High-Precision Close-to-Analog Programming of PCM Cells as Devices for AiMC Edge-AI

Alessio Antolini, *Member, IEEE*, Andrea Lico, Francesco Zavalloni, *Graduate Student Member, IEEE*, Lorenzo Greco, *Graduate Student Member, IEEE*, Riccardo Zurla, Jacopo Bertolini, Riccardo Vignali, Luca Iannelli, Emanuela Calvetti, Marco Pasotti, Alessandro Cabrini, and Eleonora Franchi Scarselli, *Member, IEEE*

***Abstract*—This article presents a high-precision close-to-analog programming methodology for phase-change memory (PCM) cells, targeting analog in-memory computing (AiMC) architectures for edge-artificial intelligence (AI) applications. By leveraging an iterative accumulation approach during the verify phase, the proposed technique enhances the effective number of bits (ENOBs) achievable in PCM cells. Experimental validation on a 28-nm STMicroelectronics FD-SOI PCM-based AiMC macro demonstrates the capability to accurately program large-scale matrix-vector multiplication (MVM) weights. The results highlight the potential of the proposed programming scheme to overcome intrinsic device variability, resulting in high accuracy in MVM computation. The methodology also shows promising applicability for mapping deep neural network (DNN) and large language model (LLM) weights with high precision, reaching an ENOB of 10.5.**

***Index Terms*—Analog in-memory computing (AiMC), analog programming, deep neural network (DNN), large language model (LLM), matrix-vector multiplication (MVM), phase-change memory (PCM).**

I. Introduction

DATA-CENTRIC applications and artificial intelligence (AI) have driven advancements in computing architectures to overcome the traditional Von Neumann model. Specifically, conventional processing systems exhibit a considerable energy consumption overhead ascribed to data movements, since processing units and memory are physically separated [1]. Along with compute-in-memory (CiM) architectures [2], [3], analog in-memory computing (AiMC) addresses this challenge by executing operations directly in the non-volatile memory unit, typically through Ohm's and Khirkhoff's laws.

Received 25 November 2025; revised 2 March 2026; accepted 24 March 2026. Date of publication 10 April 2026; date of current version 1 July 2026. This article was approved by Associate Editor Hua Wang. *(Alessio Antolini and Andrea Lico contributed equally to this work.) (Corresponding author: Alessio Antolini.)*

Alessio Antolini, Andrea Lico, Francesco Zavalloni, Lorenzo Greco, and Eleonora Franchi Scarselli are with ARCES-DEI, University of Bologna, 40126 Bologna, Italy (e-mail: alessio.antolini@unibo.it).

Riccardo Zurla is with STMicroelectronics, 27100 Pavia, Italy.

Jacopo Bertolini is with STMicroelectronics, 20864 Agrate Brianza, Italy.

Riccardo Vignali, Luca Iannelli, and Alessandro Cabrini are with the University of Pavia, 27100 Pavia, Italy.

Emanuela Calvetti and Marco Pasotti are with STMicroelectronics, 20864 Agrate Brianza, Italy.

Digital Object Identifier 10.1109/JSSC.2026.3680266

As reported in [4], a one-transistor, two-RRAM (1T2R) resistive memory cell can store 2 bits by integrating two-RRAM devices per cell with orthogonal bitlines (BLs) and employing parallel current-voltage sensing. In addition, the work in [5] presents a non-volatile metal–ferroelectric–metal (MFM) core enabling high-precision AiMC. In [6], a reconfigurable floating-gate memory supports multilevel conductance with 7-bit operation. Finally, [7] demonstrates an AiMC system with an electrochemical random access memory device. In this context, AiMC architectures based on phase-change memory (PCM) devices have emerged as a promising technology for accelerating machine learning workloads at the edge [8], offering significant improvements in energy efficiency and computational density. The core computational primitive in AiMC systems is the matrix-vector multiplication (MVM), which is typically executed by encoding matrix weights as conductance states of PCM cells and input vectors as pulsewidth modulation (PWM) signals. Furthermore, multilevel programming increases the computing density of IMC and AiMC cores with respect to conventional binary storage by minimizing the number of cells required for computation.

Emerging workloads such as large language models (LLMs) introduce additional challenges for AiMC systems [9]. In particular, LLM weight and activation distributions are characterized by high kurtosis and heavy tails, with the presence of outliers that play a critical role in preserving inference accuracy. As highlighted in [10], directly mapping the weights of LLM layers onto AiMC tiles with limited effective precision leads to significant accuracy degradation. Achieving high computational precision in such systems critically depends on the ability to program PCM cells to fine-grained multilevel conductance states [11].

Also, deploying deep neural network (DNN) weights onto AiMC cores presents challenges, as it often requires per-column rescaling and normalization to fully exploit the dynamic range of the MVM outputs [12], [13]. This rescaling significantly increases the number of distinct conductance levels that must be accurately programmed within the memory array.

Prior approaches have attempted to mitigate these issues through hardware-aware (HWA) training techniques that explicitly model quantization noise, device non-linearity, and temporal drift [14]. However, such methods are

computationally expensive and particularly challenging to apply LLMs due to their large number of parameters involved. As a result, improving the precision in programming the memory devices becomes a key enabler for accurate LLM and DNN inference on AiMC hardware, independently of the adopted training methodology.

The key contribution of this article is to address a circuit-level limitation of AiMC macros, namely, the restricted ADC resolution when the same read paths are used for both computation and verify step in the programming phase, as typically required to minimize weights programming errors originating from read-path mismatches and nonidealities. In large AiMC macros, this constraint prevents conventional programming schemes from achieving high precision. Extending [15], this article proposes a high-precision, close-to-analog programming algorithm for PCM cells that exploits iterative accumulation of ADC readouts during the verify phase to effectively increase the resolution beyond the native ADC resolution. This approach enables a substantial reduction in programming error and a significant increase in the effective number of bits (ENOBs), thereby improving the accuracy of the stored weights in AiMC arrays.

This article is organized as follows. Section II reviews PCM-based AiMC architectures and summarizes state-of-the-art multilevel programming techniques. Section III details the proposed analog programming methodology, showing the theoretical framework for resolution increase through accumulation. Section IV presents experimental results obtained from a 28-nm PCM AiMC macro [16], briefly describing the programming circuitry and demonstrating the efficacy of the method in programming MVM weights. In Section V, the influence on MVM accuracy over time is shown, also when combined with techniques to compensate cells drift. Section VI discusses case studies involving the deployment examples of DNNs and LLMs on AiMC hardware using the proposed programming scheme. Finally, Section VII concludes the article.

II. PCM-Based AiMC Architectures

A. Matrix-Vector Multiplication

The typical structure of an ePCM-based AiMC core is depicted in Fig. 1. The MVM is executed in the memory array, which leverages the physical properties of the memory cell to implement analog computations, whereas inputs encoding and the outputs readout are managed by a finite-state machine (FSM). Dedicated programming circuitry is devoted to the non-volatile weights programming. All operations are typically arranged by a microcontroller (μC) integrated in the same macro.

MVM $\mathbf{z} = \mathbf{W} \cdot \mathbf{x}$ can be executed within an ePCM array of dimensions $n \times m$ as illustrated in Fig. 2(a). Each matrix element $w_{i,j}$ is represented by conductance $g_{i,j}$ stored in one or more PCM devices. The input elements x_i of the vector $\mathbf{x} = [x_1, \ldots, x_n]$ are generally encoded in PWM intervals T_i, which are applied to the ith wordline (WL) [17]. By biasing the jth BL with a voltage V_B and integrating the resulting currents $i_{BL,i,j} = g_{i,j} V_B$, the charge Q_j^{MVM} of the whole BL accumulated

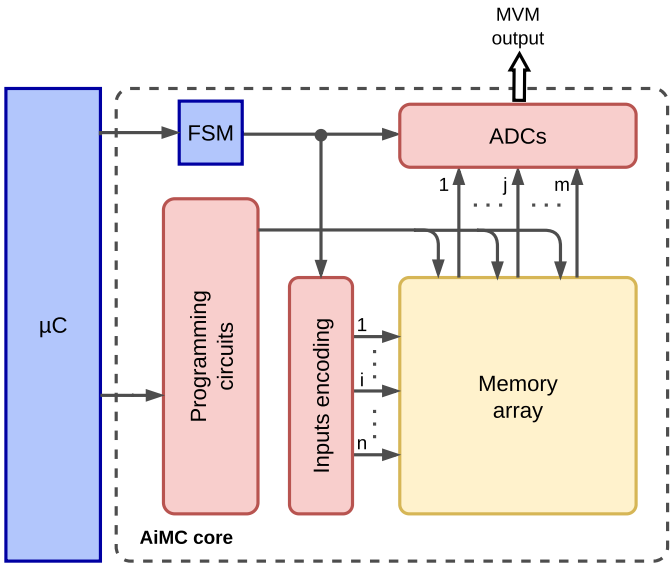

Fig. 1. Typical architecture of an ePCM-based AiMC core. The MVM is executed within the memory array, while input encoding and output readout are managed by an FSM. Dedicated programming circuitry provides non-volatile weight storage, and a microcontroller (μC) integrated in the same macro orchestrates and schedules all operations.

over the maximum integration time T^{MAX} supported by the ADC is

$$Q_j^{MVM} = \int_0^{T^{MAX}} i_{BL,j}(t)\, dt = V_B \sum_{i=1}^{n} g_{i,j} T_i. \tag{1}$$

The BL electrical charge is then converted into an N-bit digital string by the corresponding ADC to generate the MVM output [18]

$$z_j = \left\lfloor \frac{Q_j^{MVM}}{Q_{FSR}} 2^N \right\rfloor \tag{2}$$

where Q_{FSR} corresponds to the ADC full scale range.

The precision of the MVM $\mathbf{z} = [z_1, \ldots, z_m]$ is limited by different error contributions, such as current-over-resistance (IR) drop [19], A/D conversion non-linearity [20], temperature- and time-induced drift [21], [22], and limited coefficients programming precision. Focusing on the latter contribution, this work targets multilevel programming of conductances $g_{i,j}$, which is essential in PCM-based AiMC.

