = G_{ij} V_{G_{ij}}$) for multiplication (notice $V_j = V_{G_{ij}}$ in Figure 1, as all the RSMs in the same row share the same voltage) and Kirchhoff's current law ($I_i = \sum_{j=1}^{N} I_{G_{ij}}$) for summation. The output of the MAC operation, y_i, is then obtained by converting the currents I_i using current-input ADCs. We denote the mappings between the symbolic and analog variables with the functions $G_{ij} = f_G(w_{ij})$, $V_j = f_V(x_j)$, and $y_i = f_y(I_i)$. The variables w_{ij} and x_j are multilevel, as discussed in the next section.

C. Errors in Multilevel Matrix-Vector Multipliers

Ideally, if the functions f_G, f_V, and f_y were identity functions, the crossbar would operate correctly and without errors. However, employing identity functions is impractical in terms of power consumption, and is problematic in terms of physical constraints. Furthermore, the f_G function cannot be practically realized as an identity function due to inaccuracies in the conductances such as process variations, environmental factors, and device aging. This issue is mitigated by limiting the MAC operation to a finite number of levels (hence, *multilevel*) and adjusting f_V and f_y accordingly. Therefore, the entries of W and $\vec{x}$ are given by $w_{ij} \in \{0, 1, ..., w_{\max}\}$ and $x_j \in \{0, 1, ..., x_{\max}\}$.

When using this approach, the computation's range and precision is not necessarily limited to $w_{\max}$ and $x_{\max}$, since the MVM can be enhanced by employing a positional number system using multiple columns for a single MAC operation [26]. The presented methodology is applicable to this extension as is, since it verifies the MVMs in a column-by-column fashion.

The functions f_G, f_V, and f_y are designed by considering device- and process-specific limitations and variations. Since our verification methodology applies to all possible realizations of these functions, we do not delve further into the details of their realizations and point the reader to the reviews on the subject [4], [27].

It is however important to mention that even though the functions f_G and f_V are precisely designed, the variations in the RSM devices and the DACs can cause the functions to differ from their intended values. This can create a discrepancy between the analog and symbolic counterparts of the MAC operation and can cause errors. We denote the intended functions with $f_{G_{\text{nom}}}$ and $f_{V_{\text{nom}}}$, the set of all possible functions with F_G and F_V, and the minimum and maximum values of the functions with $f_{G_{\min}}$, $f_{G_{\max}}$, $f_{V_{\min}}$, and $f_{V_{\max}}$.

On a side note, even if the functions f_G and f_V were precise (i.e. voltages and conductances were set with absolute precision), an erroneous design of these functions can also cause errors, a situation that we illustrate in the case study given in Section IV-B.

The goal of the methodology introduced in the next section is to find the maximum error caused by the discussed discrepancies,

$$\delta_{y_i,\max} = \max_{\substack{\vec{w_i},\vec{x} \\ f_G,f_V}} \left(\left| \sum_{j=1}^{N} w_{ij}x_j - f_y\left(\sum_{j=1}^{N} f_G(w_{ij}) f_V(x_j) \right) \right| \right) \tag{2}$$

where the search space variables are constrained to their respective sets $\vec{w_i} \in \mathbb{Z}^N_{[0,w_{\max}]}$, $\vec{x} \in \mathbb{Z}^N_{[0,x_{\max}]}$, $f_G \in F_G$, and $f_V \in F_V$. In the rest of the paper, we refrain from explicitly writing the sets of these variables for brevity.

III. Methodology

In this section, we present our formal verification methodology that finds the maximum error for resistive-switching-based multilevel MVMs. We begin by presenting three approaches for computing $\delta_{y_i,\max}$ in Sections III-A, III-B, and III-C. Section III-A, introduces the most basic solution, and is a prerequisite for the other two. Section III-B introduces a symmetry-based approach that is more efficient than the basic solution, but still has an exponential time complexity. Section III-C introduces the solution with polynomial time complexity, our primary contribution. Afterwards, we describe the tracing feature of the methodology in Section III-D. Finally, Section III-E, provides an overview of our software implementation and presents the algorithm that drives the polynomial-time solution given in Section III-C.

A. Transformation for Computability

The maximization in (2) is not computable because F_V and F_G contain infinitely many elements, as they represent real-world, continuous variances. In this section, we remove f_V and f_G from the search space to transform (2) into a computable form, where $\delta_{y_i,\max}$ is computed separately for all columns of the crossbar array (for all $i = 1..M$).

We start with two key properties. First, both f_V and f_G are independent of $\vec{w_i}$ and $\vec{x}$. Second, for any given $\vec{w_i}$ and $\vec{x}$, the expression to the left of the minus sign in (2) is a constant, while the expression to the right is a function of f_V and f_G. Since the absolute difference between a variable and a constant is maximum when the variable is either at its minimum or maximum value, we get

$$\delta_{y_i,\max} = \max(\delta_{y_i,\max_{\min}}, \delta_{y_i,\max_{\max}}) \tag{3}$$

where

$$\delta_{y_i,\max_{\min}} = \max_{\vec{w_i},\vec{x}} \left(\left| \sum_{j=1}^{N} w_{ij}x_j - \min_{f_G,f_V} \left(f_y\left(\sum_{j=1}^{N} f_G(w_{ij}) f_V(x_j) \right) \right) \right| \right)$$

$$\delta_{y_i,\max_{\max}} = \max_{\vec{w_i},\vec{x}} \left(\left| \sum_{j=1}^{N} w_{ij}x_j - \max_{f_G,f_V} \left(f_y\left(\sum_{j=1}^{N} f_G(w_{ij}) f_V(x_j) \right) \right) \right| \right)$$

Given that calculating $\delta_{y_i,\max_{\min}}$ and $\delta_{y_i,\max_{\max}}$ is very similar, we only focus on the latter for brevity in the rest of the paper.

