npj | unconventional computing

Review

https://doi.org/10.1038/s44335-025-00044-2

Achieving high precision in analog in-memory computing systems

Piergiulio Mannocci ✉, Giacomo Larelli, Marco Bonomi & Daniele Ielmini ✉

Modern workloads challenge von Neumann architectures due to memory-processor data transfer. In-memory computing (IMC) enables in situ processing, with analog IMC (AIMC) based on resistive memories offering high-throughput and energy-efficient multiply-accumulate operations. Precision is limited by noise, device/circuit variations, and non-idealities. This work reviews error sources in AIMC and surveys mitigation strategies: bit slicing, residue number system, error correction codes, and mixed-precision iterative refinement, analyzing hardware implementations, overheads, and tradeoffs.

According to the von Neumann architecture (Fig. 1a), a computer consists of two essential parts, namely the central processing unit (CPU) and the memory unit[1]. The latter must support both instructions and data for the computation, which is executed in the CPU. With the massive increase of data and the widespread use of artificial intelligence (AI) in our modern digital society, memory and computing demands have seen an exponential increase[2] for which the processing time and energy consumption are largely limited by the data movement between the memory and the CPU[3–5]. Overcoming this fundamental gap of performance requires the introduction of novel computing paradigms, leveraging unconventional physical mechanisms such as quantum effects or photonics.

Among the novel computing paradigms, in-memory computing (IMC) aims at executing data processing within the memory unit, thus alleviating the memory bottleneck[6,7] (Fig. 1b). Various approaches and technologies have been proposed for IMC[8–10]. In general, IMC must rely on a precise, scalable, reliable, low-voltage/low-current memory device, which might be either a volatile or a nonvolatile memory technology. In addition to the conventional CMOS-based memories, such as static random-access memory (SRAM), dynamic random access memory (DRAM), and non-volatile flash memory, various emerging memory technologies have been studied in the last 25 years. These NVM technologies include resistive switching memory (RRAM), phase change memory (PCM), spin-transfer torque (STT) magnetic random-access memory (MRAM), and ferroelectric random access memory (FeRAM)[11]. Volatile and nonvolatile memory technologies enable IMC with various architectures relying on either single-core IMC[12–16] or multi-core IMC (Fig. 1c)[17–19].

Analog IMC (AIMC) was shown to be particularly effective in accelerating data-intensive algebraic operations, such as the matrix-vector multiplication (MVM), when implemented using crossbar arrays (CBAs) of resistive memory devices. Figure 1d shows a conceptual scheme of an in-memory MVM accelerator, also known as open-loop AIMC, where a voltage vector $\mathbf{v}$ is applied on the columns of a CBA programmed with a conductance matrix $\mathbf{G}$. Thanks to Ohm's and Kirchhoff's laws, acting as physical surrogates of multiply-accumulate (MAC) operations, a current vector $\mathbf{i}$ can be probed on the grounded CBA rows, namely:

$$\mathbf{i} = \mathbf{G}\mathbf{v}, \tag{1}$$

corresponding to the MVM of matrix $\mathbf{G}$ and vector $\mathbf{v}$. Thanks to the intrinsic parallelism provided by the CBA structure, the operation is performed in just one computational step or $\mathcal{O}(1)$, with a significant advantage over the typical $\mathcal{O}(N^2)$ computational complexity, where N is the matrix size, of MVM in digital processors. Open-loop AIMC has been experimentally demonstrated in a wide range of applications, including image processing[20,21], neural networks[18,19,22], and combinatorial optimization[23,24].

The AIMC concept can be extended to inverse operations such as inverse-matrix-vector multiplication (IMVM) by using the closed-loop scheme illustrated in Fig. 1e[25–27]. The CBA is now enclosed in an analog feedback loop using on-chip operational amplifiers (OAs) in negative feedback configuration, which gives rise to virtual ground nodes at the CBA rows. By injecting a current vector $\mathbf{i}$, the column voltages settle to match the same Ohm's and Kirchhoff's law of the MVM accelerator, i.e.:

$$\mathbf{v} = -\mathbf{G}^{-1}\mathbf{i}, \tag{2}$$

yielding the solution of the linear system described by the conductance matrix $\mathbf{G}$ and known term vector $\mathbf{i}$. In the last few years, several closed-loop AIMC circuits have been proposed for a wide range of applications, including 5G and 6G communications[28–31], ranking systems[32,33], machine learning[34–36], sensor fusion[30,37,38], combinatorial optimization[39,40] and general linear algebra[41–43], demonstrating superior performance with respect to digital systems, which typically operate at $\mathcal{O}(N^3)$ complexity.

IMC has been generally applied to AI accelerators, where the MVM operation is used to accelerate each layer of a deep neural network (DNN), such as state-of-the-art convolutional neural networks (CNNs) for image

Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy. ✉e-mail: piergiulio.mannocci@polimi.it; daniele.ielmini@polimi.it