B. State-of-the-Art PCM Cells Multilevel Programming Techniques

PCM cells' programming algorithms have evolved to address intrinsic challenges such as device variability, conductance drift, and the stringent requirements on programming accuracy [19]. The programming process typically involves iterative execution of program-and-verify (P&V) steps until the cell conductance $g_{i,j}$ converges within a specified tolerance to the target value. During each iteration, a current or voltage pulse is applied to incrementally adjust the cell conductance [23], modulating it between two extreme states: the high-conductance SET state and the low-conductance RESET state. The programming operation is managed by dedicated write

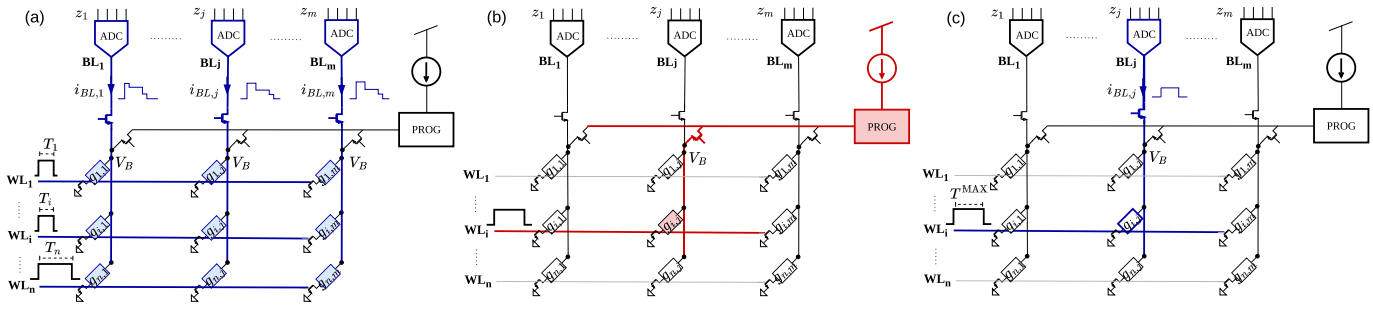

Fig. 2. AiMC operation modes. The computational phase of an MVM operation is represented in (a), showing the BL currents as a function of time. The P&V of cell $g_{i,j}$ requires first the application of a current pulse to the selected PCM device (b) through dedicated programming circuitry; then, by activating the corresponding ith WL for a time interval T^{MAX} the cell conductance is measured and read on the corresponding N-bit ADC (c).

circuity, which commonly exploits a distinct path with respect to analog computations [see Fig. 2(b)]. Consequently, several P&V strategies have been developed, primarily aiming to maximize the ENOBs per cell.

Statistical models that capture the stochastic crystallization dynamics to optimize programming pulses have been proposed in [24]. In [25], an adaptive pulsewidth and pipeline-based programming methodology is introduced, achieving an ENOB of 4. Hybrid programming techniques, such as those presented in [26], employ bidirectional programming curves to accelerate convergence, reducing the time-to-program and attaining an ENOB of 4.

The work in [13] leverages a dual-cell architecture combined with a tunable P&V algorithm, reporting an ENOB of 4. Differential weight encoding approaches, exemplified by [27], utilize voltage-driven weak SET-staircase (SSC) programming to mitigate drift-induced fluctuations, achieving an ENOB of 6. Similarly, the quad-cell architecture described in [28] compensates for undershoot and overshoot effects during programming, yielding an ENOB of 5.

Gradient descent-based programming, as demonstrated in [29], directly minimizes MVM errors by iteratively updating conductances using stochastic gradient information derived from on-chip computations. The methodology in [20] employs current-driven SSC with P&V, achieving an ENOB of 5.5.

III. Analog Programming of PCM Cells

A. ADC-Based Cells Programming

As previously mentioned, PCM cells' multilevel programming requires a conductance readout at each verify step. As illustrated by Fig. 2(c), this step typically employs the same ADCs used for the MVM computation, ensuring that the programming process accounts for non-idealities in the computation chain, such as variability and non-linearity [30]. The readout of the conductance of a single PCM device, denoted as $g_{i,j}$, during the verify step can be accurately performed by selectively activating the corresponding ith WL for the maximum activation interval T^{MAX}. This selective activation ensures that only the targeted device contributes to the integrated current. As a result, the jth ADC produces a digital output string $z_{i,j}$, which encodes the quantity $Q_{i,j}$ defined as

$$Q_{i,j} = V_B g_{i,j} T^{MAX} \tag{3}$$

so that

$$z_{i,j} = \left\lfloor \frac{Q_{i,j}}{Q_{FSR}} 2^N \right\rfloor . \tag{4}$$

Since the ADC full scale range Q_{FSR} is chosen in relation to (1), Q_{FSR} is reasonably much higher than the maximum value $Q_{i,j}^{MAX} = V_B g^{MAX} T^{MAX}$ of (3), where g^{MAX} is the maximum conductance of PCM cells. Since the ADC is assumed to have a resolution of N bits, the effective number of bits N_{eff}, of the digital output $z_{i,j}^{MAX}$ encoding $Q_{i,j}^{MAX}$, can be expressed as

$$N_{eff} = N - \log_2 \left(\frac{1}{\gamma} \right) \tag{5}$$

where

$$\gamma := \frac{Q_{i,j}^{MAX}}{Q_{FSR}} < 1 . \tag{6}$$

The effective number of bits in (5) quantifies the actual resolution available for representing a single cell reading within the ADC dynamic range. A proof of the previous equation can be found in the Appendix.

The value of γ depends on the actual hardware implementation and technology of the AiMC core and quantifies the resolution loss of the single cell programming output with respect to the available ADC N bits. Assuming that the ADC is specifically dimensioned to have Q_{FSR} corresponding to the maximum value of the MVM output (1), i.e., $Q_{FSR} = nV_B g^{MAX} T^{MAX}$ and $\gamma = 1/n$, (5) becomes $N_{eff} = N - \log_2 n$. As a result, the increase in the number n of cells per BL produces a decrease in the precision of the programming, yielding to a trade-off between computing capacity and MVM weights programming resolution.

Since g^{MAX} is generally limited by technology and the BL biasing voltage V_B must remain consistent with the value used during MVM computations, increasing γ can be achieved by increasing the integration time T^{MAX} in the programming phase, which corresponds to lowering the clock frequency that drives the PWM input signal. Nevertheless, this approach cannot increase N_{eff} beyond the ADC native resolution N.

B. Accumulation for Fairly Analog Programming

To overcome the previous limitation, we propose to iterate the computation of $Q_{i,j}$ and its conversion to $z_{i,j}$ at each verify

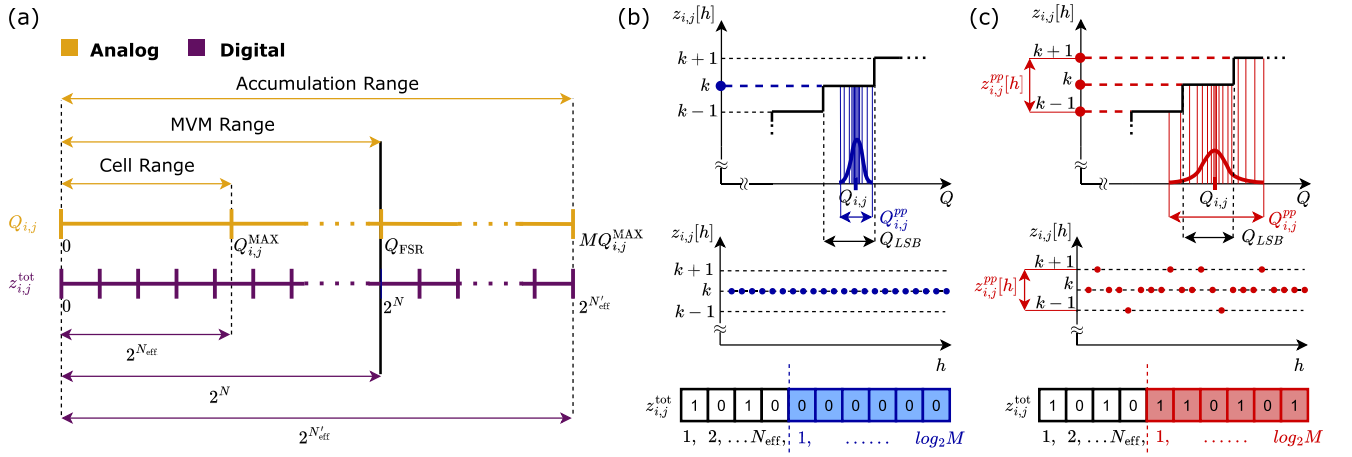

Fig. 3. (a) Relationship between analog or analog-equivalent quantities: $Q_{i,j}^{\mathrm{MAX}}$, Q_{FSR}, and $MQ_{i,j}^{\mathrm{MAX}}$, along with their counterparts in the digital domain: $2^{N_{\mathrm{eff}}}$, 2^{N}, and $2^{N'_{\mathrm{eff}}}$. The figure highlights how the native ADC resolution can be extended by exploiting the accumulation factor M. (b) and (c) Schematic representation of the variability on samples $z_{i,j}[h]$ to allow for programming resolution increase with accumulation; k is representative of the generic ADC output level. The cell variability in (b) is not sufficient to grant an accuracy increase, whereas in (c), variability enhances the output resolution. More specifically, in (b), additional $\log_2 M$ bits of the digital string do not convey any information, since the accumulation effect would correspond to a left-shift.

step. The digital samples $z_{i,j}$ obtained at each iteration h are then accumulated in the microcontroller as

$$z_{i,j}^{\mathrm{tot}} = \sum_{h=1}^{M} z_{i,j}\,[h] \tag{7}$$

where M denotes the total number of iterations. Therefore, the overall effective number of bits, N'_{eff}, associated with $z_{i,j}^{\mathrm{tot}}$ can be expressed as

$$N'_{\mathrm{eff}} = N_{\mathrm{eff}} + \log_2 M = N - \log_2\left(\frac{1}{\gamma}\right) + \log_2 M\,. \tag{8}$$

Consequently, if $M > 1/\gamma$, the number of possible target values for $z_{i,j}^{\mathrm{tot}}$, given by $2^{N'_{\mathrm{eff}}}$, can exceed the N-bit resolution of the ADC [see Fig. 3(a)]. With this choice, the maximum value for the digital string $z_{i,j}^{\mathrm{tot}}$ is equal to $Mz_{i,j}^{\mathrm{MAX}} = 2^{N'_{\mathrm{eff}}} - 1$ and corresponds to an equivalent analog charge $MQ_{i,j}^{\mathrm{MAX}} > Q_{\mathrm{FSR}}$, as shown in Fig. 3(a). Equation (8) proves that the accumulation-based P&V algorithm theoretically enables a fairly analog programming of the PCM cell conductance with enhanced accuracy. The theoretical achieved displacement of N'_{eff} relative to the resolution N of the ADC is shown in Fig. 4(a), which reports the trend of N'_{eff} as a function of M for different values of γ [as defined (8)]. It should be noted that the condition $N'_{\mathrm{eff}} < N$ represents a degradation of the programming resolution with respect to the ADC resolution, and this loss is more evident the lower values for γ and M are. A gain in programming resolution ($N'_{\mathrm{eff}} \geq N$) can be achieved when $M > 1/\gamma$, as represented by the vertical dashed lines.