The association between analog and symbolic MAC operations suggests that the output function f_y is designed such that an increasing current at the output of the crossbar column corresponds to a non-decreasing symbolic output. Therefore, we can safely assume that the function f_y is non-decreasing, and we get

$$\delta_{y_i,\max_{\max}} = \max_{\vec{w_i},\vec{x}} \left(\left| \sum_{j=1}^{N} w_{ij}x_j - f_y\left(\max_{f_G,f_V} \left(\sum_{j=1}^{N} f_G(w_{ij}) f_V(x_j) \right) \right) \right| \right)$$

Furthermore, since the functions f_G and f_V only produce non-negative values, all of the terms in the summation are individually maximized for $f_G = f_{G_{\max}}$ and $f_V = f_{V_{\max}}$, and we get

$$\delta_{y_i,\max_{\max}} = \max_{\vec{w_i},\vec{x}} \left(\left| \sum_{j=1}^{N} w_{ij}x_j - f_y\left(\sum_{j=1}^{N} f_{G_{\max}}(w_{ij}) f_{V_{\max}}(x_j) \right) \right| \right) \tag{4}$$

The search space of the maximization in (4) is the Cartesian product of the state spaces of $\vec{w}_i$ and $\vec{x}$, denoted as $\mathbb{Z}^N_{[0,w_{\max}]}$ and $\mathbb{Z}^N_{[0,x_{\max}]}$, respectively. The number of elements in the search space is therefore $(w_{\max}+1)^N(x_{\max}+1)^N$.

Definition 1. *Brute Force Algorithm (BFA):* An algorithm that exhaustively searches the entire space of the maximization in (4) to compute $\delta_{y_i,\max_{\max}}$ (and similarly $\delta_{y_i,\max_{\min}}$) and finds $\delta_{y_i,\max}$ with (3). The time complexity of this algorithm is $\mathcal{O}(w^N_{\max}x^N_{\max})$.

B. Search Space Reduction through Symmetry

Notice that in (4), the order of the elements in $\vec{w}_i$ and $\vec{x}$ does not affect the outcome, provided their relative orders remain unchanged. This is attributed to the commutative and associative properties of the summation operators. Therefore, the search space of the maximization in (4) can be reduced from permutations of the elements to only their combinations.

Since the relative orders in $\vec{w}_i$ and $\vec{x}$ must be preserved, we introduce an abstraction to $\vec{w}_i$ and $\vec{x}$ as follows:

$$\delta_{y_i,\max_{\max}} = \max_{\vec{p}\in P^N}\left(\left|\sum_{j=1}^{N} p_j - f_y\left(\sum_{j=1}^{N} f_{I_{\max},p_j}(p_j)\right)\right|\right) \quad (5)$$

where P is a set containing all possible $w_{ij}x_j$ values, and

$$f_{I_{\max},p_j}(p_j) = \max\left(\left\{f_{G_{\max}}(w_{ij})f_{V_{\max}}(x_j) : w_{ij}x_j = p_j\right\}\right) \quad (6)$$

In the reformulation of (5), the order of vector $\vec{p}$ does not impact the value of $\delta_{y_i,\max_{\max}}$. To consider only a unique permutation, we mandate that the elements of $\vec{p}$ form a non-decreasing sequence ($p_i \le p_j$ for $i<j$), and we get

$$\delta_{y_i,\max_{\max}} = \max_{\vec{p}\in P^N_{\text{ndec}}}\left(\left|\sum_{j=1}^{N} p_j - f_y\left(\sum_{j=1}^{N} f_{I_{\max},p_j}(p_j)\right)\right|\right) \quad (7)$$

where

$$P^N_{\text{ndec}} = \left\{\vec{p}\in P^N : p_i \le p_j \text{ for } i<j\right\}$$

Definition 2. *Symmetry-Based Algorithm (SBA):* An algorithm that uses (7) to compute $\delta_{y_i,\max_{\max}}$ (and similarly $\delta_{y_i,\max_{\min}}$) and finds $\delta_{y_i,\max}$ with (3).

C. Recursive Decomposition and Dynamic Programming

In the following, we show that the task of finding $\delta_{y_i,\max_{\max}}$ can be divided into smaller subproblems and that a polynomial time complexity can be achieved with dynamic programming.

Before a recursive decomposition can be achieved, we must redefine the maximization problem expressed in (4) in terms of the symbolic output value y_i:

$$\delta_{y_i,\max_{\max}} = \max_{y_i\in S_y(N)}\left(\left|y_i - f_y\big(f_{I_{\max},y_i}(y_i,N)\big)\right|\right) \quad (8)$$

where

$$S_y(N) = \left\{\vec{w}_i\cdot\vec{x} : \left(\vec{w}_i \in \mathbb{Z}^N_{[0,w_{\max}]}\right) \wedge \left(\vec{x}\in\mathbb{Z}^N_{[0,x_{\max}]}\right)\right\} \quad (9)$$

and

$$f_{I_{\max},y_i}(y_i,N) = \max\left(\left\{\sum_{j=1}^{N} f_{G_{\max}}(w_{ij})f_{V_{\max}}(x_j) : \big(\vec{w}_i\cdot\vec{x}=y_i\big)\right\}\right) \quad (10)$$

To see that this is a valid reformulation, notice that the set $S_y(N)$ includes all possible $\vec{w}_i\cdot\vec{x}$ values. Furthermore, the maximization in (8) is done over all $\vec{w}_i$ and $\vec{x}$ values, as in (4). To see this, notice that for all $\vec{w}_i$ and $\vec{x}$ values, there exists a $y_i\in S_y(N)$ such that the constraint $\vec{w}_i\cdot\vec{x}=y_i$ in the predicate of the set-builder notation in (10) is satisfied.