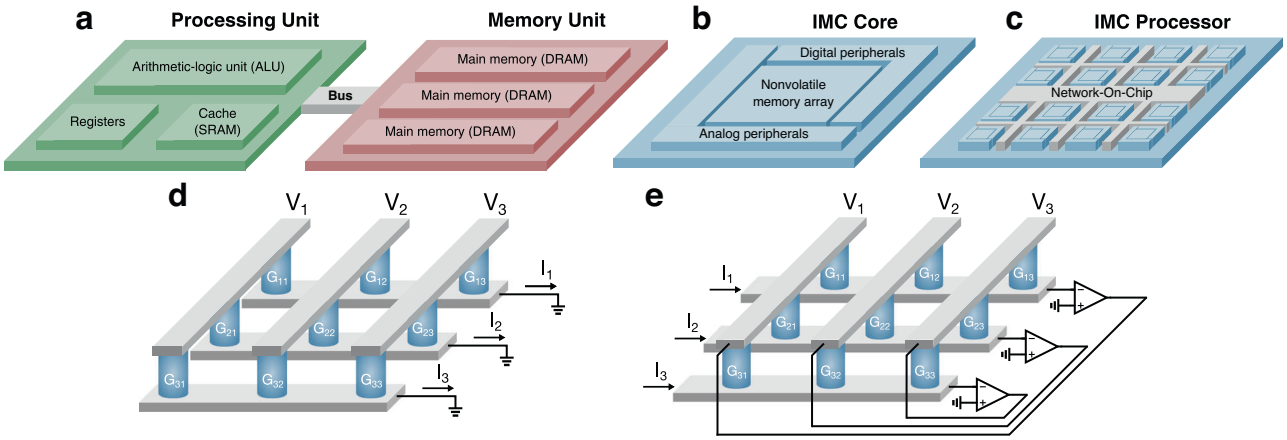

Fig. 1 | From von Neumann to in-memory computing. a Prototypical von Neumann architecture, where memory and processing units are physically separated and interconnected by a data bus. **b** IMC core, where computation happens directly within the embedded nonvolatile memory array. **c** IMC processor, composed of several IMC cores interconnected by a dedicated Network-On-Chip. **d** Open-loop IMC-based MVM circuit. **e** Closed-loop IMC-based IMVM circuit.

Table 1 | Mitigation techniques for analog computing errors

Error source	Impacted block	Impacted operations	Mitigation technique
Quantization	Memory, Analog/digital and digital/analog converters	MVM, IMVM	Slicing, Residue number systems
Variability Noise Faults	Memory, Readout circuit, Analog/digital and digital/analog converters	MVM, IMVM	Error correction codes
Analog non-idealities	Array, Readout circuit, Analog/digital and digital/analog converters	IMVM	Iterative refinement

recognition[18,44,45]. While DNNs are relatively robust with respect to errors and variations of IMC, other applications, such as scientific computing for quantum simulations[46,47], computational biology[48], genomics[49], mechanical analysis[50], as well as telecommunications[51] and robot kinematics[52], have limited tolerance with respect to computing inaccuracies. As a result, various types of mitigation techniques and architectures have been proposed to achieve high precision in limited-precision IMC systems. Table 1 summarizes the main mitigation schemes, including:

- slicing techniques, where the operands (coefficients and input values) are decomposed in multiple components in the digital (binary) or analog domain,
- residue number system (RNS), where the operands are decomposed as the modulo of chosen coprime numbers before carrying out multiplication and summation, finally reconstructing the results via the Chinese remainder theorem (CRT),
- error correction schemes, where a digital or analog error correction code (ECC) is adopted to detect and/or correct errors during IMC, and
- iterative refinement (IRF), where the computation results of a low-precision IMC unit are iteratively corrected by a higher precision unit in the so-called mixed-precision approach.

This work provides a detailed study of these schemes, providing a description and a comparison of various approaches in terms of accuracy and their tradeoffs with energy efficiency, area, and performance.

The rest of this work is organized as follows. Sec. "Error sources in AIMC systems" illustrates the main nonidealities and error sources of AIMC systems. Sec. "Analog and bit slicing" provides an overview of the slicing techniques, also known as *bit-width decomposition*, aimed at alleviating the impact of memory quantization. Sec. "Residue number system" describes RNS as a closely-related alternative approach to the slicing techniques for the mitigation of the requirements of converter precision. Sec. "Error correction codes" illustrates ECCs as a means to reject the impact of noise and variability on analog computing primitives. Sec. "Iterative refinement" addresses the IMVM-specific issues of circuit non-ideality and problem sensitivity by IRF. Finally, overheads and tradeoffs of the reviewed mitigation techniques are discussed in Sec. "Discussion".

Error sources in AIMC systems

The lower latency and energy of AIMC come at the cost of reduced precision compared to the high precision of floating-point, von-Neumann-based digital processors[8,53]. Figure 2 highlights the main error sources in a typical AIMC primitive, including memory device and array parasitics, various nonidealities of the periphery circuit blocks, such as the analog/digital converter (ADC), digital/analog converter (DAC), as well as deterministic and stochastic variations of the readout circuitry. For instance, resistive memory devices typically suffer from quantization, i.e. limited bit precision[11,54,55], programming variability[56,57], i.e. deviation of the conductance from its nominal value at program time, drift, i.e. conductance increase or decrease over time[58], and read variability and noise[59] (Fig. 2a). Parasitic elements at the array level that typically affect the computing accuracy include the line resistance, which causes a voltage (IR) drop along columns and rows of memory arrays[60–64], the line capacitance, which affects computation latency[65], and device yield, which may result in unrecoverably stuck memory cells[66] (Fig. 2b). Analog currents output by the memory array are typically preprocessed by dedicated readout circuitry before conversion, which may suffer from offset, variability, and nonlinearity[67] (Fig. 2c). Circuit nonidealities and operational conditions play a significant role in closed-loop AIMC, where the OA finite gain and bandwidth contribute to the computing accuracy and latency, respectively, which are typically a function of the inverse problem sensitivity measured by the condition number κ[27,68]. Finally, ADCs and DACs inherently contribute to computation error owing to their limited bit-precision, differential and/or integral nonlinearity, and noise floor[69] (Fig. 2d). Figs. 2e-h particularly report experimental data for nonidealities in resistive memory devices, including quantization and