However, an additional condition must be added on $Q_{i,j}$ variability over samples to guarantee that the additional $\log_2 M$ bits in (8) convey information. As shown in Fig. 3(c), $Q_{i,j}$ read at each step h need to vary sufficiently so that the converted samples $z_{i,j}[h]$ spread across different quantization levels. If, instead, the output samples all fall into the same quantization level at every step h, as in Fig. 3(b), the accumulation does not add any information. Let us assume that the variability of $Q_{i,j}$ could be represented as a Gaussian noise with standard deviation σ_Q. The condition to ensure that the samples are distributed over multiple quantization levels is given by $\sigma_Q \geq Q_{\mathrm{LSB}}/\sqrt{12}$, where Q_{LSB} denotes the ADC quantization step defined as $Q_{\mathrm{LSB}} := Q_{\mathrm{FSR}}/2^N$, and $Q_{\mathrm{LSB}}/\sqrt{12}$ corresponds to the standard deviation of the quantization noise [31], which is characterized by a uniform probability distribution. Under this assumption, let us assume the peak-to-peak amplitude $Q_{i,j}^{pp}$ of the charge $Q_{i,j}$ as $Q_{i,j}^{pp} = 6\sigma_Q$. This yields to a condition on $Q_{i,j}^{pp}$

$$\frac{Q_{i,j}^{pp}}{Q_{\mathrm{FSR}}} \geq \frac{\sqrt{3}}{2^N} \tag{9}$$

which has to be satisfied so that multiple accumulations grant an increase of the verify accuracy for all possible cell values over the conductance range. The sources of variability are mainly ascribed to PCM cells' noise and computation chain non-idealities. It should be noted that the constraint on the variability of $Q_{i,j}$ becomes less severe as the ADC number of bits N increases with fixed Q_{FSR}. In PCM devices, the noise-induced sample distribution across multiple levels can be empirically verified with the sample amplitude $z_{i,j}^{pp}$, as confirmed by the experimental results shown in Section IV.

C. Programming Algorithm and Equivalent Number of Bits (ENOB)

Fig. 5 reports the sketch of the iterative P&V algorithm of a PCM cell. The process begins with the definition of a target value $\hat{z}_{i,j}^{\mathrm{tot}}$, which is subsequently compared to the accumulated measured output $z_{i,j}^{\mathrm{tot}}$ in order to verify the state of the cell. If the programming error

$$\varepsilon_p := \left|\hat{z}_{i,j}^{\mathrm{tot}} - z_{i,j}^{\mathrm{tot}}\right| \tag{10}$$

is less than the programming tolerance $\Delta_p \geq 1$, the programming process terminates. Otherwise, a new programming attempt, typically involving the application of an appropriate

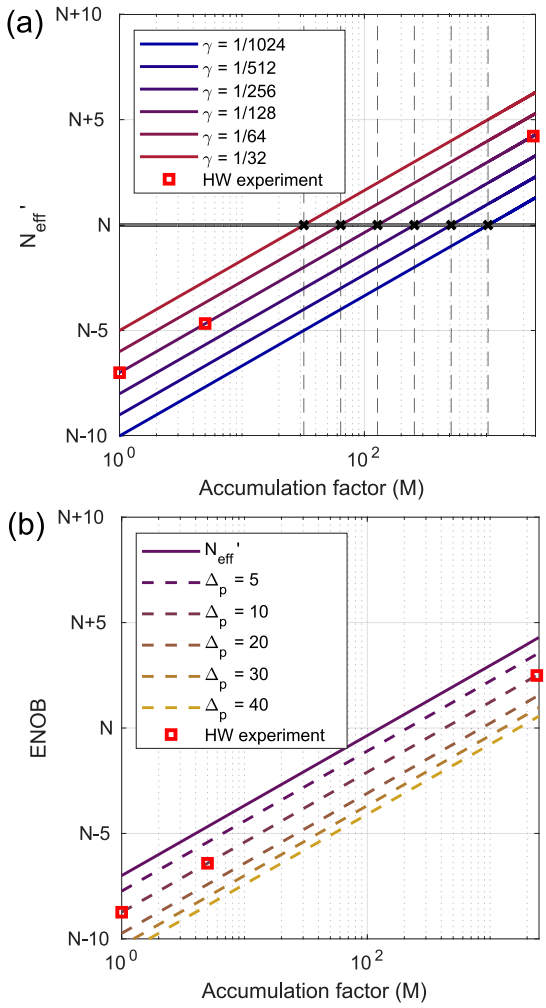

Fig. 4. (a) Theoretical trend of the effective programming resolution N'_{eff} as a function of the accumulation factor M for different values of γ, as given by (8). Vertical dashed lines show that increasing M allows to increase the programming resolution even for $\gamma \ll 1$. (b) Achievable ENOB versus M for different programming tolerances Δ_p, highlighting that the finite programming precision imposes ENOB $\leq N'_{\text{eff}}$ (assuming $\Delta_p/3 > 1$). In all panels, red squares mark the HW constraints and programming conditions adopted in the experimental demonstration of Section IV.

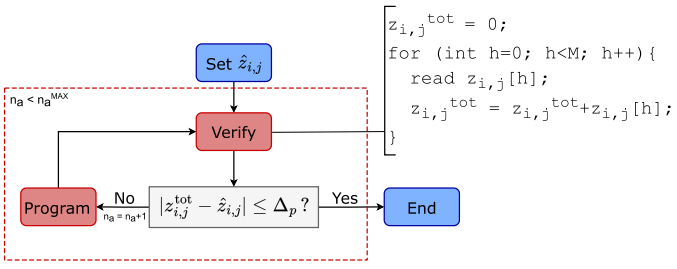

Fig. 5. Sketch of the proposed programming algorithm for PCM cells with accumulation phase at each verify step. The P&V sequence is executed by increasing an iterations counter n_a which is kept under an upper threshold n_a^{MAX}.

current pulse, is executed. As a result of the accumulation performed during the verify phase, the target value $\hat{z}_{i,j}^{\text{tot}}$ can vary within the range $\left[0; \hat{z}_{i,j}^{\text{MAX}}\right]$, where

$$\hat{z}_{i,j}^{\text{MAX}} := \max\left(z_{i,j}^{\text{tot}}\right) = 2^{N'_{\text{eff}}} - 1. \tag{11}$$

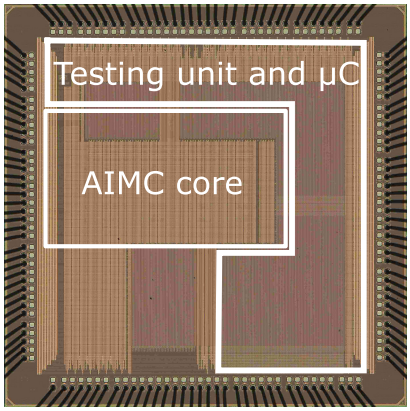

Fig. 6. Micrograph of the employed AiMC core. A testing unit and a microcontroller are integrated in the same macro.

The precision of the programming outcome is quantified through the ENOBs [27], which can be empirically evaluated as

$$\text{ENOB} = \log_2\left(\frac{\hat{z}_{i,j}^{\text{MAX}}}{\sigma_{\varepsilon_p}}\right) \tag{12}$$

where σ_{ε_p} is the standard deviation of the programming error ε_p. Assuming ε_p to be a Gaussian distribution delimited by $\pm\Delta_p$, its standard deviation σ_{ε_p} can be approximated as $\sigma_{\varepsilon_p} \simeq \Delta_p/3$; moreover, recalling (11), (12) can be estimated as

$$\text{ENOB} \simeq N'_{\text{eff}} - \log_2\left(\frac{\Delta_p}{3}\right). \tag{13}$$

Equation (13) shows that N'_{eff} represents an upper-bound of the achievable ENOB, which is lowered by the intrinsic finite programming precision. Indeed, to guarantee the convergence of the programming algorithm (i.e., the number of programming attempts n_a being under a maximum threshold n_a^{MAX}), the tolerance Δ_p should be sufficiently large; thus, it is reasonable to assume $(\Delta_p)/3 > 1$, which yields to ENOB $\leq N'_{\text{eff}}$. This effect is shown in Fig. 4(b), which reports ENOB as a function of M considering different values for Δ_p, assuming $\gamma = 1/128$. As a conclusion, the accumulation factor M must be properly chosen in relation to both the architecture's requirements and programming constraints.

In both plots, the red squares indicate the HW constraints and programming conditions that will be shown in Section IV-C to demonstrate the proposed technique.

It should also be noted that the introduction of accumulation in the verify step increases both the time and the power required to program a cell. However, such overheads are generally acceptable in inference-oriented DNN accelerators, where synaptic weights are programmed during an offline provisioning phase and remain fixed during operation, as commonly assumed in AiMC architectures [32], [33], [34], [35]. A more detailed analysis of this matter, supported by experimental data related to the employed test vehicle [16], is presented in Section IV-C.

IV. Experimental Results

The test vehicle utilized is a Ge-rich GST PCM-based AiMC macro fabricated in a STMicroelectronics 28-nm

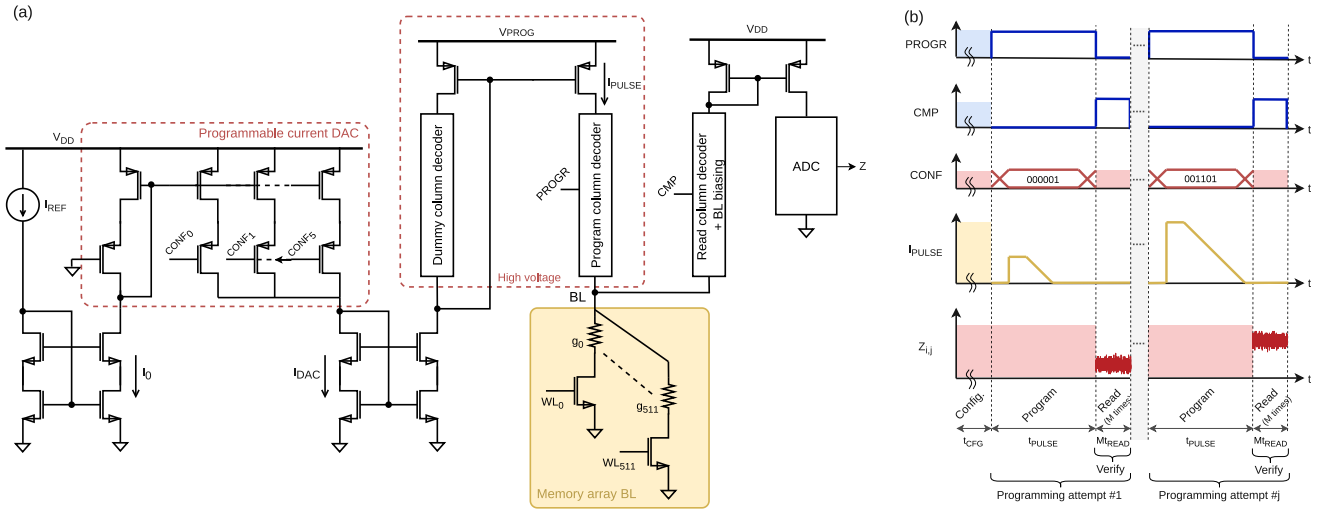

Fig. 7. (a) Schematic of the on-chip programming circuit. A reference current I_{REF} is mirrored to I_0 and converted by a 6-bit current DAC (controlled by CONF[5 . . . 0]) to generate I_{DAC}, which is delivered to the selected cell through a high-voltage mirror biased at $V_{\mathrm{PROG}} > V_{\mathrm{DD}}$ along the programming path (PROGR = 1) and column decoder. (b) Timing diagram of the main signals during program–verify: after a configuration phase t_{CFG}, the system enables V_{PROG} and the programming path to synthesize the 64-level I_{PULSE} over t_{PROG}; then the read path (CMP = 1) is enabled and the cell state $z_{i,j}$ is read by the ADC.