At this point, we focus on calculating $f_{I_{\max},y_i}(y_i,N)$. The error $\delta_{y_i,\max_{\max}}$ is then found by iterating over all $y_i\in S_y(N)$. Calculating $f_{I_{\max},y_i}(y_i,N)$ can be divided into smaller subproblems by using the following theorem:

Theorem 1. *For any $n\in\mathbb{N}_{[2,N]}$ and $y\in S_y(n)\setminus\{0,1\}$, we have*

$$\begin{aligned} f_{I_{\max},y_i}(y,n) = \max\big(\big\{ & f_{I_{\max},y_i}(y_{\text{sub}},n_{\text{sub}}) + f_{I_{\max},y_i}(y-y_{\text{sub}},n-n_{\text{sub}}) \\ & : (n_{\text{sub}}\in\mathbb{N}^+_{\le n-1}) \wedge (y_{\text{sub}}\ge y/2) \\ & \wedge (y_{\text{sub}}\in S_y(n_{\text{sub}})) \wedge (y-y_{\text{sub}}\in S_y(n-n_{\text{sub}}))\big\}\big) \end{aligned} \quad (11)$$

The four conditions in the predicate of (11) are explained in the proof below.

Proof. We start from the definition of $f_{I_{\max},y_i}(y_i,N)$ given in (10). We have

$$\begin{aligned} f_{I_{\max},y_i}(y,n) &= \max\left(\left\{\sum_{j=1}^{n} f_{G_{\max}}(w_{ij})f_{V_{\max}}(x_j) : \big(\vec{w}_i\cdot\vec{x}=y\big)\right\}\right) \\ &= \max\Bigg(\Bigg\{\sum_{j\in S_1} f_{G_{\max}}(w_{ij})f_{V_{\max}}(x_j) + \sum_{j\in S_2} f_{G_{\max}}(w_{ij})f_{V_{\max}}(x_j) \\ &\qquad : \left(\sum_{j\in S_1} w_{ij}x_j + \sum_{j\in S_2} w_{ij}x_j = y\right) \\ &\qquad \wedge (S_1\sqcup S_2 = \mathbb{N}^+_{\le n}) \wedge (|S_1|\in\mathbb{N}^+_{\le n-1}) \wedge (y_{\text{sub}}\ge y/2) \\ &\qquad \wedge (y_{\text{sub}}\in S_y(|S_1|)) \wedge (y-y_{\text{sub}}\in S_y(|S_2|))\Bigg\}\Bigg) \end{aligned}$$

where $y_{\text{sub}} = \sum_{j\in S_1} w_{ij}x_j$.

Here we have grouped the $f_{G_{\max}}(w_{ij})f_{V_{\max}}(x_j)$ terms into two disjoint, nonempty partitions and expanded the search space to include all feasible partitions S_1 and S_2, in accordance with the constraints. The constraint that the $w_{ij}x_j$ terms sum to y is still present, the constraint $S_1\sqcup S_2=\mathbb{N}^+_{\le n}$ enforces a disjoint partition, and the constraint $|S_1|\in\mathbb{N}^+_{\le n-1}$ enforces that neither S_1 nor S_2 is empty.

Furthermore, since the $w_{ij}x_j$ terms of S_1 and S_2 collectively add up to y, one of them must add up to at least $y/2$. We designate S_1 as this set with the constraint $y_{\text{sub}}\ge y/2$ to prevent double checking identical partitions.

The statements $(y_{\text{sub}}\in S_y(|S_1|))$ and $(y-y_{\text{sub}}\in S_y(|S_2|))$ are always true, since $S_y(|S_1|)$ and $S_y(|S_2|)$ include every possible $\sum_{j=1}^{|S_1|} w_{ij}x_j$ and $\sum_{j=1}^{|S_2|} w_{ij}x_j$ value, respectively (see (9)). Therefore, they do not constrain the set-builder.

Note that the total sums of $f_{G_{\max}}(w_{ij})f_{V_{\max}}(x_j)$ within each set, S_1 and S_2, are independent. Since for independent terms the sum is maximized when each individual term is maximized, we have

$$\begin{aligned} f_{I_{\max},y_i}(y,n) = \max\Bigg(\Big\{ & \\ & \max\Big(\Big\{\sum_{j\in S_1} f_{G_{\max}}(w_{ij})f_{V_{\max}}(x_j) : \Big(\sum_{j\in S_1} w_{ij}x_j = y_{\text{sub}}\Big)\Big\}\Big) \\ & + \max\Big(\Big\{\sum_{j\in S_2} f_{G_{\max}}(w_{ij})f_{V_{\max}}(x_j) : \Big(\sum_{j\in S_2} w_{ij}x_j = y - y_{\text{sub}}\Big)\Big\}\Big) \\ & : (S_1 \sqcup S_2 = \mathbb{N}^+_{\leq n}) \wedge (|S_1| \in \mathbb{N}^+_{\leq n-1}) \wedge (y_{\text{sub}} \geq y/2) \\ & \wedge (y_{\text{sub}} \in S_y(|S_1|)) \wedge (y - y_{\text{sub}} \in S_y(|S_2|))\Big\}\Bigg) \end{aligned}$$

We then change the variable names of w_{ij} and x_j, and denote $|S_1|$ as n_{sub} (implying $|S_2| = n - n_{\text{sub}}$) to get

$$\begin{aligned} f_{I_{\max},y_i}(y,n) = \max\Bigg(\Big\{ & \\ & \max\Big(\Big\{\sum_{j=1}^{n_{\text{sub}}} f_{G_{\max}}(w_{ij})f_{V_{\max}}(x_j) : \Big(\sum_{j=1}^{n_{\text{sub}}} w_{ij}x_j = y_{\text{sub}}\Big)\Big\}\Big) \\ & + \max\Big(\Big\{\sum_{j=1}^{n-n_{\text{sub}}} f_{G_{\max}}(w_{ij})f_{V_{\max}}(x_j) : \Big(\sum_{j=1}^{n-n_{\text{sub}}} w_{ij}x_j = y - y_{\text{sub}}\Big)\Big\}\Big) \\ & : (n_{\text{sub}} \in \mathbb{N}^+_{\leq n-1}) \wedge (y_{\text{sub}} \geq y/2) \\ & \wedge (y_{\text{sub}} \in S_y(n_{\text{sub}})) \wedge (y - y_{\text{sub}} \in S_y(n - n_{\text{sub}}))\Big\}\Bigg) \end{aligned}$$

which, per the definition of $f_{I_{\max},y_i}(y,n)$ in (10), simplifies to (11). □

The recursive relation in (11) is complete with the base cases

$$f_{I_{\max},y_i}(0,n) = n \cdot f_{I_{\max},p_j}(0) \qquad \forall\, n \in \mathbb{N}^+_{\leq N} \tag{12a}$$

$$f_{I_{\max},y_i}(1,n) = (n-1)f_{I_{\max},p_j}(0) + f_{I_{\max},p_j}(1) \qquad \forall\, n \in \mathbb{N}^+_{\leq N} \tag{12b}$$

$$f_{I_{\max},y_i}(y,1) = f_{I_{\max},p_j}(y) \qquad \forall\, y \in S_y(1) \tag{12c}$$

where $f_{I_{\max},p_j}$ is defined as in (6).