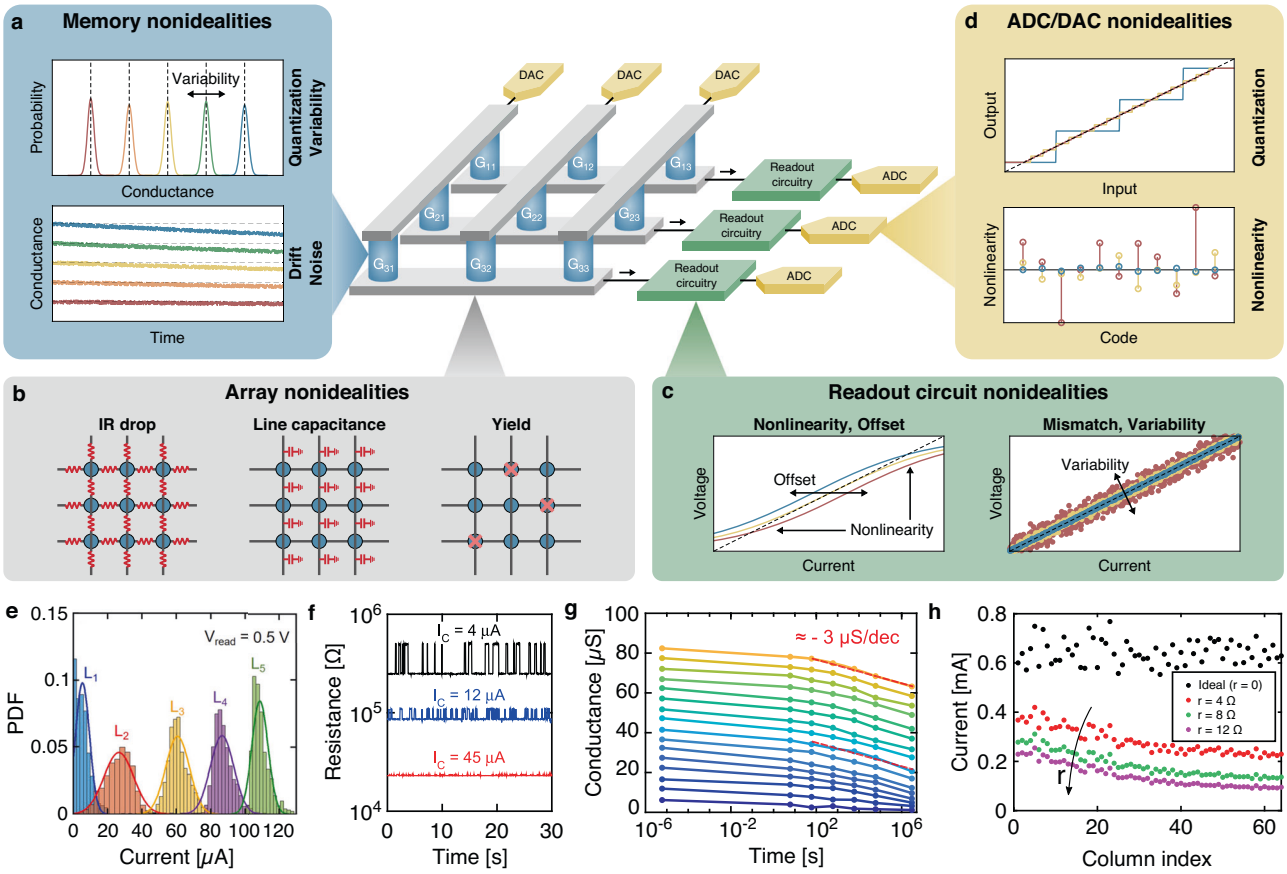

Fig. 2 | AIMC error sources. Overview of the main error sources impacting computation accuracy in AIMC, including (**a**) memory quantization, variability, and drift, (**b**) array-level IR drop, line capacitances, and device yield, (**c**) nonlinearity, offset, mismatches, and variability of readout circuits, and (**d**) nonlinearity and quantization of ADCs and DACs. Experimental data for (**e**) memory quantization, variability, and (**f**) noise in RRAMs, (**g**) conductance drift in PCMs, and (**h**) simulations of IR drop impact on current outputs of crosspoint arrays of resistive memory devices. **e** Reproduced with permission from[70], licensed under a Creative Commons Attribution License. **f** Reproduced with permission from[59]. Copyright 2015 IEEE. **g** Reproduced with permission from[72], licensed under a Creative Commons Attributions License. **h** Reproduced with permission from[63]. Copyright 2022 IEEE.

programming variability (Fig. 2e) and random telegraph noise (Fig. 2f) in resistive random access memories (RRAMs)[70,71], conductance drift in phase change memories (PCM, Fig. 2g)[72], and simulations of IR drop impact on crosspoint arrays of resistive memory devices (Fig. 2h)[63]. With respect to the ideal MVM of Eq. (1), the summation current in a real AIMC system is thus given by:

$$\tilde{\mathbf{i}} = (\mathbf{G} + \delta\mathbf{G}) \times (\mathbf{v} + \delta\mathbf{v}) + \delta\mathbf{i}, \tag{3}$$

where $\delta\mathbf{G}$, $\delta\mathbf{v}$, $\delta\mathbf{i}$ describe errors on the conductance matrix (e.g., memory and array nonidealities), input voltage vector (e.g., DAC nonidealities), and output current vector (e.g., readout circuitry and ADC nonidealities). Notably, errors may either be static, e.g., conductance mismatches arising from programming variability and IR drop or DAC offsets, or dynamically changing over time, e.g., as a result of read variability, conductance drift, and noise, thus requiring different compensation techniques. Similarly, with respect to the ideal IMVM of Eq. (2), the real IMVM operation performed by an AIMC system reads:

$$\tilde{\mathbf{v}} = -(\mathbf{G} + \delta\mathbf{G})^{-1} \times (\mathbf{i} + \delta\mathbf{i}) + \delta\mathbf{v}. \tag{4}$$

Several techniques have been proposed to mitigate the impact of device- and array-level nonidealities, including hardware-aware training of neural networks[62,73], differential schemes for drift cancellation[58,72], and mitigation schemes for IR drop[61,63,64], allowing partial restoration of the nominal bit-precision of the memory cell.