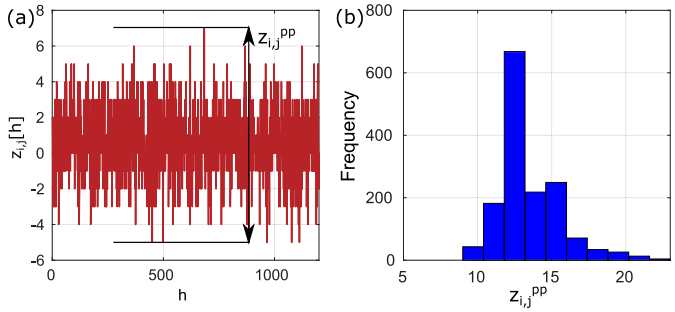

Fig. 8. (a) Example of experimentally measured samples $z_{i,j}[h]$ as a function of $h = [0, \ldots, 1200]$ [as in the theoretical sketch in Fig. 3(c)] and corresponding definition of $z_{i,j}^{pp}$. (b) Distribution of $z_{i,j}^{pp}$ over a set of 1600 PCM cells.

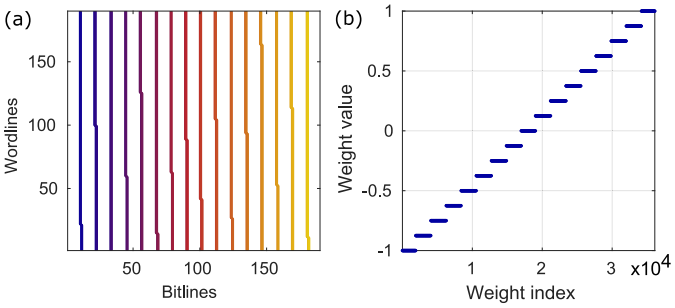

Fig. 9. Contour map (a) and coefficients distribution (b) of the 190 × 190 programmed coefficients without accumulation ($M = 1$).

FD-SOI process, featuring a 4M-cell array (see Fig. 6) [16]. The AiMC macro executes a 512 × 512 signed MVM by leveraging 512 integrated ADCs to digitize the BL charge, which is biased at $V_B = 0.1$ V, into an 11-bit sign-magnitude format. The input data x_i are encoded in an 8-bit sign-magnitude format. The signed MVM weights $w_{i,j}$ are stored in two cells (as detailed in [16]), whose conductance difference gives the absolute value of weights. Accordingly, N'_{eff} and ENOB are referred in the following to the programmed signed weights. Based on the electrical characteristics of the integrated ePCM array cells and of the ADCs, the prototype exhibits a $\gamma \simeq 1/128$.

A. Programming Circuitry

The programming circuits integrated in the testchip are shown in Fig. 7(a) [36]. The scheme allows a current pulse to be applied to a single cell during the programming phase. This pulse is generated starting from a current reference I_{REF}, which is appropriately mirrored to I_0. The latter is fed into an additional programmable mirroring stage (a 6-bit current DAC, configured by CONF[5 . . . 0]), whose control is handled by the digital logic of the testchip. The resulting current I_{DAC} is first delivered to the cell through a high-voltage ($V_{\mathrm{PROG}} > V_{\mathrm{DD}}$) mirroring stage, and flows along the dedicated programming path (enabled with PROGR = 1) and the column decoding selector. The read path is used during the verify phase (as well as during MVM computation, as mentioned in the introduction), and is enabled by setting CMP = 1. The column decoder as well is split in two paths to handle both standard and high-voltage operations.

Fig. 7(b) schematically illustrates the dynamics of the main signals involved in the programming procedure. After an initial configuration phase of duration t_{CFG}, the system enables the programming path (PROGR = 1) and, by suitably modulating the DAC at the system clock frequency, generates the shape (discretized into 64 possible levels) of the desired pulse I_{PULSE}, resulting in a duration t_{PULSE}. Subsequently, to verify the actual state of the cell, the read path is enabled (CMP = 1) and, after properly biasing the BL at V_B, the WL corresponding to the cell is enabled for a time t_{READ}, and the result $z_{i,j}$ is computed by the ADC. The whole verify step t_{V} consists in M successive readouts, in accordance with the accumulation factor, so that

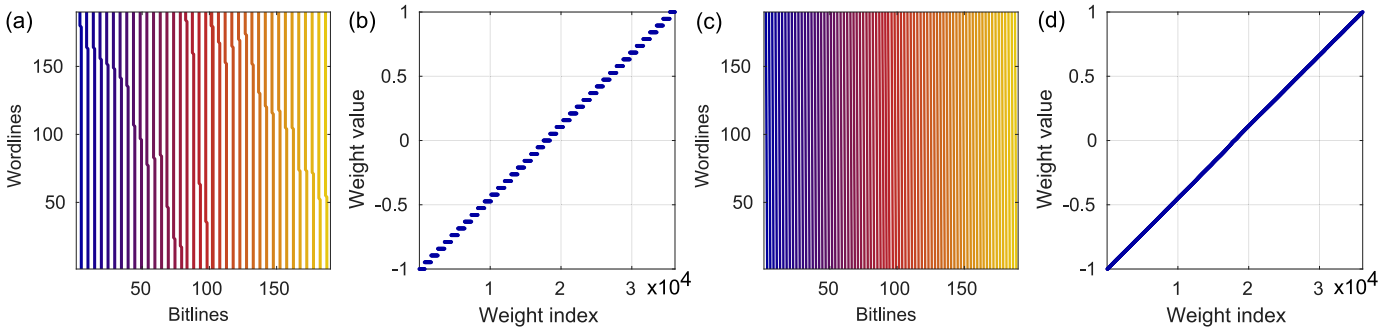

Fig. 10. (a) Contour map and (b) coefficients distribution of the 190 × 190 programmed coefficients with accumulation $M = 5$. (c) Contour map and (d) coefficients distribution of the 190 × 190 programmed coefficients with accumulation $M = 2400$.

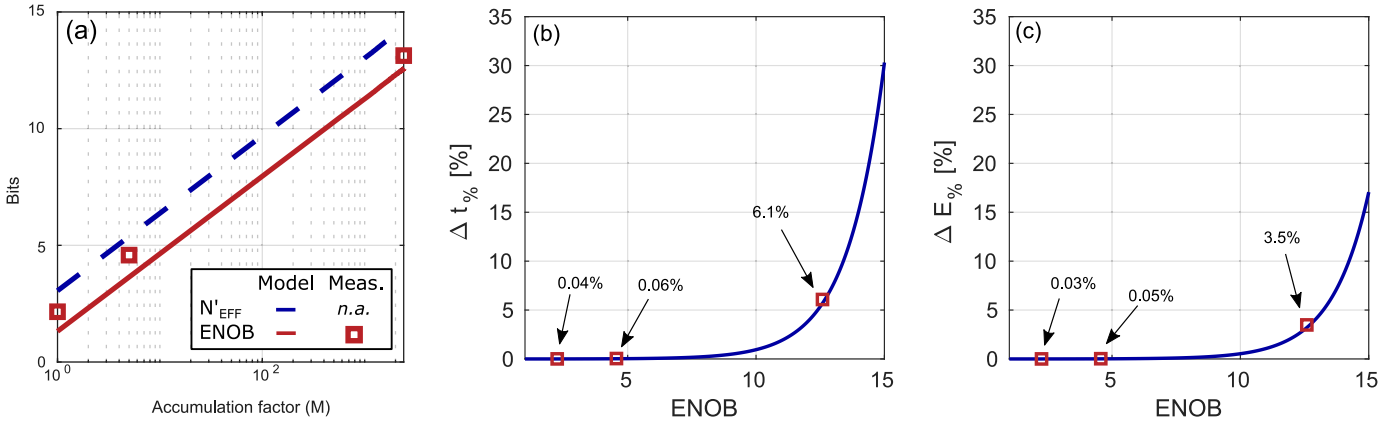

Fig. 11. (a) Theoretical N'_{eff} [dashed line, see (8)] as a function of M, highlighting the possibility to achieve arbitrarily high precision of the MVM coefficients. The estimated and measured ENOB of Section IV-C are also reported as continuous lines and dots, respectively, where the former is obtained by setting $\sigma_{\varepsilon_p} = \Delta_p/3$ in (13), and the latter by using the experimentally extracted σ_{ε_p}. (b) Relative programming time overhead $t_{\%}$ versus the required ENOB, computed by combining (14) with (13). (c) Relative programming energy overhead $E_{\%}$ versus the required ENOB, obtained from (15) and (13). In (b) and (c), continuous lines refer to the current test vehicle, whereas squared dots correspond to the programming setup of Section IV-C.

$t_{\text{V}} = Mt_{\text{READ}}$. The accumulation is performed concurrently by the microcontroller. It should be noted that the configuration phase t_{CFG} is considerably longer than the remaining steps. All the P&V steps are repeated in relation to the n_a attempts required to properly program the cell (see Fig. 5).

As a result, the time demanded to program a single cell can be estimated as

$$t_{\text{PROG}} = t_{\text{CFG}} + n_a \left[t_{\text{PULSE}} + Mt_{\text{READ}}\right] \tag{14}$$

and the corresponding energy as

$$E_{\text{PROG}} = E_{\text{CFG}} + n_a \left[E_{\text{PULSE}} + ME_{\text{READ}} + (M-1)\,E_{\text{A}}\right] \tag{15}$$

where all terms are referred to previous operations, and E_{A} is the energy contribution of the single accumulation (out of $M - 1$) performed by the microcontroller to implement the proposed programming procedure.

B. PCM Cells Variability Characterization

Before validating the accumulation method, the variability of charge samples $z_{i,j}$ was first characterized as stated in Section III-B. To this purpose, a set of 1600 cells was programmed to different intermediate conductance states by applying random current pulses with no verify step. Then, each cell was read 1200 times through the ADC to monitor its read noise amplitude $z^{pp}_{i,j}$ (see Fig. 8(a), where the y-axis represents the ADC output for each sample $z_{i,j}[h]$). Fig. 8(b) shows the distribution of the collected $z^{pp}_{i,j}$ over the 1600 cells, demonstrating that their value is invariably above the ADC quantization step, so that it allows for the precision increase brought by N'_{eff}.

C. PCM Cells Programming

To assess the previous method, a set of ~36 000 weights was programmed with the algorithm detailed in Section III-C. A first programming outcome was implemented measuring $z_{i,j}$ with no accumulation ($M = 1$) through the ADCs. Fig. 9(a) reports a contour map of the measured programmed 190 × 190-weights matrix, whose normalized distribution is reported in (b) as a function of the programmed weight. Results show that 8 signed, nonzero levels are programmed within the matrix. This is in accordance with (5), since, because of the value of γ, the expected effective number of bit is $N'_{\text{eff}} = N_{\text{eff}} = 3$.