Calculating $f_{I_{\max},y_i}(y_i,N)$ with the recursive relation in (11), along with the base cases, still leads to exponential time complexity. However, we can use dynamic programming to calculate $f_{I_{\max},y_i}(y_i,N)$ in polynomial time as follows:

Definition 3. *Dynamic-Programming-Based Alg. (DPBA):* An algorithm that computes $\delta_{y_i,\max_{\max}}$ by finds $f_{I_{\max},y_i}(y_i,N)$ for all $y_i \in S_y(N)$ and using Equation (8). It stores $f_{I_{\max},y_i}(y,n)$ as a table with $(\max(S_y(N))+1)$ rows and N columns. The table is populated in a bottom-up manner, starting with the base cases in (12), and using (11) for the rest of the table. It also similarly calculates $\delta_{y_i,\max_{\min}}$ and finds $\delta_{y_i,\max}$ with (3). The time complexity of this algorithm will be analyzed in Section III-E.

D. Trace Generation

Generating a trace to pinpoint when the maximum error occurs is crucial since it allows the precise identification of the worst-case scenario, facilitating targeted design improvements. We mark the values that result in $\delta_{y_i,\max_{\max}}$ with a superscript asterisk (*).

Since BFA iterates over $\vec{w}_i$ and $\vec{x}$, finding $\vec{w}_i{}^*$ and $\vec{x}^*$ is straightforward. However, for SBA and DPBA, these values are not directly calculated.

Since SBA iterates over $\vec{p}$, we get $\vec{p}^*$. We then find $\vec{w}_i{}^*$ and $\vec{x}^*$ by reversing the abstraction in (6):

$$\begin{matrix} \vec{w}_i{}^* = [w^*_{i1}, \ldots, w^*_{iN}] \\ \vec{x}^* = [x^*_1, \ldots, x^*_N] \end{matrix} : (w^*_{ij}, x^*_j) = f_{w^*_{ij},x^*_j}(p^*_j) \text{ for } j \in \mathbb{N}^+_{\leq N} \tag{13}$$

where

$$f_{w^*_{ij},x^*_j}(p^*_j) = \underset{w_{ij},x_j}{\operatorname{argmax}}\Big(\big\{f_{G_{\max}}(w_{ij})f_{V_{\max}}(x_j) : w_{ij}x_j = p^*_j\big\}\Big)$$

For DPBA, $\vec{w}_i{}^*$ and $\vec{x}^*$ can be found by analyzing the subproblems. Since we will need these values for the subproblems too, we generalize the notation to $\vec{w}_i{}^*(y,n)$ and $\vec{x}^*(y,n)$ for the maximization of $f_{I_{\max},y_i}(y,n)$. Vectors $\vec{w}_i{}^*$ and $\vec{x}^*$ are then found by

$$\begin{aligned} \vec{w}_i{}^* &= \vec{w}_i{}^*(y^*,N) \\ \vec{x}^* &= \vec{x}^*(y^*,N) \end{aligned} \tag{14}$$

where y^* is the y value that maximizes $f_{I_{\max},y_i}(y,N)$.

On the other hand, for all $n \in \mathbb{N}_{[2,N]}$ and $y \in S_y(n)\backslash\{0,1\}$, we find $\vec{w}_i{}^*(y,n)$ and $\vec{x}^*(y,n)$ from the subproblems by

$$\begin{aligned} \vec{w}_i{}^*(y,n) &= \Big[\vec{w}_i{}^*_{y^*_{\text{sub}}(y,n),n^*_{\text{sub}}(y,n)}, \vec{w}_i{}^*_{y-y^*_{\text{sub}}(y,n),n-n^*_{\text{sub}}(y,n)}\Big] \\ \vec{x}^*(y,n) &= \Big[\vec{x}^*_{y^*_{\text{sub}}(y,n),n^*_{\text{sub}}(y,n)}, \vec{x}^*_{y-y^*_{\text{sub}}(y,n),n-n^*_{\text{sub}}(y,n)}\Big] \end{aligned} \tag{15}$$

for $y^*_{\text{sub}}(y,n)$ and $n^*_{\text{sub}}(y,n)$ that maximize $f_{I_{\max},y_i}(y,n)$.

Since this is defined only for $y \geq 2$ and $n \geq 2$, we need to give the base cases $\vec{w}_i{}^*(0,n)$, $\vec{x}^*(0,n)$, $\vec{w}_i{}^*(1,n)$, and $\vec{x}^*(1,n)$ for $n \in \mathbb{N}^+_{\leq N}$. We also need to give the base cases $\vec{w}_i{}^*(y,1)$ and $\vec{x}^*(y,1)$ for $y \in S_y(1)$. We find these by first finding $\vec{p}^*(0,n)$, $\vec{p}^*(1,n)$, and $\vec{p}^*(y,1)$ as follows:

$$\vec{p}^*(0,n) = \underset{1\times n}{\vec{0}} \qquad \forall\, n \in \mathbb{N}^+_{\leq N} \tag{16a}$$

$$\vec{p}^*(1,n) = \Big[\underset{1\times n_l-1}{\vec{0}}, \quad 1\Big] \qquad \forall\, n \in \mathbb{N}^+_{\leq N} \tag{16b}$$

$$\vec{p}^*(y,1) = [y] \qquad \forall\, y \in S_y(1) \tag{16c}$$

and then finding the respective $\vec{w}$ and $\vec{x}$ vectors with (13).