Analog and bit slicing

Figure 3a shows a conceptual scheme of the slicing technique, which allows the processing of high bit-precision operations by computing modules with reduced bit-precision. Each operand $\mathbf{o}$ is represented by a linear combination of weighted *slices*, namely:

$$\mathbf{o} = w_0\mathbf{o}_0 + w_1\mathbf{o}_1 + \ldots + w_k\mathbf{o}_k = \sum_{i=0}^{k} w_i\mathbf{o}_i, \tag{5}$$

where $\mathbf{o}_i$, w_i are the i-th slice and slice weight, respectively, and the slicing basis $\{w_0, w_1, \ldots, w_k\}$ is chosen such that each slice $\mathbf{o}_i$ can be represented using fewer bits than the original operand $\mathbf{o}$. Under the slicing framework, the MVM operation between matrix $\mathbf{A}$ and vector $\mathbf{b}$ reads:

$$\mathbf{Ab} = \sum_{i=0}^{k_1} w_{\mathbf{A},i}\mathbf{A}_i \sum_{j=0}^{k_2} w_{\mathbf{b},j}\mathbf{b}_j = \sum_{i=0}^{k_1}\sum_{j=0}^{k_2} w_{\mathbf{A},i}w_{\mathbf{b},j}\mathbf{A}_i\mathbf{b}_j, \tag{6}$$

where each of the partial operations $\mathbf{A}_i\mathbf{b}_j$ is therefore an MVM between matrix $\mathbf{A}_i$ and vector $\mathbf{b}_j$ of reduced bit-precision. Among the most common choices for slicing bases are the *binary* and *unary* encoding, corresponding to bases $\{2^0, 2^1, \ldots, 2^k\}$ and $\{1, 1, \ldots, 1\}$ respectively, owing to the relative ease

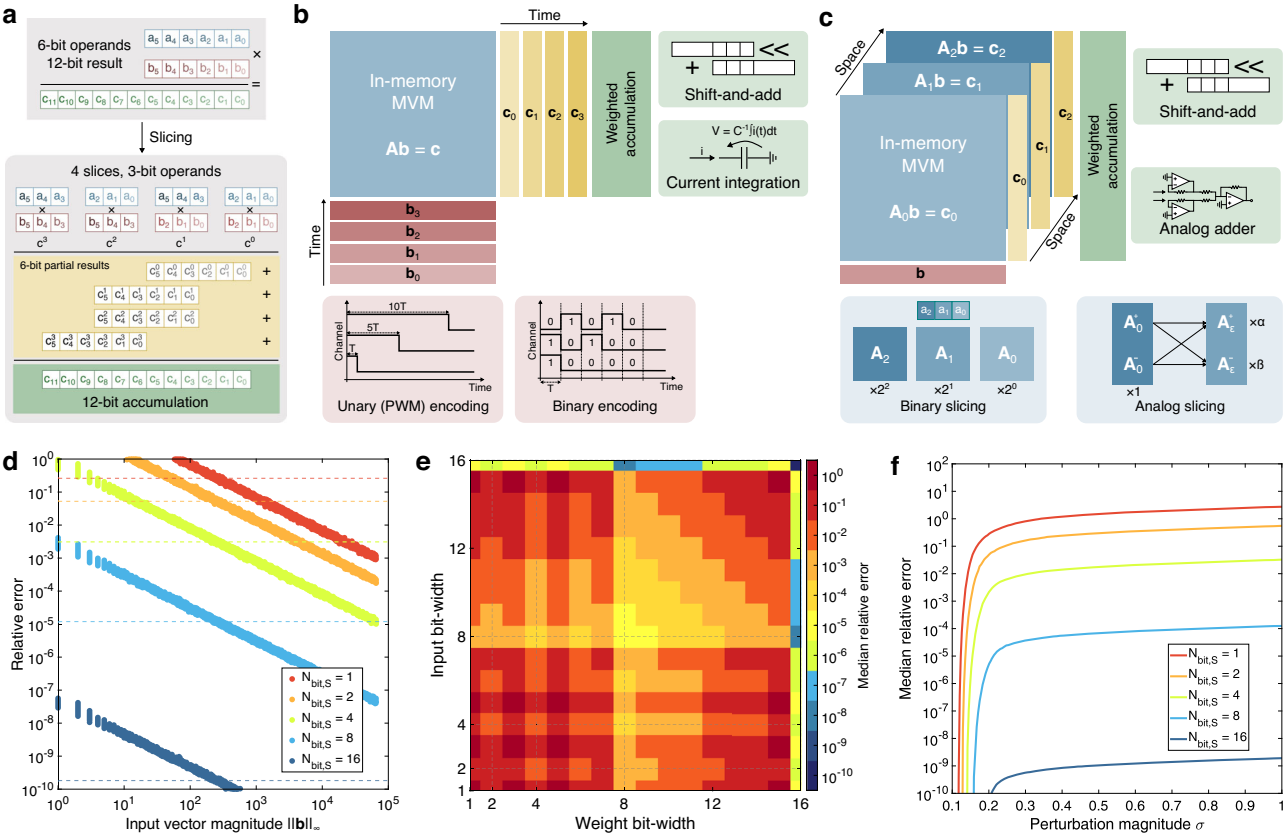

Fig. 3 | Slicing in AIMC. a Concept scheme of bit slicing technique for integer multiplication, where high-bit-width operands are decomposed in multiple low-bit-width slices. Partial results obtained by each sub-operation are shifted to reconstruct the equivalent significance of the processed slices and finally accumulated to reconstruct the target result. **b** Input vector slicing, where slices are typically processed sequentially by the same AIMC core. Different encoding schemes are used, including unary, also known as PWM, where values are represented by pulses of proportional duration and fixed amplitude, or binary encoding, where fixed-duration time slots are associated with different bit-significances. Each encoding must be matched to a conforming readout scheme for best performance, namely voltage-conversion followed by S&A for binary encoding and current integration-based schemes for PWM. **c** Owing to their stationarity, matrix slices are conversely encoded in space over different portions of the same memory array or different sub-arrays. Slice weights can be chosen as power-of-two multipliers, such as binary or quaternary encoding, or by dynamic computation at programming time in an analog fashion. In the first binary slicing approach, weight reconstruction is performed by digital-friendly S&A techniques. Conversely, the analog slicing approach is well-suited for reconstruction by analog adders with tunable gain. **d**–**f** Simulation results for a bit-sliced 128 × 128 MVM with 16-bit operands in a real IMC system for (**d**, **e**) $\sigma = 0.1$ LSB and (**f**) increasing values of σ. **d** Relative error as a function of the input vector magnitude for different values of input and weight slice bit-width. Dashed lines highlight the corresponding median relative error. **e** Median relative error as a function of the input and slice bit-width. **f** Median relative error as a function of σ.

of implementation of partial results accumulation. Binary encoding allows shift-and-add (S&A) to be used for implementing the weighted accumulation of slices. Conversely, digital counters can be used to efficiently combine partial results under a unary encoding.