The same set of weights was then programmed with accumulation $M = 5$. This leads, [from (8)], to an estimated $N'_{\text{eff}} \simeq 5.3$. As reported by the experimental results in Fig. 10(a) and (b), this case allows for the programming of 80 weights (half negative and half positive), which confirms

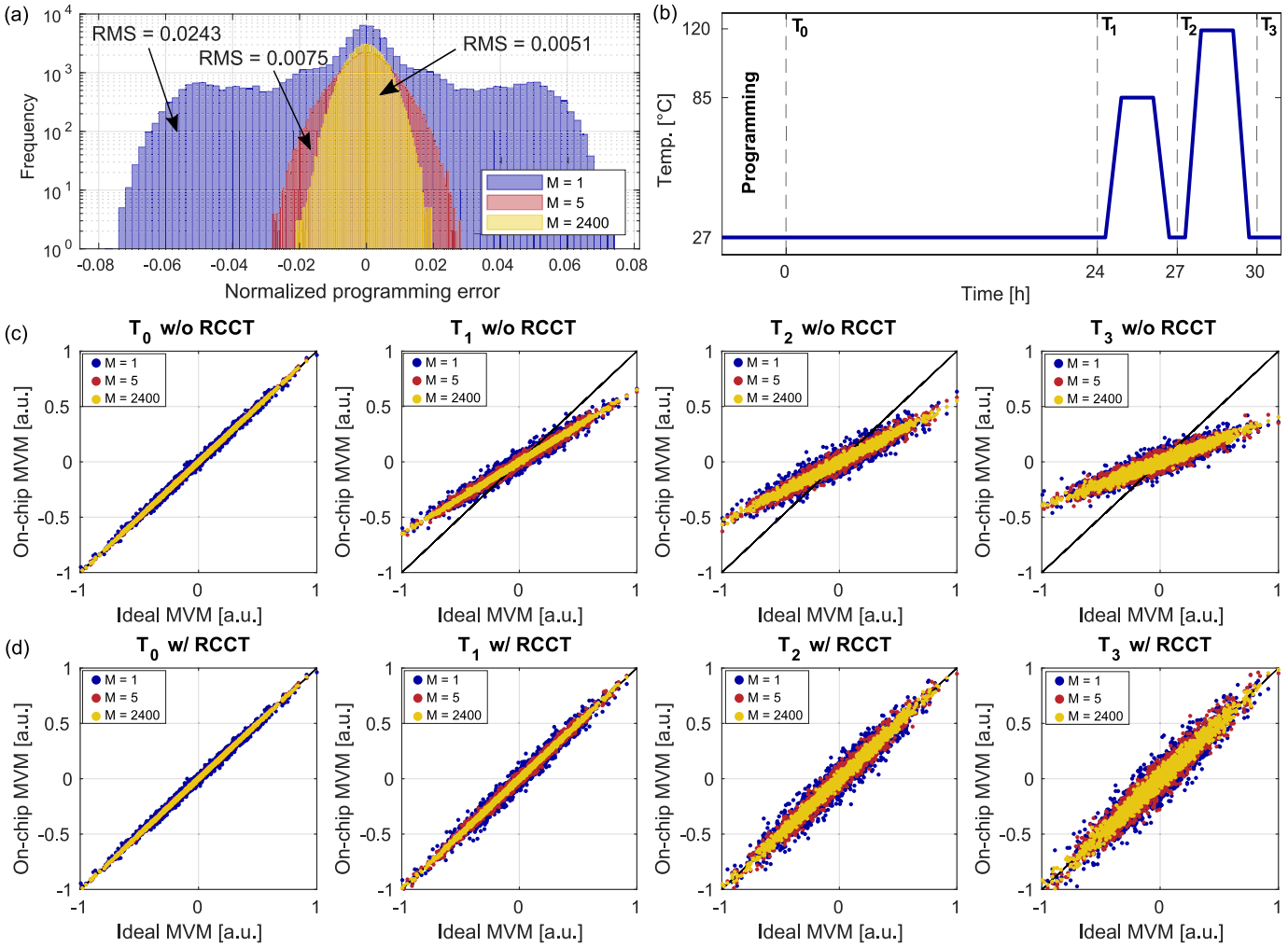

Fig. 12. (a) Log-scale histogram of the programming error $\varepsilon_{\mathbf{z}}$, along with the corresponding rms values, for different accumulation factors. (b) Illustration of the experimental protocol used to evaluate the accuracy of the MVM over time, with measurements taken immediately after programming (T_0), after 24 h (T_1), and following two high-temperature annealing cycles with subsequent room-temperature readouts (T_2, T_3). (c) and (d) Temporal evolution of MVM accuracy: scatter plots of on-chip MVM results $\mathbf{z}_i$ versus ideal values $\mathbf{z}_i^{\text{id}}$ for different programming conditions (accumulation factors $M = 1, 5, 2400$) and measurement times. The top plots show degradation caused by conductance drift, while the bottom plots illustrate partial recovery enabled by the RCCT strategy.

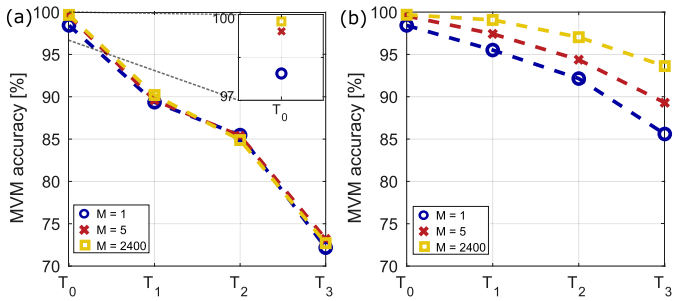

Fig. 13. MVM accuracy α_z [defined in (19)] as a function of time and temperature, varying the accumulation factor M, without (a), and with (b) drift compensation through RCCT.

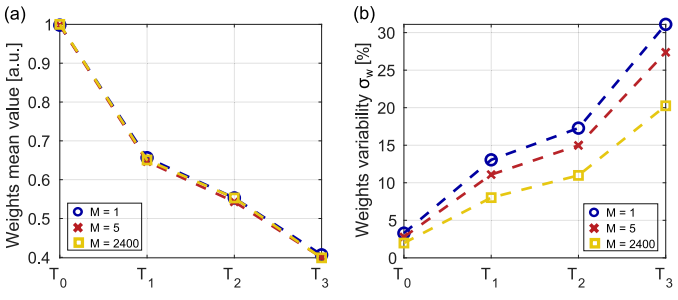

Fig. 14. Evolution in time of (a) mean value and (b) relative variability σ_w of the weights employed for MVM accuracy evaluation.

the previous estimation of N'_{eff}. An accumulation factor $M = 2400$ was then tested, so that the expected effective number of bits can be increased to $N'_{\text{eff}} \simeq 14.2$. The programming outcome is reported in Fig. 10(c) and (d), where more than 18 000 different signed weights were successfully programmed among the PCM cells array.

As additional analysis of previous results, Fig. 11(a) summarizes the theoretical N'_{eff} [see (8)] as a function of the accumulation factor M (dashed line), considering the parameters constrained by the employed test vehicle. The estimated

TABLE I
COMPARATIVE TABLE OF THE MOST RECENT MULTILEVEL PROGRAMMING ALGORITHMS OF PCM CELLS FOR AIMC

Reference	Technology	Programming type	Programming sequence	Programming approach	ENOB
[24] (2018)	90 nm	Current-driven	Partial-SET pulse	Statistical	*n.a.*
[28] (2021)	14 nm	Current-driven	*n.a.*	P&V*	5
[20] (2022)	14 nm	Current-driven	SET-staircase	P&V*	5.5
[43] (2022)	40 nm	Current-driven	*n.a.*	*n.a.*	8
[29] (2023)	14 nm	Current-driven	RESET-staircase	Gradient Descent P&V*	*n.a.*
[13] (2023)	14 nm	Current-driven	SET-staircase	P&V*	4
[26] (2023)	40 nm	Current-driven	Hybrid	P&V*	4
[27] (2024)	90 nm	Voltage-driven	Weak SET-staircase	P&V*	6
[25] (2025)	40 nm	Current-driven	Width SET-staircase	P&V*	4
This work	**28 nm**	**Current-driven**	**SET-staircase**	**P&V*with accumulation**	**14.2**** **10.5*****

* P&V refers to Program-and-Verify.
** Obtained in programming uniform random weights.
*** Obtained in programming a real-world DNN application weights.

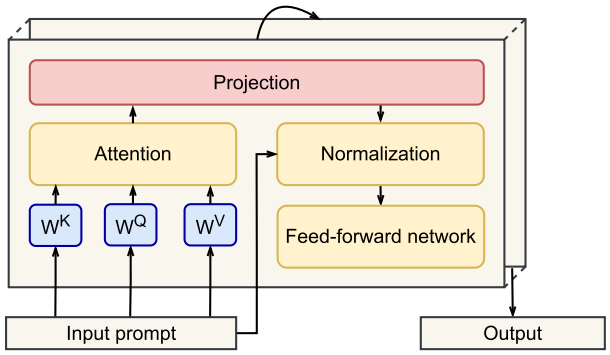

Fig. 15. Typical structure of a transformer-based LLM architecture. Layers W^K, W^Q, and W^V are implemented as MVMs, so that they can be performed on AiMC cores.

and measured ENOB are also reported as continuous lines and dots, respectively. The former is calculated with (13), whereas the latter is obtained with (12).

The proposed programming procedure implies programming time and energy overheads (with respect to a conventional readout, i.e., with $M = 1$), which are proportional to the accumulation coefficient M [see (14) and (15)], i.e., to the targeted programming ENOB. Such overheads can be quantified with respect to conventional programming ($M = 1$) as

$$\Delta t_{\%} = 1 + \frac{n_a\,(M-1)\,t_{\mathrm{READ}}}{t_{\mathrm{CFG}} + n_a\,(t_{\mathrm{PULSE}} + t_{\mathrm{READ}})} \tag{16}$$

and

$$\Delta E_{\%} = 1 + \frac{n_a\,(M-1)\,(E_{\mathrm{READ}} + E_{\mathrm{A}})}{E_{\mathrm{CFG}} + n_a\,(E_{\mathrm{PULSE}} + E_{\mathrm{READ}})} \tag{17}$$

where M can be rewritten as a function of ENOB, i.e.,

$$M = \frac{\Delta_p}{3\gamma} 2^{\mathrm{ENOB}-N}. \tag{18}$$

The numerical values of $\Delta t_{\%}$ and $\Delta E_{\%}$, specialized to current test vehicle, are plotted in Fig. 11(b) and (c) as a function of the targeted ENOB. The squared dots refer to the three programming setups employed in Section IV-C, underlying that a 6×-gain in ENOB ($M = 2400$) can be achieved with barely 6.1% and 3.5% time and energy overheads, respectively.