The traces $\vec{w}_i{}^*$ and $\vec{x}^*$ can be used to make targeted design improvements since they pinpoint the worst-case scenario. We have demonstrated this use case in the case study given in Section IV-B.

E. Implementation

We have developed a Python package[1] incorporating BFA, SBA, and DPBA along with additional functions, as illustrated

[1] https://github.com/KCaglarCoskun/multi-level-mvm-verification

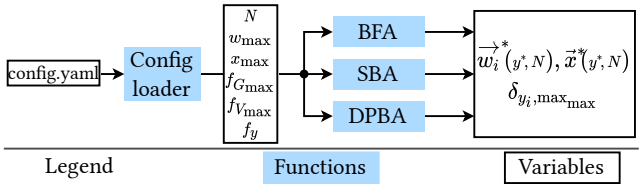

Fig. 2: Block diagram overview of the implementation.

1: $S_y(1) \leftarrow \{w_{ij}x_j : w_{ij} \in \mathbb{Z}_{[0,w_{\max}]} \wedge x_j \in \mathbb{Z}_{[0,x_{\max}]}\}$ ▷ See (9)
2: $f_{I_{\max},p_j}(\cdot) \leftarrow$ (6)$(f_{G_{\max}}, f_{V_{\max}})$
3: $f_{I_{\max},y_i}(\cdot,\cdot) \leftarrow$ Initialize table with $-\infty$
4: $(\vec{w}_i{}^*(\cdot,\cdot), \vec{x}^*(\cdot,\cdot)) \leftarrow$ Initialize empty table
5: $(y^*_{\text{sub}}(\cdot,\cdot), n^*_{\text{sub}}(\cdot,\cdot)) \leftarrow$ Initialize with zeros
6: $f_{I_{\max},y_i}(0,n) \leftarrow$ (12a)$(\, f_{I_{\max},p_j}(\cdot)\,)$
7: $f_{I_{\max},y_i}(1,n) \leftarrow$ (12b)$(\, f_{I_{\max},p_j}(\cdot)\,)$
8: $f_{I_{\max},y_i}(y,1) \leftarrow$ (12c)$(\, f_{I_{\max},p_j}(\cdot)\,, S_y(1)\,)$
9: $(\vec{w}_i{}^*(0,n), \vec{x}^*(0,n)) \leftarrow$ (13)$(\,$(16a)$\,)$
10: $(\vec{w}_i{}^*(1,n), \vec{x}^*(1,n)) \leftarrow$ (13)$(\,$(16b)$\,)$
11: $(\vec{w}_i{}^*(y,1), \vec{x}^*(y,1)) \leftarrow$ (13)$(\,$(16c)$(\, S_y(1)\,)\,)$
12: **for** $n \in \mathbb{N}_{[2,N]}$ ▷ See Theorem 1 for lines 12–17
13: **for** $y \in \mathbb{N}_{[2,y_{\max}(n)]}$
14: **for** $n_{\text{sub}} \in \mathbb{N}^+_{<n-1}$
15: **for** $y_{\text{sub}} \in \bar{\mathbb{N}}_{[y/2,y]}$
16: **if** $f_{I_{\max},y_i}(y,n) < f_{I_{\max},y_i}(y_{\text{sub}},n_{\text{sub}}) + f_{I_{\max},y_i}(y-y_{\text{sub}},n-n_{\text{sub}})$
17: $f_{I_{\max},y_i}(y,n) \leftarrow f_{I_{\max},y_i}(y_{\text{sub}},n_{\text{sub}}) + f_{I_{\max},y_i}(y-y_{\text{sub}},n-n_{\text{sub}})$
18: $y^*_{\text{sub}}(y,n), n^*_{\text{sub}}(y,n) \leftarrow y_{\text{sub}}, n_{\text{sub}}$ ▷ See (15)
19: **if** $n = N$
20: $S_y(N) \leftarrow S_y(N) \cup \{y\}$
21: $\delta_{y_i,\max_{\max}}, y^* \leftarrow$ (8)$(S_y(N), f_y(\cdot), f_{I_{\max},y_i}(\cdot,N))$
22: TRACE(y^*, N) ▷ Fills $\vec{w}_i{}^*(y^*,N)$ and $\vec{x}^*(y^*,N)$

• For clarity we assume that $\vec{w}_i{}^*(\cdot,\cdot)$, $\vec{x}^*(\cdot,\cdot)$, $y^*_{\text{sub}}(\cdot,\cdot)$, $n^*_{\text{sub}}(\cdot,\cdot)$, N, $w_{\max}$, and $x_{\max}$, are globally available to all functions in the pseudocode.
• At Line 16, we consider $-\infty < -\infty$ to be false.

Algorithm 1: Dynamic-Programming-Based Algorithm

by the simplified block diagram in Figure 2. For clarity, we have omitted some utility functions from the diagram, such as a timer function used in the timing analysis, and only show the parameters and variables relevant to calculating $\delta_{y_i,\max_{\max}}$, as we described the calculation of only $\delta_{y_i,\max_{\max}}$ in Section III. Nevertheless, the implementation also calculates $\delta_{y_i,\max_{\min}}$ by mirroring the algorithms and finds $\delta_{y_i,\max}$ with (3).

DPBA returns the same results as BFA and SBA with lower time complexity and is our main contribution. Therefore, and for the sake of brevity, we omit the pseudocode for BFA and SBA, presenting only DPBA's pseudocode in Algorithm 1. As we have done in Section III-E, we present the relevant pseudocode only for calculating $\delta_{y_i,\max_{\max}}$, since finding $\delta_{y_i,\max_{\min}}$ is a dual problem.