Figure 3b shows a conceptual scheme of input slicing in an AIMC primitive, where input slices are processed sequentially in different time steps. Among the most common input slicing bases for AIMC are unary or pulse-width modulation (PWM) encoding, where different input values are represented by pulses of varying duration at fixed amplitude[13,74], and binary encoding, where timeslots of the input waveform represent different bit-significances of the input word[75], allowing reduction of DAC precision requirements to 1-bit[76]. To efficiently reconstruct the full-precision output, partial results provided by the AIMC macro must be accumulated in a weighted fashion by a conforming readout circuitry. In binary encoding schemes, partial current vectors are typically converted to voltage by transimpedance amplifiers (TIAs)[21,77], digitized by ADCs, and accumulated by S&A. Conversely, PWM encoding is well-suited for accumulation by current integration-based converters, either realized by dedicated column capacitances[71] or mixed-signal time-to-digital converters[13].

A similar slicing can be applied to the MVM matrix as shown in Fig. 3c, where different slices are spatially mapped either to different columns or regions of a memory array[75,78] or different memory arrays in a multi-core approach[21,79]. On the one hand, slice replication onto multiple cores enables the processing of different input slices in the same computational cycle[76,80], albeit at the expense of increased hardware overhead. Conversely, spatially parallelizing the computation within the same core allows for a reduction in inter-tile communication overheads, at the cost of reduced array utilization efficiency. Common slicing bases for matrix elements include binary[75] or larger bases such as quaternary encoding[76,80], i.e. $w_k = \{4^0, 4^1, \ldots, 4^k\}$, by exploiting the multi-level cell (MLC) capability of resistive memory devices. Similar to binary slicing, quaternary encoding is compatible with S&A techniques, with two shifts being performed before each accumulation step. Fine-grained conductance tunability of memory devices[56,81] can be further leveraged to allow using analog values for the slicing basis, with weighted accumulation performed by analog adders with tunable gain[21,79]. In this analog slicing approach, the slice weight is dynamically computed at program time by first calculating the error matrix between the target and programmed conductance matrices and subsequently rescaling the error matrix to fit the available conductance window of memory devices, allowing significant precision increase[21] as well as mitigation of device variability[79].

A major drawback of slicing techniques lies in *noise amplification*, namely the unwanted magnification of computing errors during the

reconstruction process. Considering a real IMC system subject to non-idealities, the real MVM operation between matrix **A** and vector **b** reads:

$$\tilde{\mathbf{c}} = \mathbf{Ab} + \mathbf{n} = \mathbf{c} + \mathbf{n}, \tag{7}$$

where $\mathbf{n} \sim \mathcal{N}(0, \sigma^2)$ models random and deterministic errors detailed in Sec. "Error sources in AIMC systems", Eq. (3). In the bit-slicing framework, a high-precision result is thus obtained by IMC-computed low-precision results as:

$$\tilde{\mathbf{c}} = \sum_{i=0}^{k} w_i(\mathbf{c}_i + \mathbf{n}_i) \simeq \mathbf{c} + \sum_{i=0}^{k} w_i\mathbf{n} = \mathbf{c} + \alpha\mathbf{n}, \tag{8}$$

where α is the *noise amplification factor*, such that the relative error reads:

$$\varepsilon = \frac{\| \tilde{\mathbf{c}} - \mathbf{c} \|}{\| \mathbf{c} \|} = \alpha\frac{\| \mathbf{n} \|}{\| \mathbf{c} \|} = \alpha\frac{\sigma\sqrt{N}}{\| \mathbf{c} \|} > \frac{\sigma\sqrt{N}}{\| \mathbf{c} \|}, \tag{9}$$

where N is the matrix size. Figure 3d shows the relative error as a function of the input vector magnitude $\|\mathbf{b}\|$ for a bit-sliced 128 × 128 MVM with 16-bit operands **A** and **b**, highlighting a dependence on the slice bit-width $N_{bit,S}$ owing to the different weighting factors w_i. For example, considering 1-bit slices with weights $\{2^0, 2^1, \ldots, 2^{15}\}$, the overall amplification factor is equal to $\alpha = 2^{16} - 1$. Conversely, by using wider 8-bit slices with weights $\{2^0, 2^7\}$, the amplification factor is reduced to $\alpha = 2^7 + 1 \ll 2^{16}$, thus resulting in significantly reduced relative error. The bit-widths of both the input and weight slices equally contribute to determining the noise amplification, as illustrated by Fig. 3e, although in a non-monotonic fashion. Conversely, the overall relative error exhibits a linear dependence on σ as shown in Fig. 3f.

Residue number system

RNS is a number decomposition technique in which integers are represented by division remainders against a selected set of moduli[82], i.e.:

$$\mathbf{o} = \{|\mathbf{o}|_{m_1}, |\mathbf{o}|_{m_2}, \ldots, |\mathbf{o}|_{m_k}\}, \tag{10}$$

where m_i is the i-th integer of the moduli set, and $|\mathbf{o}|_m$ (**o** modulo m) denotes the remainder of the integer division between elements of operand **o** and m, i.e.:

$$|\mathbf{o}|_m = \mathbf{o} - m \times \mathbf{n}, \tag{11}$$

where **n** is the vector of *folding integers*, such that for each element o_j of **o** it holds $o_j < m$. A number represented under the RNS framework can be reconstructed by leveraging the CRT or Sunzi Theorem[83],

$$\mathbf{o} = \left| \sum_{i=1}^{k} |\mathbf{o}|_{m_i} \cdot M_i \cdot T_i \right|_M, \tag{12}$$

where $M = \prod_i m_i$ is the *dynamic range*, $M_i = M/m_i$, and T_i is the multiplicative inverse of M_i under the m_i modulo, i.e. $|M_i T_i|_{m_i} = 1$. When all m_i are chosen to be pairwise coprime, each number in the range $[0, M)$ is uniquely identified by a set of remainders, thus guaranteeing an exact reconstruction. Figure 4a schematically illustrates the RNS representation of numbers from 0 to 29 using moduli {6,5}, where each number is uniquely associated to a remainder pair, e.g., 8 corresponds to {2,3}.