V. MVM ACCURACY

In the context of PCM-based AiMC, the accuracy of the MVM on-chip computation $\mathbf{z} = \mathbf{W} \cdot \mathbf{x}$ can be defined as

$$\alpha_z := 100\left(1 - \frac{\sigma_{\varepsilon_z}}{z^{\mathrm{MAX}}}\right) \tag{19}$$

where z^{MAX} corresponds to the maximum ideal MVM and σ_{ε_z} is standard deviation of the MVM error $\varepsilon_{\mathbf{z}}$. The latter can be defined with respect to the ideal MVM $\mathbf{z}^{\mathrm{id}}$ as

$$\varepsilon_{\mathbf{z}} := \left|\mathbf{z} - \mathbf{z}^{\mathrm{id}}\right|. \tag{20}$$

The MVM error $\varepsilon_{\mathbf{z}}$ is primarily affected by three types of weights errors [27]: 1) a drop in time of the average values of the stored coefficients induced by cells conductance drift [37]; 2) a cell-to-cell variability ascribable to programming error and to the randomness of the conductance drift; and 3) the weight quantization error [19].

To evaluate the influence of the proposed programming algorithm on α_z, a 380 × 380 random coefficients matrix $\mathbf{W}$ was defined along with a set of $N_{\mathrm{IN}} = 10\,000$ inputs $\mathbf{x}$, resulting in N_{IN} ideal MVMs $\mathbf{z}^{\mathrm{id}}$. Three sets of cells $g_{i,j}$ were then programmed to each $w_{i,j} \in \mathbf{W}$ using $M = 1$, $M = 5$ and $M = 2400$, respectively. The measured programming error ε_p [see (10)], normalized to $\hat{z}_{i,j}^{\mathrm{MAX}}$ (11), is shown as a log-scale histogram in Fig. 12(a) for different accumulation factors. As expected and proven by the error rms values (defined as $\|\varepsilon_p\|_2$) reported in the same plot, ε_p is reduced with increased M.

The MVMs were then performed in time, as in the scheme in Fig. 12(b), at room temperature immediately after cell programming (T_0) and after 24 h (T_1); in addition, two 3-h, high-temperature annealings (followed by two room-temperature readouts T_2 and T_3) were included to accelerate the effects of PCM cell drift [38], [39]. MVM results are depicted in Fig. 12(c) and (d), where each plot reports the on-chip MVM $\mathbf{z}$ as a function of the corresponding ideal value

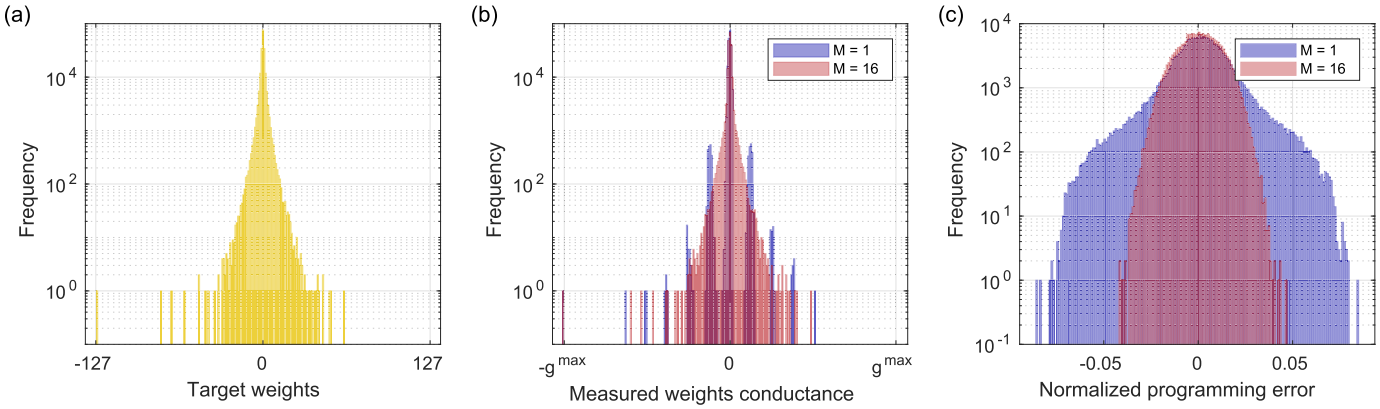

Fig. 16. (a) Distribution of the target weights of the 8-bit quantized matrix from the smolLM2 MVM layer. (b) Resulting measured weights distribution after programming, showing that $M = 16$ enables mapping across the full conductance range with increased resolution. (c) Normalized programming error resulting from both programming procedures, demonstrating a significant error reduction with the accumulation factor $M = 16$.

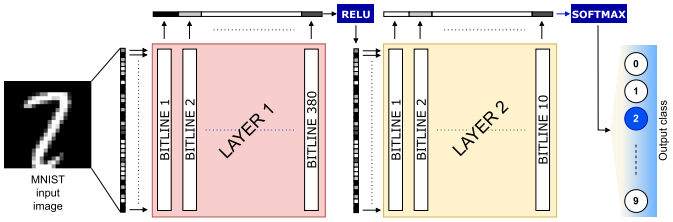

Fig. 17. Schematic representation of the DNN implemented on the PCM AiMC prototype.

$\mathbf{z}^{\text{id}}$; different colors refer to cells programmed with previous values of accumulation factor M.

Fig. 12(c) shows the MVM evolution in time under the effect of conductance drift, which implies a drop of the average MVM output and increased spread of the results (and thus of the MVM error $\varepsilon_{\mathbf{z}}$). As known [27], the standard deviation of $\varepsilon_{\mathbf{z}}$ can be estimated accordingly as

$$\sigma_{\varepsilon_{\mathbf{z}}} \simeq n\sqrt{\sigma_d^2 + \sigma_w^2 + \sigma_q^2} \tag{21}$$

where the single standard deviation contributions descend from the single errors previously mentioned. Specifically, σ_d refers to the drift-induced average drop in MVM output, σ_w quantifies weights variability, and σ_q results from weights quantization. It should be noted that $\sigma_{\varepsilon_{\mathbf{z}}}$ is proportional to the size of the MVM (i.e., the number n of weights per BL).

Since the close-to-analog programming achieves arbitrary programming precision [as in (13)], the quantization error scales proportionally to M, and equivalently its standard deviation σ_q. This is due to the weights variability ascribed to the programming error ε_p being reduced with increased M, as pointed out in Fig. 12(a). This is experimentally proven by evaluating the MVM accuracy right after programming (T_0), as reported in the inset in Fig. 13(a). The MVM accuracy heavily drops in T_1, T_2, and T_3 due to the predominant contribution of σ_d on the MVM error standard deviation $\sigma_{\varepsilon_{\mathbf{z}}}$, and no significant accuracy increase is achieved varying M to program the MVM weights [see Fig. 13(a)].

A. Combined Effect With HW Drift Compensation

In the measurements of Fig. 12(d), MVM outputs are partially recovered from cells conductance drop through a reference cell conductance tracking (RCCT) [16], [30] strategy. Thus, σ_d becomes negligible; this yields to

$$\sigma_{\varepsilon_{\mathbf{z}}} \propto \sqrt{\sigma_w^2 + \sigma_q^2}. \tag{22}$$

In this case, the reduction of σ_q and σ_w introduced by M becomes more evident and therefore contributes to a significant overall improvement in the MVM accuracy. Indeed, as reported in Fig. 13(b), an increased M effectively achieves an improved MVM precision over time and temperature thanks to the reduction of both σ_w and σ_q. In particular, α_z increases with respect to $M = 1$ by 2.5% and 5% in T_1 and T_3 with $M = 5$, and by 4.8% and 9.7% with $M = 2400$.

Previous results are supported by an analysis of the weight distribution in time. To this purpose, all weights have been normalized to their programmed value (in T_0), then, each weight was measured in T_1, T_2, and T_3 to monitor its time evolution in terms of mean value and variability σ_w, previously mentioned. Fig. 14(a) and (b) reports indeed the normalized average value and variability of the programmed weights, considering previous values for the accumulation factor M. It should be noted that, while the programming condition does not affect the mean value of the weights, on the contrary, their variability is sensibly reduced with increased M, so that higher MVM accuracy is achieved as well.

VI. Case Studies

Two representative LLM and DNN workloads mapped onto the PCM-based AiMC macro are used to evaluate the proposed programming methodology. The selected case studies illustrate practical programming scenarios, highlighting the achievable precision improvements and the conditions under which the proposed accumulation-based programming scheme yields significant advantages. Since the computation in the verify phase is performed through the ADC, the target weights are represented by integer numbers. However, the weights

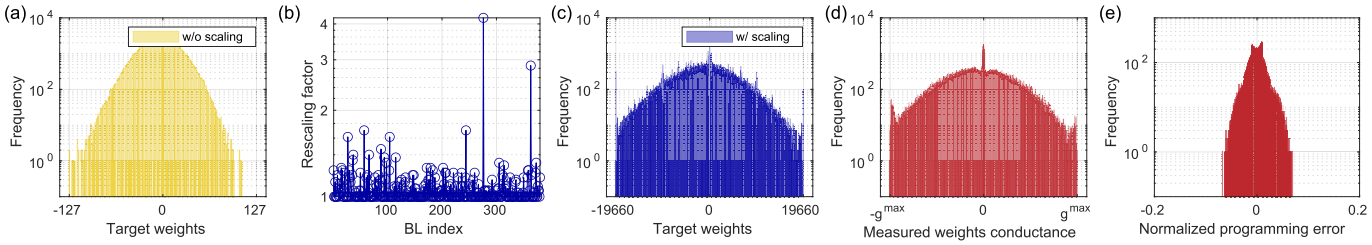

Fig. 18. (a) Distribution of the target weights of the 8-bit quantized matrix of the two DNN layers. (b) Rescaling factors used for each BL (log scale). (c) Target weights after rescaling. (d) Resulting measured weights distribution after programming using an accumulation factor $M = 2400$. (e) Normalized programming error.

can still represent matrices in other formats by applying an appropriate rescaling of the result during the computation phase.

A. Programming LLM Weights in AiMC Cores

In transformer-based architectures, projection layers are implemented through MVMs and are therefore well suited for execution on AiMC cores [40] (see Fig. 15). The proposed close-to-analog programming approach is evaluated using an 8-bit quantized weight matrix extracted from a smolLM2-135M projection layer [41], whose high-kurtosis distribution makes accurate mapping onto PCM-based AiMC cores particularly challenging.