At the beginning of the algorithm, some values that will be needed later are calculated at lines 1 and 2, and the dynamic

1: **function** TRACE(y,n)
2: **if** $\vec{w}_i{}^*(y,n)$ and $\vec{x}^*(y,n)$ are already calculated
3: **return**
4: TRACE$(y^*_{\text{sub}}(y,n), n^*_{\text{sub}}(y,n))$
5: TRACE$(y - y^*_{\text{sub}}(y,n), n - n^*_{\text{sub}}(y,n))$
6: $(\vec{w}_i{}^*(y,n), \vec{x}^*(y,n)) \leftarrow$ (15)
7: **return**

Algorithm 2: Trace Generation Function

programming tables are initialized at lines 3–5. Then, the base cases are calculated at lines 6–11. The table $f_{I_{\max},y_i}(\cdot,\cdot)$ is filled in a bottom-up manner at lines 12–17, using the recursive relation in Theorem 1. Line 18 is necessary for efficient tracing and lines 19 and 20 are needed since $S_y(N)$ is not known in advance. Finally, $\delta_{y_i,\max_{\max}}$ is calculated at line 21 and the trace is generated at line 22.

Note that, in Line 13 we should have $y \in S_y(n)\backslash\{0,1\}$ as seen in Theorem 1. However, we utilize the set $\mathbb{N}_{[2,y_{\max}(n)]}$ instead, with $y_{\max}(n)$ defined as $nw_{\max}x_{\max}$. This is because the set $S_y(n)$ is not known in advance, and the set $\mathbb{N}_{[2,y_{\max}(n)]}$, which is a superset of $S_y(n)\backslash\{0,1\}$, is used to ensure that no y values are omitted. Any y that is not in $S_y(n)$ will result in $f_{I_{\max},y_i}(y,n) = -\infty$, and will not affect the final result. The same reasoning applies to Line 15. Furthermore, since for any $y \in S_y(N)$, $f_{I_{\max},y_i}(y,N)$ is bigger than $-\infty$, $S_y(N)$ can be generated with Line 20.

The TRACE function, as given in Algorithm 2, implements the recursion given in (15) to find $\vec{w}_i{}^*(y^*,N)$ and $\vec{x}^*(y^*,N)$ by calculating only the necessary cells of the tables $\vec{w}_i{}^*(\cdot,\cdot)$ and $\vec{x}^*(\cdot,\cdot)$.

The time complexity of DPBA is the sum of the time complexity of the nested loops at lines 12–20 and the time complexity of the call to TRACE in line 22. Every operation inside the for loops (Lines 16 to 20) is constant time, therefore the time complexity of the nested loops is $O(Ny_{\max}(N)Ny_{\max}(N)) = O(N^2y_{\max}(N)^2) = O(N^2(Nw_{\max}x_{\max})^2) = O(N^4w^2_{\max}x^2_{\max})$.

To analyze the time complexity of the call to TRACE in Algorithm 1, at line 22, we find a maximum bound to the recursion depth and to the number of TRACE instances at any depth d. The recursion depth is capped at $d \leq N$, since the number of rows decreases with every recursion ($n_{\text{sub}} < n$). To find a bound to the number of processes at any depth d, notice that the local n values of all TRACE processes at the same depth add up to N, since the local n values of the two calls at lines 4 and 5 sum up to the n value of the parent caller. Since n is minimally 1, the maximum number of TRACE processes per level is N. Therefore, the total number of calls to TRACE is at most N^2. Since the concatenation and copy operations in line 6 take $O(N)$ time, the total time complexity of the call to TRACE by DPBA is $O(N^3)$.

The time complexity of DPBA is therefore

$$O(N^4w^2_{\max}x^2_{\max} + N^3) = O(N^4w^2_{\max}x^2_{\max})$$

In the next section, we will conduct experiments with the

TABLE I: Results of the Timing Analysis

N	$w_{\max}$	$x_{\max}$	Runtime BFA	Runtime SBA	Runtime DPBA	Speedup[a]	$\delta_{y_i,\max}$
10	3	3	1751 s	0.08 s	0.09 s	-	1.81
20	3	3	-[b]	2.05 s	1.36 s	1.5×	3.62
40	3	3	-[b]	204 s	20 s	10×	7.24
80	3	3	-[b]	3.23 h	331 s	35×	14.47
10	7	7	-[b]	1043 s	2.55 s	409×	9.85
20	7	7	-[b]	-[b]	39.6 s	>2181×	19.7
40	7	7	-[b]	-[b]	638 s	-	39.4

[a] Speedup from SBA to DPBA
[b] Terminated after exceeding 24 hours

implementation to demonstrate the efficiency and effectiveness of the proposed methodology.

IV. Experimental Evaluation

In this section, we conduct two experiments to demonstrate the efficiency and effectiveness of our methodology. First, we perform a timing analysis by running the methodology with all three algorithms, incrementally increasing the complexity of the analyzed MVM. Then, we assess the methodology's effectiveness by analyzing a recently fabricated MVM from the literature with real-world parameters.

All computations were performed on an octa-core AMD Ryzen 7 PRO 4750U machine with 32 GB RAM.

A. Timing Analysis

In this section, we conduct a timing analysis using a multilevel MVM with the function mappings

$$f_{G_{\text{nom}}}(w) = \frac{323\,\mu\text{S}}{w_{\max}} w \qquad f_{V_{\text{nom}}}(x) = \frac{5\,\text{V}}{x_{\max}} x$$
$$f_{G_{\min}}(w) = 0.99 f_{G_{\text{nom}}}(w) \qquad f_{V_{\min}}(x) = 0.99 f_{V_{\text{nom}}}(x)$$
$$f_{G_{\max}}(w) = 1.01 f_{G_{\text{nom}}}(w) \qquad f_{V_{\max}}(x) = 1.01 f_{V_{\text{nom}}}(x)$$
$$f_y(I) = \frac{w_{\max} x_{\max}}{f_{G_{\text{nom}}}(w_{\max}) f_{V_{\text{nom}}}(x_{\max})} I$$

The parameters N, $w_{\max}$, and $x_{\max}$ as well as the timing and error results are displayed in Table I. It is clearly seen from the runtimes and speedup values that DPBA is more efficient and scalable than BFA and SBA. BFA is not able to solve the problem for $N \geq 20$ under 24 hours due to its high time complexity, and SBA is not able to solve the problem for $N \geq 20$ when $w_{\max} = 7$ and $x_{\max} = 7$. On the other hand, DPBA is able to solve these problems in minutes thanks to its polynomial time complexity.