RNS can be used to decompose operations on high bit-precision operands into partial operations on operands with reduced bit-precision. For an n-moduli set, the MVM between a weight matrix **A** and a vector **b** is represented under RNS as:

$$\mathbf{Ab} = \{|\mathbf{Ab}|_{m_1}, |\mathbf{Ab}|_{m_2}, \ldots, |\mathbf{Ab}|_{m_n}\}. \tag{13}$$

Thanks to the compatibility of modulo operation with addition and multiplication, each partial operation can be rewritten as:

$$|\mathbf{Ab}|_{m_i} = ||\mathbf{A}|_{m_i}|\mathbf{b}|_{m_i}|_{m_i}, \tag{14}$$

which reduces the required bit-precision to $\log_2(m_i)$ for memory and input values while simultaneously allowing to obtain computation results at $\log_2(M) > \log_2(m_i)$ precision. Residuals provided by the low-precision moduli are combined to identify the unique reconstruction candidate in the $\mathbb{N}^k$ space spanned by the set of moduli[84]. Fig. 4b shows the number of required moduli while Fig. 4c shows their corresponding bit-width as a function of the matrix size and of the overall bit-width of partial multiplications $a_{ij} \times b_j$, where a_{ij} and b_j are elements of the matrix **A** and vector **b** respectively. Figure 4d shows a conceptual scheme of an AIMC-based RNS implementation[85], where partial operations are performed by AIMC cores operating at reduced bit-precision, whereas reconstruction is performed in the digital domain by CRT.

RNS has been widely explored for implementing power-efficient digital signal processors (DSPs) in embedded systems[86–89], as well as for accelerating DNN inference in fully-digital[90–92] and GPU-based systems[93]. More recently, RNS has been proposed for digital near-memory[94] and in-memory oriented architectures[95,96], with core MAC operations realized by digital logic. Demirkiran et al.[85] first explored the possibility of implementing RNS in AIMC systems, with specific application to photonic MVM accelerators for DNN training[97]. With respect to slicing techniques, RNS has the compelling feature of reducing the required precision of output ADC in AIMC systems to the same precision of the memory devices and DACs[85].

On the other hand, RNS suffers from high sensitivity to computational errors, owing to the large amplification applied to the residuals during reconstruction by CRT[84], similar to noise amplification in bit slicing. Various techniques have been proposed to improve RNS robustness by leveraging information redundancy. Watson and Hastings[98] proposed to augment the moduli set with additional redundant moduli, as shown in Fig. 4e. Partial results from different moduli can then be grouped into several distinct sets of moduli, each providing a potential reconstruction via CRT. Candidate solutions are then selected by a suitable decision logic such as majority voting[85,99,100], typically allowing rejection of inaccurate results from few faulty units with large, burst-type errors.

A more suitable alternative for the inaccuracy scenario in AIMC, where IMC cores are typically characterized by small, distributed errors, is represented by the modulo-level redundancy proposed by Xiao et al.[84]. By lifting the requirement for moduli in the selected set to be pairwise coprime, common divisors provide redundant information implicitly shared across the different moduli, enabling both accurate and approximate reconstruction by identifying the nearest valid remainder tuple following geometrical considerations. The added complexity of the reconstruction process is entirely handled by the digital logic, while the AIMC cores continue to operate on moduli of slightly larger bit-width due to the common divisor. Fig. 4f schematically illustrates the added redundancy for a RNS system with moduli {15,6}, sharing a common divisor $k = 3$. With respect to the {5,6} RNS system of Fig. 4a, the redundant system can tolerate inaccuracies up to $\pm k/4 = \pm 0.75$ in the computation of residuals without incurring in reconstruction error. Figure 4g shows the beneficial impact of non-coprime moduli in terms of the median relative error as a function of the AIMC system perturbation magnitude σ, for an RNS-based 128 × 128 MVM between 16-bit operands, for an RNS system operating with 6-bit moduli and zero redundancy, and redundant RNS systems with increasing common divisors $k = 2, 4, 8, 16$, corresponding to 7-bit, 8-bit, 9-bit, and 10-bit moduli respectively. Note that RNS systems typically exhibit abrupt transitions between optimal low-error regions and high-error regimes, in stark contrast with the linear dependence on σ observed in the bit-slicing framework, owing to the nonlinear reconstruction process exploiting modulo-based operations in place of linear S&A.