The size of the matrix representing the weights of the layer is 380×512, involving 512 BLs of the PCM array. The 8-bit quantized weights matrix distribution is shown in Fig. 16(a). As discussed in Section IV-C, the maximum programming resolution achievable in the AiMC macro with standard programming ($M = 1$) is 3 bits. Achieving an effective 8-bit resolution when mapping the weight values onto the PCM conductances requires an accumulation factor of $M = 16$, as derived from (8) assuming $\gamma = 1/128$. These two programming configurations ($M = 1$ and $M = 16$) are therefore considered to compare their programming performance. The weight matrix was programmed into the array conductances using both accumulation factors in order to evaluate the resulting precision and error characteristics. The resulting measured weight distributions are shown in Fig. 16(b). The higher accumulation factor effectively increases the resolution of the weight representation, allowing a more precise representation of the weight values, thereby reducing quantization-induced accuracy degradation. The normalized programming error for both programming outcomes is reported in Fig. 16(c). The results demonstrate a clear reduction of the programming error with the accumulation factor $M = 16$, achieving an ENOB = 7.56 which approaches the 8-bits precision required to represent the LLM coefficients.

The improved precision is obtained at the cost of an increased number of verify iterations, highlighting a trade-off between programming time and achievable ENOB. However, this trade-off only affects the initial programming phase, which is performed once prior to inference, and does not introduce additional overhead during inference execution.

B. Programming DNN Weights in AiMC Tiles Using Rescaling Factors

As a second validation scenario, the proposed programming scheme is evaluated in the context of a conventional DNN deployment that requires per-BL rescaling to maximize the MVM output dynamic range [12], [13]. This case study highlights a practical condition in which the number of distinct conductance levels to be programmed significantly exceeds the native ADC resolution of the AiMC macro.

The DNN illustrated in Fig. 17, intended for MNIST image classification, was programmed and executed on the testchip implementing the MVMs of the two layers on the AiMC core (shown in Fig. 6 [16]), while the non-linear operations are performed by the microcontroller. The pixels of the MNIST input images were cropped to 19×20 and encoded in a 7-bit gray scale format. The first layer of the network comprises 380 neurons and employs the ReLU activation function, while the second layer comprises ten neurons and a softmax-operation. The size of the matrix representing the weights of the first and the second layers are 380×380 and 380×10, respectively, involving 380 and 10 BLs of the PCM array.

The DNN was initially trained in FP and then quantized in 8-bits signed precision for the coefficients, without employing any specific HW-aware training techniques [42]. To maximize the corresponding MVM output range, a rescaling factor was computed for each column (i.e., BL). The two resulting rescaled MVM matrices were programmed into the array. Since each BL requires a specific rescaling factor [as shown in Fig. 18(b)], the target weights increase from 127 [see Fig. 18(a)] (i.e., reduced to 7-bit due to the clipping) to 1901 [see Fig. 18(c)], which require ENOB $\simeq$ 11 for accurate representation.

The DNN was subsequently deployed within the AiMC core by programming the cells using the near-analog programming algorithm with an accumulation factor $M = 2400$, resulting in the target weight distribution shown in Fig. 18(c). The measured weight conductances are presented in Fig. 18(d), with the corresponding normalized programming error shown in Fig. 18(e). The programming procedure achieved a measured ENOB of 10.5, which closely approaches the 11 bits required to accurately represent the DNN coefficients.

VII. Conclusion

In this work, a programming technique for PCM cells in AiMC systems employing an iterative accumulation strategy

during the verify phase has been introduced. This method achieves quasi-analog programming precision at the cost of increased time and energy required in the programming phase, which, however, are not a significant constraint due to infrequent weight updates.

Experimental results are obtained on a STMicroelectronics 28-nm FD-SOI PCM AiMC macro [16]. The enhanced programming accuracy translates into improved MVM precision and robustness against conductance drift when combined with drift compensation techniques such as Reference Cell Conductance Tracking (RCCT), even considering two 3-h annealing at 120 ° C. Furthermore, the proposed approach is effective for both DNN and LLM models' weight mapping, thereby broadening the applicability of PCM-based AiMC architectures in edge-AI scenarios.

Table I presents a comparison of state-of-the-art techniques for programming PCM, evaluated using the ENOBs as the target metric. This work achieves an ENOB of 14.2 in random uniform weights programming and of 10.5 in a real-world DNN application, thus advancing the state-of-the-art in PCM cell programming.

APPENDIX
DERIVATION OF (5)

Considering Q_{LSB} as the electrical charge which corresponds to the LSB of the ADC, one can write

$$Q_{\mathrm{LSB}} = \frac{Q_{\mathrm{FSR}}}{2^N}. \tag{23}$$

The number of bits required to represent the maximum electrical charge $Q_{i,j}^{\mathrm{MAX}}$ associated with the single PCM cell in the verify step is given by

$$2^{N_{\mathrm{eff}}} := \frac{Q_{i,j}^{\mathrm{MAX}}}{Q_{\mathrm{LSB}}}. \tag{24}$$

Consequently, by defining γ as in (6) one obtains

$$2^{N_{\mathrm{eff}}} = \gamma 2^N \tag{25}$$

which proves (5).

REFERENCES

[1] W. Haensch, T. Gokmen, and R. Puri, "The next generation of deep learning hardware: Analog computing," *Proc. IEEE*, vol. 107, no. 1, pp. 108–122, Jan. 2019, doi: 10.1109/JPROC.2018.2871057.

[2] D. Wang, C.-T. Lin, G. K. Chen, P. Knag, R. K. Krishnamurthy, and M. Seok, "DIMC: 2219TOPS/W 2569F2/b digital in-memory computing macro in 28 nm based on approximate arithmetic hardware," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 65, Feb. 2022, pp. 266–268.

[3] A. Kneip, M. Lefebvre, J. Verecken, and D. Bol, "IMPACT: A 1-to-4b 813-TOPS/W 22-nm FD-SOI compute-in-memory CNN accelerator featuring a 4.2-POPS/W 146-TOPS/mm^2 CIM-SRAM with multi-bit analog batch-normalization," *IEEE J. Solid-State Circuits*, vol. 58, no. 7, pp. 1871–1884, Jul. 2023.

[4] W. Yue et al., "A 40 nm 48.7F 2 bit 1T2R resistive memory bitcell array for adaptive vector-symbolic in-memory computing," in *Proc. IEEE Eur. Solid-State Electron. Res. Conf. (ESSERC)*, Sep. 2025, pp. 241–244.

[5] K. Ota et al., "Non-volatile inverter with 3D cylindrical metal-ferroelectric-metal capacitor realizing digitized voltage output for computing-in-memory," in *Proc. IEEE Eur. Solid-State Electron. Res. Conf. (ESSERC)*, Sep. 2025, pp. 61–64.

[6] M. C. Nguyen et al., "Reconfigurable and ultrafast non-volatile floating-gate memory based on van der Waals heterostructures," in *Proc. IEEE Eur. Solid-State Electron. Res. Conf. (ESSERC)*, Sep. 2025, pp. 245–248.

[7] S. Kim et al., "Ultra-fast CV-ECRAM-based analog PIM with dynamic retention compensation techniques," in *Proc. IEEE Eur. Solid-State Electron. Res. Conf. (ESSERC)*, Sep. 2025, pp. 601–604.

[8] J. Hartmann, P. Cappelletti, N. Chawla, F. Arnaud, and A. Cathelin, "Artificial intelligence: Why moving it to the edge?," in *Proc. IEEE 47th Eur. Solid State Circuits Conf. (ESSCIRC)*, Sep. 2021, pp. 1–6.

[9] N. Leroux et al., "Analog in-memory computing attention mechanism for fast and energy-efficient large language models," *Nature Comput. Sci.*, vol. 5, no. 9, pp. 813–824, Sep. 2025, doi: 10.1038/s43588-025-00854-1.

[10] Y. Hou, H. Tsai, K. El Maghraoui, T. Gokmen, G. W. Burr, and L. Liu, "NORA: Noise-optimized rescaling of LLMs on analog compute-in-memory accelerators," in *Proc. Design, Autom. Test Eur. Conf. (DATE)*, Mar. 2025, pp. 1–7.

[11] W. Kim et al., "Confined PCM-based analog synaptic devices offering low resistance-drift and 1000 programmable states for deep learning," in *Proc. Symp. VLSI Technol.*, Jun. 2019, pp. T66–T67.

[12] M. J. Rasch, F. Carta, O. Fagbohungbe, and T. Gokmen, "Fast and robust analog in-memory deep neural network training," *Nature Commun.*, vol. 15, no. 1, p. 7133, Aug. 2024, doi: 10.1038/s41467-024-51221-z.

[13] M. Le Gallo et al., "A 64-core mixed-signal in-memory compute chip based on phase-change memory for deep neural network inference," *Nature Electron.*, vol. 6, no. 9, pp. 680–693, Aug. 2023, doi: 10.1038/s41928-023-01010-1.

[14] M. J. Rasch et al., "Hardware-aware training for large-scale and diverse deep learning inference workloads using in-memory computing-based accelerators," *Nature Commun.*, vol. 14, no. 1, p. 5282, Aug. 2023, doi: 10.1038/s41467-023-40770-4.

[15] A. Antolini et al., "High-precision close-to-analog programming of PCM cells as devices for AiMC edge-AI," in *Proc. IEEE Eur. Solid-State Electron. Res. Conf. (ESSERC)*, Sep. 2025, pp. 461–464.

[16] M. Pasotti et al., "28 M weights × Tops /W/mm^2 PCM-based analog in-memory computing core with 8 512 × 512-weight layers in 28 nm FD-SOI CMOS," in *Proc. 2025 IEEE Eur. Solid-State Electron. Res. Conf. (ESSERC)*, Nov. 2025, pp. 129–132.

[17] S. Cosemans et al., "Towards 10000TOPS/W DNN inference with analog in-memory computing—A circuit blueprint, device options and requirements," in *IEDM Tech. Dig.*, Dec. 2019, pp. 22.2.1–22.2.4.

[18] R. Vignali et al., "Designing circuits for AiMC based on non-volatile memories: A tutorial brief on trade-off and strategies for ADCs and DACs co-design," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 71, no. 3, pp. 1650–1655, Mar. 2024, doi: 10.1109/TCSII.2023.3340112.

[19] M. Bertuletti, I. M. Noz-Martín, S. Bianchi, A. G. Bonfanti, and D. Ielmini, "A multilayer neural accelerator with binary activations based on phase-change memory," *IEEE Trans. Electron Devices*, vol. 70, no. 3, pp. 986–992, Mar. 2023.

[20] R. Khaddam-Aljameh et al., "HERMES-core—A 1.59-TOPS/mm^2 PCM on 14-nm CMOS in-memory compute core using 300-ps/LSB linearized CCO-based ADCs," *IEEE J. Solid-State Circuits*, vol. 57, no. 4, pp. 1027–1038, Apr. 2022, doi: 10.1109/JSSC.2022.3140414.

[21] I. Boybat et al., "Temperature sensitivity of analog in-memory computing using phase-change memory," in *IEDM Tech. Dig.*, Dec. 2021, pp. 1–4.

[22] A. Antolini et al., "Controlled acceleration of PCM cells time drift through on-chip current-induced annealing for AIMC multilevel MVM computation," *IEEE Trans. Electron Devices*, vol. 72, no. 1, pp. 215–221, Jan. 2025.

[23] A. Sebastian, M. Le Gallo, and E. Eleftheriou, "Computational phase-change memory: Beyond von Neumann computing," *J. Phys. D, Appl. Phys.*, vol. 52, no. 44, Oct. 2019, Art. no. 443002.