The $\delta_{y_i,\max}$ values are, as expected, the same for all algorithms. We further observe that the $\delta_{y_i,\max}$ values increase with N, $w_{\max}$, and $x_{\max}$. This is due to the accumulation of errors for increased N, and the increased y values for increased $w_{\max}$ and $x_{\max}$.

B. Case Study: A Fabricated 32×32 MVM

In this case study, we analyze a fabricated 32×32 MVM [21] using its measured, post-fabrication parameters. The device employs three-terminal FGFETs, whose gate voltage is used to set the conductance between the drain and source terminal. The input voltage V_j is applied to the drain terminal and only takes an *on* ($V_j = 1$ V) or *off* ($V_j = 0$ V) state. Therefore, we have $x_{\max} = 1$. The input voltage is considered precise, which gives us

$$f_{V_{\min}}(x) = f_{V_{\max}}(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{if } x = 1 \end{cases}$$

The maximum symbolic value for the weights is $w_{\max} = 3$ which are set with the gate voltages -4 V, -6 V, -8 V, and -10 V. The corresponding conductances have a very wide spread as seen in terms of arbitrary units in [21, Fig. 2]. We compute the conductances in Siemens from the data provided along with the paper, resulting in the mappings $f_{G_{\min}}(w)$ and $f_{G_{\max}}(w)$ given in Table II.

TABLE II: Conductances per State of the Fabricated MVM

w	0	1	2	3
$f_{G_{\min}}(w)$ (µS)	0	0	0	0
$f_{G_{\max}}(w)$ (µS)	0.67	3.68	8.57	14.63
$f_{G_{\text{nom}}}(w)$ (µS)	0.23	1.04	2.29	4.98

The MVM facilitates the measurement of the column-wise output current, by converting it into a voltage with a trans-impedance amplifier and providing a voltage terminal for readouts. Therefore, any f_y function may be used during applications. For an initial analysis, we use the gain that was used to convert the impedances into arbitrary units, i.e., we use $f_y(I) = 2 \times 10^6 I$.

Applying the methodology, we find $\delta_{y_i,\max} = 96$. This is due to the very wide range of the conductance values. In fact, for all states, the minimum observable conductance is zero, which means that for any applied input, the output may be zero. This simply means that the technology is not ready for safety-critical applications that require error bounds. Next, we will analyze possible future improvements to the technology.

We assume that the precision of the conductances is improved such that the minimum and maximum conductances are 10% below and above the nominal conductances, respectively, and fit a second-order f_y function to the data, resulting in

$$f_y(I) = \min(0.156 \times 10^9 \cdot I^2 + 8.79705 \times 10^5 \cdot I, 96)$$

For this case, we get $\delta_{y_i,\max} = 20.29$ at

$$\vec{w}_i{}^* = [3\,3\,3\,3\,3\,3\,3\,3\,3\,3\,3\,3\,3\,3\,3\,0\,3\,3\,3\,3\,3\,3\,0\,3\,3\,0\,0\,0\,3\,3\,0\,3\,0]$$
$$\vec{x}^* = [1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1]$$

Next, we assume that the conductances can be set exactly to the nominal values, but still get a maximum error of $\delta_{y_i,\max} = 11.35$ at

$$\vec{w}_i{}^* = [3\,3\,3\,3\,3\,3\,3\,3\,3\,3\,3\,0\,3\,3\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0]$$
$$\vec{x}^* = [1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1]$$

We see from the trace that the maximum error is related to the conductance state 3. Indeed, as seen in Table II, the conductance increases by a factor of 2.17 from state 2 to state 3. When the conductance of state 3 is lowered to 3.45 µS, we get $\delta_{y_i,\max} = 4.08$.

Therefore, for safety-critical applications requiring error bounds, it is necessary to both improve the precision of the conductances and to adjust conductance values.

V. Conclusion

In this paper, we have introduced a methodology for finding formally verified error bounds to resistive-switching-based multilevel MVMs. We have proposed our polynomial-time algorithm, DPBA, and demonstrated its efficiency and scalability through a timing analysis. We have also demonstrated the methodology's effectiveness through a case study with a fabricated MVM, particularly in analyzing error bounds and guiding future device improvements.

To ensure the correctness of our methodology, we have provided the proof of Theorem 1. The applicability to real devices is solely dependent on the correct choice of F_G and F_V. To account for real-world inaccuracies, F_G and F_V can be chosen bigger for a conservative analysis.

Next, we plan to use this formal verification methodology to create a framework that optimizes ANN weight to conductance mappings such that the error bounds are minimized. Furthermore, we plan to analyze error mitigation methods, such as the retraining of ANN weights and the use of redundant neurons and synapses.

References

[1] A. Ankit, I. Chakraborty, A. Agrawal, M. Ali, and K. Roy, "Circuits and Architectures for In-Memory Computing-Based Machine Learning Accelerators," *IEEE Micro*, vol. 40, no. 6, pp. 8–22, Nov. 2020.

[2] P. Mannocci, M. Farronato, N. Lepri, L. Cattaneo, A. Glukhov, Z. Sun, and D. Ielmini, "In-memory computing with emerging memory devices: Status and outlook," *APL Machine Learning*, vol. 1, no. 1, p. 010902, 2023.

[3] Y. Xi, B. Gao, J. Tang, A. Chen, M.-F. Chang, X. S. Hu, J. V. D. Spiegel, H. Qian, and H. Wu, "In-memory Learning with Analog Resistive Switching Memory: A Review and Perspective," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 14–42, Jan. 2021.

[4] H. Bao *et al.*, "Toward memristive in-memory computing: Principles and applications," *Frontiers of Optoelectronics*, vol. 15, no. 1, p. 23, May 2022.