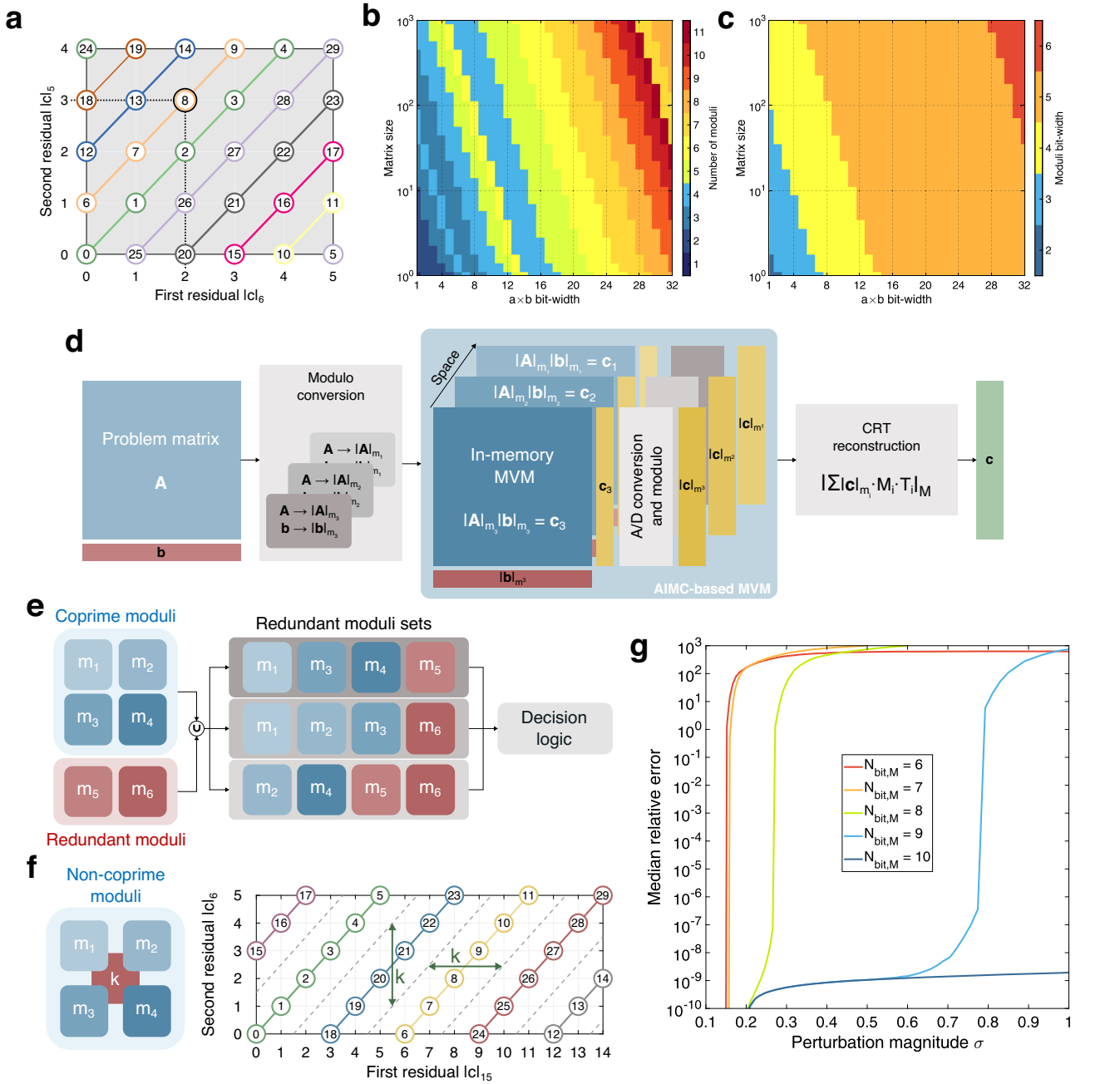

Fig. 4 | RNS-based AIMC architectures. a Concept scheme of RNS-based representation of integers {0, …, 29} using the moduli set {5,6}. Each integer is uniquely identified by a set of remainders with respect to the moduli set, with e.g., 8 corresponding to {2,3}. **b** Number of required moduli and (**c**) corresponding moduli bit-width as a function of the bit-width of partial multiplications $a_{ij} \times b_j$ and matrix size. **d** Concept scheme of in-memory RNS. Remainders of the problem matrix **A** and input vector **b** against a set of coprime moduli $\{m_1, m_2, m_3\}$ are first computed and forwarded to separate AIMC macros. Partial analog results are converted to the digital domain under the corresponding modulo. In the final reconstruction stage, the CRT identifies the unique number associated with the computed remainders. **e** The moduli set can be augmented with additional coprime moduli to create redundant moduli sets. Multiple candidate results are reconstructed by randomly grouping the available moduli and forwarded to a decision logic, typically implemented by majority voting, to mitigate the impact of burst-type errors. **f** Alternatively, non-coprime moduli sharing a common divisor k can be selected, allowing to tolerate small, distributed inaccuracies on the output of each modulo computation. **g** Median relative error for an RNS-based 128 × 128 MVM with 16-bit operands as a function of the perturbation magnitude σ, for increasing values of redundancy factor k from $k = 1$ (6-bit moduli) up to $k = 16$ (10-bit moduli), highlighting a nonlinear dependence on σ.

Error correction codes

ECCs were first proposed in 1950 by Hamming[101] as a way of controlling errors in data transmission over unreliable communication channels. Figure 5a shows a conceptual scheme of an ECC system where a payload p containing sensitive data is first processed by an *encoder*, which computes r error correction (EC) bits. The *codeword* formed by the payload data and the EC bits is processed by the target operation $f(x)$, ideally providing an exact result $f(p|r)$. Different sources of inaccuracy, such as computation error, noise, or transmission faults, typically result in an inaccurate operation $\tilde{f}(x) = f(x) + e$ being performed in place of $f(x)$, such that the obtained output is $\tilde{f}(p|r) = f(p|r) + e$. A *decoder* block processes the inaccurate output codeword, leveraging the redundant information carried by the EC bits to detect and, eventually, correct errors, thus allowing the extraction of the ideal result of the computation, namely $f(p)$[102]. Several different ECC