[24] S. R. Nandakumar, M. Le Gallo, I. Boybat, B. Rajendran, A. Sebastian, and E. Eleftheriou, "A phase-change memory model for neuromorphic computing," *J. Appl. Phys.*, vol. 124, no. 15, Oct. 2018, Art. no. 152135.

[25] X. Wang, L. Yan, X. Li, Y. Tao, Z. Song, and Y. Yang, "An efficient pipeline programming scheme based on 40 nm PCM compute-in-memory chip for CNNs," in *Proc. 9th IEEE Electron Devices Technol. Manuf. Conf. (EDTM)*, Mar. 2025, pp. 1–3.

[26] Q. Wu et al., "Hybrid program algorithm enables significant reduction in write latency and power consumption for multilevel phase change memory," *IEEE Trans. Electron Devices*, vol. 70, no. 8, pp. 4145–4149, Aug. 2023.

[27] L. Pistolesi et al., "Differential phase change memory (PCM) cell for drift-compensated in-memory computing," *IEEE Trans. Electron Devices*, vol. 71, no. 12, pp. 7447–7453, Dec. 2024.

[28] P. Narayanan et al., "Fully on-chip MAC at 14 nm enabled by accurate row-wise programming of PCM-based weights and parallel vector-transport in duration-format," *IEEE Trans. Electron Devices*, vol. 68, no. 12, pp. 6629–6636, Dec. 2021.
[29] J. Büchel et al., "Programming weights to analog in-memory computing cores by direct minimization of the matrix-vector multiplication error," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 13, no. 4, pp. 1052–1061, Dec. 2023.
[30] A. Antolini et al., "A readout scheme for PCM-based analog in-memory computing with drift compensation through reference conductance tracking," *IEEE Open J. Solid-State Circuits Soc.*, vol. 4, pp. 69–82, 2024.
[31] T. C. Carusone, D. A. Johns, and K. W. Martin, *Analog Integrated Circuit Design*, 3rd ed., Hoboken, NJ, USA: Wiley, 2012.
[32] G. W. Burr et al., "Experimental demonstration and tolerancing of a large-scale neural network (165000 synapses) using phase-change memory as the synaptic weight element," *IEEE Trans. Electron Devices*, vol. 62, no. 11, pp. 3498–3507, Nov. 2015.
[33] S. R. Nandakumar et al., "Precision of synaptic weights programmed in phase-change memory devices for deep learning inference," in *IEDM Tech. Dig.*, Dec. 2020, pp. 1–4.
[34] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnol.*, vol. 15, no. 7, pp. 529–544, Jul. 2020, doi: 10.1038/s41565-020-0655-z.
[35] A. Chen et al., "Demonstration of transformer-based ALBERT model on a 14nm analog AI inference chip," *Nature Commun.*, vol. 16, no. 1, p. 8661, Sep. 2025, doi: 10.1038/s41467-025-63794-4.
[36] M. Pasotti et al., "A 32-KB ePCM for real-time data processing in automotive and smart power applications," *IEEE J. Solid-State Circuits*, vol. 53, no. 7, pp. 2114–2125, Jul. 2018.
[37] A. Antolini, A. Lico, E. F. Scarselli, M. Carissimi, and M. Pasotti, "Phase-change memory cells characterization in an analog in-memory computing perspective," in *Proc. 17th Conf. Ph.D Res. Microelectron. Electron. (PRIME)*, Jun. 2022, pp. 233–236.
[38] F. G. Volpe, A. Cabrini, M. Pasotti, and G. Torelli, "Drift induced rigid current shift in Ge-rich GST phase change memories in low resistance state," in *Proc. 26th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Nov. 2019, pp. 418–421.
[39] D. Ielmini, A. L. Lacaita, and D. Mantegazza, "Recovery and drift dynamics of resistance and threshold voltages in phase-change memories," *IEEE Trans. Electron Devices*, vol. 54, no. 2, pp. 308–315, Feb. 2007, doi: 10.1109/TED.2006.888752.
[40] P. Li, H. Seferoglu, V. R. Dasari, and E. Koyuncu, "Model-distributed DNN training for memory-constrained edge computing devices," in *Proc. IEEE Int. Symp. Local Metrop. Area Netw. (LANMAN)*, Jul. 2021, pp. 1–6.
[41] L. Ben Allal et al., "SmolLM2: When smol goes big-data-centric training of a small language model," 2025, *arXiv:2502.02737*.
[42] V. Joshi et al., "Accurate deep neural network inference using computational phase-change memory," *Nature Commun.*, vol. 11, no. 1, May 2020, doi: 10.1038/s41467-020-16108-9.
[43] W.-S. Khwa et al., "A 40-nm, 2M-cell, 8b-precision, hybrid SLC-MLC PCM computing-in-memory macro with 20.5–65.0TOPS/W for tiny-Al edge devices," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 65, Feb. 2022, pp. 1–3.

Alessio Antolini (Member, IEEE) received the Ph.D. degree in electronic engineering from the University of Bologna, Bologna, Italy, in March 2023.

Since November 2019, he has been with the "Ercole De Castro" Advanced Research Centre on Electronic Systems for Information and Communication Technologies, where his activities are focused on the physical characterization of phase-change memory (PCM) devices, and on the design of integrated circuits for analog in-memory (AIMC) architectures. He is currently a Junior Assistant Professor with the "Guglielmo Marconi" Department of Electrical, Electronic and Information Engineering (DEI), University of Bologna.

Andrea Lico received the M.Sc. degree in electronics engineering and the Ph.D. degree in electronics, telecommunications, and information technologies from the University of Bologna, Bologna, Italy, in 2021 and 2024, respectively.

From 2021 to 2025, he was at the Advanced Research Center on Electronic Systems (ARCES), University of Bologna, as a Researcher and a Post-Doctoral Fellow. Since 2026, he has been an Analog IC Research Engineer with Fondazione Chips-IT, Bologna, Italy. His research interests include analog integrated circuits for analog in-memory computing, with emphasis on phase-change memory (PCM)-based architectures.

Francesco Zavalloni (Graduate Student Member, IEEE) received the Dr.Eng. degree (Hons.) in electronic engineering from the University of Bologna, Bologna, Italy, in 2021, where he is currently pursuing the Ph.D. degree with the ET-IT Ph.D. Program, which is developed inside the Advanced Research Center on Electronic Systems (ARCES).

His research activity focuses on analog in-memory computing, PCM-based programming algorithms, neural networks, and modeling PCM-based system behaviour.

Lorenzo Greco (Graduate Student Member, IEEE) received the master's degree (Hons.) in electronics engineering from the University of Bologna, Bologna, Italy, in 2024, where he is currently pursuing the Ph.D. degree in electronics engineering, telecommunications, and information technologies with the "Guglielmo Marconi" Department of Electrical, Electronic, and Information Engineering.

His research is conducted in collaboration with STMicroelectronics, Pavia, Italy, within the joint Laboratory ST-Advanced Research Center on Electronic Systems (ARCES). His main research interests lie in integrated circuit design, with a particular focus on analog circuits. His current research activity focuses on analog in-memory computing (AIMC) using phase-change memory (PCM).

Riccardo Zurla received the Ph.D. degree in microelectronics from the University of Pavia, Pavia, Italy, in 2018.

From 2018 to 2021, he was a Post-Doctoral Research Fellow at the University of Pavia, where he focused on the design of analog circuits for non-volatile memories, particularly phase-change memories (PCMs). Since 2021, he has been with STMicroelectronics, Pavia, where, within the framework of the Joint Research Laboratory "Studio di Microelettronica" between STMicroelectronics and the University of Pavia, he contributes to the development of analog in-memory computing solutions and PCM-based hardware accelerators for edge-AI applications.

Jacopo Bertolini received the bachelor's degree in computer engineering and the master's degree in electronic engineering.

His work focuses on low-power digital hardware design for RF signal processing and in-memory computing for edge-AI devices.

Riccardo Vignali received the master's degree from the University of Pavia, Pavia, Italy, in 2022. He is currently pursuing the Ph.D. degree from the University of Pavia in collaboration with STMicroelectronics, Pavia, working on analog mixed-signal design for compute-in-memory applications.

Luca Iannelli received the Ph.D. degree from the University of Pavia, Pavia, Italy, in collaboration with STMicroelectronics, Pavia, working on analog mixed-signal design for compute-in-memory applications.

Emanuela Calvetti received the master's and Ph.D. degrees in electronic engineering from the University of Brescia, Brescia, Italy, in 2003 and 2006, respectively.

She has been working with ST Agrate, NVM Design Team, Agrate Brianza, Italy, as a Digital Designer. She designed digital controllers for FTP memories, PCM memories, and in the last years, she has been focusing on the development of analog in-memory computing IPs.

Marco Pasotti received the degree in electronic engineering from Universitá degli Studi di Pavia, Pavia, Italy, in 1991.

In 1994, he joined STMicroelectronics, Agrate Brianza, Italy, where he collaborated on the design of digital and mixed analog/digital ICs for image acquisition and processing. In 1996, he was engaged in designing flash memory for analog applications, multilevel storage, and multimedia products. In 2005, he led the team responsible for developing cost-effective eNVM solutions, including embedded flash and PCM memories for smart power applications. Currently, he is a Senior Member of STMicroelectronics' Technical Staff. His research interests focus on high-performance embeddable NVM memories, analog in-memory computing, innovative technologies for non-volatile memories, and their application aspects.

Alessandro Cabrini was born in Pavia, Italy. He received the "Laurea" (summa cum laude) and Ph.D. degrees in electronic engineering from the University of Pavia, Pavia.

From 2004 to 2011, he was a Research Fellow at the Department of Electronics, University of Pavia, working on non-volatile storage devices based on flash and phase-change materials. From 2011 to 2019, he was an Assistant Professor at the Department of Electrical, Computer and Biomedical Engineering, University of Pavia, where he is currently an Associate Professor of Electronics. He has authored more than 100 papers in international journals and conference proceedings, and two book chapters. He holds six international patents. His research interests mainly focus on non-volatile memories and, in particular, on the design, characterization, and modeling of high-density storage devices, such as multilevel phase-change and flash memories.

Dr. Cabrini has been an Associate Editor of *IET Electronics Letters* and a Guest Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS. He acts as a reviewer for different international journals and conferences.

Eleonora Franchi Scarselli (Member, IEEE) received the M.S. degree in electrical engineering and the Ph.D. degree in electrical engineering and computer science from the University of Bologna, Bologna, Italy, in 1992.

From 1994 to 2004, she was a Research Assistant with the Faculty of Engineering, University of Bologna, where she has been an Associate Professor since 2005 and currently teaches the design of digital integrated circuits. She is a member of the Scientific Committee of the joint STMicroelectronics-ARCES Laboratory, Bologna. Her main research activities include architecture and circuits for low-power sensing and actuating IoT nodes and for in-memory computing based on phase-change memory.

Open Access funding provided by 'Universita di Bologna' within the CRUI CARE Agreement