[5] H. Tann, S. Hashemi, R. I. Bahar, and S. Reda, "Hardware-Software Codesign of Accurate, Multiplier-free Deep Neural Networks," in *Proceedings of the 54th Annual Design Automation Conference 2017*. Austin TX USA: ACM, Jun. 2017, pp. 1–6.

[6] S. Channamadhavuni, S. Thijssen, S. K. Jha, and R. Ewetz, "Accelerating AI Applications using Analog In-Memory Computing: Challenges and Opportunities," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, ser. GLSVLSI '21. New York, NY, USA: Association for Computing Machinery, Jun. 2021, pp. 379–384.

[7] C. Mackin *et al.*, "Optimised weight programming for analogue memory-based deep neural networks," *Nature Communications*, vol. 13, no. 1, p. 3765, Jun. 2022.

[8] N. Uysal, B. Zhang, S. K. Jha, and R. Ewetz, "DP-MAP: Towards resistive dot-product engines with improved precision," in *Proceedings of the 39th International Conference on Computer-Aided Design*, ser. ICCAD '20. New York, NY, USA: Association for Computing Machinery, Dec. 2020, pp. 1–9.

[9] B. Zhang, N. Uysal, D. Fan, and R. Ewetz, "Redundant Neurons and Shared Redundant Synapses for Robust Memristor-based DNNs with Reduced Overhead," in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, ser. GLSVLSI '20. New York, NY, USA: Association for Computing Machinery, Sep. 2020, pp. 339–344.

[10] D. Joksas, E. Wang, N. Barmpatsalos, W. H. Ng, A. J. Kenyon, G. A. Constantinides, and A. Mehonic, "Nonideality-Aware Training for Accurate and Robust Low-Power Memristive Neural Networks," *Advanced Science*, vol. 9, no. 17, p. 2105784, 2022.

[11] M. J. Rasch *et al.*, "Hardware-aware training for large-scale and diverse deep learning inference workloads using in-memory computing-based accelerators," *Nature Communications*, vol. 14, no. 1, p. 5282, Aug. 2023.

[12] A. Eldebiky, G. L. Zhang, G. Böcherer, B. Li, and U. Schlichtmann, "CorrectNet+: Dealing With HW Non-Idealities in In-Memory-Computing Platforms by Error Suppression and Compensation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 2, pp. 573–585, Feb. 2024.

[13] D. Joksas, P. Freitas, Z. Chai, W. H. Ng, M. Buckwell, C. Li, W. D. Zhang, Q. Xia, A. J. Kenyon, and A. Mehonic, "Committee machines—a universal method to deal with non-idealities in memristor-based neural networks," *Nature Communications*, vol. 11, no. 1, p. 4273, Aug. 2020.

[14] B. Zhang, N. Uysal, D. Fan, and R. Ewetz, "Representable Matrices: Enabling High Accuracy Analog Computation for Inference of DNNs using Memristors," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Beijing, China: IEEE, Jan. 2020, pp. 538–543.

[15] S. Jain, A. Sengupta, K. Roy, and A. Raghunathan, "RxNN: A Framework for Evaluating Deep Neural Networks on Resistive Crossbars," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 2, pp. 326–338, Feb. 2021.

[16] S. Valancius, E. Richter, R. Purdy, K. Rockowitz, M. Inouye, J. Mack, N. Kumbhare, K. Fair, J. Mixter, and A. Akoglu, "FPGA Based Emulation Environment for Neuromorphic Architectures," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. New Orleans, LA, USA: IEEE Computer Society, May 2020, pp. 90–97.

[17] M. N. Mondal, S. Sur-Kolay, and B. B. Bhattacharya, "Test Optimization in Memristor Crossbars Based on Path Selection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 1, pp. 294–307, Jan. 2023.

[18] K. Ç. Coşkun, M. Hassan, L. Hedrich, and R. Drechsler, "Efficient Equivalence Checking of Nonlinear Analog Circuits using Gradient Ascent," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, Nov. 2024, pp. 1–6.

[19] M. Češka, J. Matyáš, V. Mrazek, L. Sekanina, Z. Vasicek, and T. Vojnar, "Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished," in *Proceedings of the 36th International Conference on Computer-Aided Design*, ser. ICCAD '17. Irvine, CA, USA: IEEE Press, Nov. 2017, pp. 416–423.

[20] C. K. Jha, M. Hassan, and R. Drechsler, "cecApprox: Enabling Automated Combinational Equivalence Checking for Approximate Circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 7, pp. 3282–3293, Jul. 2024.

[21] G. Migliato Marega, H. G. Ji, Z. Wang, G. Pasquale, M. Tripathi, A. Radenovic, and A. Kis, "A large-scale integrated vector–matrix multiplication processor based on monolayer molybdenum disulfide memories," *Nature Electronics*, vol. 6, no. 12, pp. 1–8, Nov. 2023.

[22] R. Drechsler, *Formal Verification of Circuits*. New York: Springer, 2000.

[23] R. Drechsler, Ed., *Advanced Formal Verification*. New York: Springer, 2004.

[24] R. Drechsler, "PolyAdd: Polynomial Formal Verification of Adder Circuits," in *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. Vienna, Austria: IEEE, Apr. 2021, pp. 99–104.

[25] R. Drechsler and A. Mahzoon, "Polynomial Formal Verification: Ensuring Correctness under Resource Constraints," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '22. New York, NY, USA: Association for Computing Machinery, Dec. 2022, pp. 1–9.

[26] S. Agarwal, R. B. Jacobs Gedrim, A. H. Hsia, D. R. Hughart, E. J. Fuller, A. A. Talin, C. D. James, S. J. Plimpton, and M. J. Marinella, "Achieving ideal accuracies in analog neuromorphic computing using periodic carry," in *2017 Symposium on VLSI Technology*. Kyota, Japan: IEEE, Jun. 2017, pp. T174–T175.

[27] D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," *Nature Electronics*, vol. 1, no. 6, pp. 333–343, Jun. 2018.