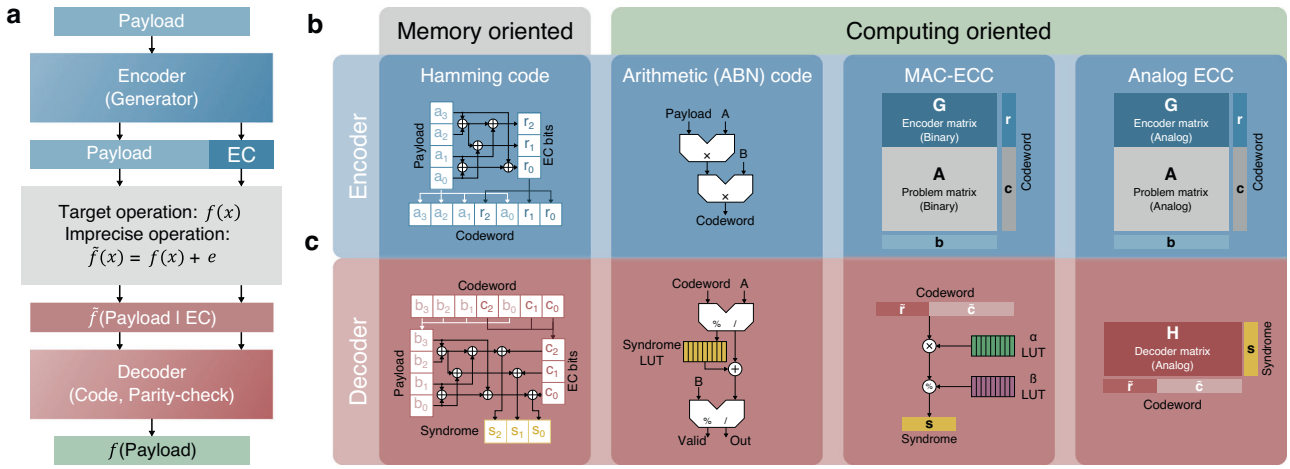

Fig. 5 | Error correction code system. a Concept scheme of an ECC system, where an encoder generates EC bits which are appended to the to-be-protected payload to form a codeword. Inaccuracies in the target operation result in a corrupted codeword, which is forwarded to a decoder exploiting the redundancy carried by EC bits to detect and correct errors, yielding the accurate computation output. **b** Encoder and (**c**) decoder schemes for AIMC, including memory-oriented Hamming codes for bit-level protection, and computing-oriented approaches, namely arithmetic ABN codes, MAC-ECC, and analog ECC, for protection of MVM outputs.

variants have been proposed, mainly divided into fixed-size block codes and arbitrary-length convolutional codes[103], providing single-bit[101] and multi-bit[104–106] correction capability.

Figures 5b, c highlight the main (b) encoder and corresponding (c) decoder structures proposed in the latest years to realize AIMC-specific ECCs. Conventional *memory-oriented ECC* schemes typically used in DRAM can be employed to protect resistive memories at the level of individual bits[107,108], where the required encoding and decoding logic can be realized either in the digital domain by XOR gates[107] or using fully in-memory approaches via stateful logic gates based on resistive memory devices[108]. On the other hand, *computing-oriented ECCs* directly operate on the MVM output, providing higher compatibility with the AIMC framework[109–114]. Feinberg et al.[109] first proposed an AIMC-specific implementation of AN and ABN codes, relying on modular computation and divisibility properties to detect and correct errors at the computing level. In AN codes, input data are digitally multiplied by a known integer A to form the codeword, which is then forwarded to a linear operation such as MVM. ABN codes further provide correction capability by including a second multiplier B and a more complex decoder scheme[109] providing additive corrections to the decoded codeword. Li et al.[110] introduced MAC-ECC, a computing-oriented scheme based on modified Berlekamp codes for the protection of partial sums on binary data matrices, exploiting a modulo-based digital decoder to ascertain computation validity and perform error detection. To reduce the associated digital logic overhead and allow protection of continuously-valued analog conductances, Roth et al.[111,112] introduced analog ECC relying on MVM-based encoder/decoder pairs, which can be embedded in the AIMC primitive with reduced area overhead. Li et al.[66] experimentally demonstrated the effectiveness of analog ECC for single-error correction in the event of noise injection and stuck-on/off devices, with ECC-based primitives achieving nominal accuracy on the MNIST classification task. Multiple-error-correcting codes have also been proposed[114] to extend further the capability of analog ECC for AIMC cores, enabling resilience from multiple faulty devices and error sources, typically at the cost of increased redundancy[113].

Iterative refinement

Inverse problems arise naturally in many applications, including machine learning[26], telecommunications[28], autonomous vehicles[30], genomics[49], and robotics[115]. However, their numerical solution shows limited robustness to inaccuracies owing to the magnification of algebraic errors[116]. The sensitivity of an inverse problem $\mathbf{Ax} = \mathbf{b}$ to perturbations in both the coefficient matrix $\mathbf{A}$ or constant vector $\mathbf{b}$ is typically controlled by the condition number κ, namely the ratio of the maximal and minimal singular values σ_1, σ_n of $\mathbf{A}$, i.e. $\kappa = \sigma_1/\sigma_n$. As a common rule of thumb, up to $\log_{10}(\kappa)$ digits of precision are lost if no mitigation techniques are adopted[116]. To improve the solution accuracy of inverse problems, iterative refinement (IRF) was first proposed by Wilkinson in 1963[117,118] in the context of linear systems, and subsequently extended to more complex inverse computations such as linear regression and general least squares[119,120] and matrix decomposition[121].

IRF is based on the iterative computation of a sequence of candidate solutions $\{\mathbf{x}^{(k)}\}$ which converges to the exact solution $\mathbf{x}$ of the linear system $\mathbf{Ax} = \mathbf{b}$. Figure 6a conceptually illustrates the workflow of IRF. Starting from a solution candidate $\mathbf{x}^{(k-1)}$, IRF computes the residual $\mathbf{r}^{(k)}$ as:

$$\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k-1)}. \tag{15}$$

A correction $\tilde{\mathbf{x}}^{(k)}$ is then computed by solving a linear system over an inaccurate matrix $\tilde{\mathbf{A}}$ and known term $\mathbf{r}^{(k)}$, i.e.:

$$\tilde{\mathbf{x}}^{(k)} = \tilde{\mathbf{A}}^{-1}\mathbf{r}^{(k)}, \tag{16}$$

such that a new solution candidate is obtained as:

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \tilde{\mathbf{x}}^{(k)} = \sum_{i=0}^{k} \tilde{\mathbf{x}}^{(i)}, \tag{17}$$

Figures 6b–d show the relative solution error as a function of the iteration number for 50×50 linear systems, and varying (b) residual computation and (c) inaccurate linear system solution accuracies, and (d) linear system condition number, highlighting an exponential improvement with each iteration. Notably, the IRF solution precision is ultimately limited by the residual computation accuracy $N_{bit,H}$ of Eq. (15) (Fig. 6b), whereas the accuracy of the approximate linear system solution $N_{bit,L}$ typically influences the convergence rate (Fig. 6c), i.e. the overall number of iterations required to meet a desired tolerance. As a consequence, IRF-based systems typically employ units with different precisions to implement Eqs. (15), (16), with high-precision (HP) unit used for residual computation, and low-precision (LP) unit providing linear system solutions with reduced computational complexity at each iteration. Finally, both the convergence rate and ultimate precision are influenced by the linear system condition number κ as highlighted in Fig. 6d, both typically exhibiting $\mathcal{O}(\kappa)$ complexity[122].

The possibility of combining the high accuracy of IRF with the high parallelism of AIMC was first proposed by Richter[123]. In[49], Le Gallo et al.

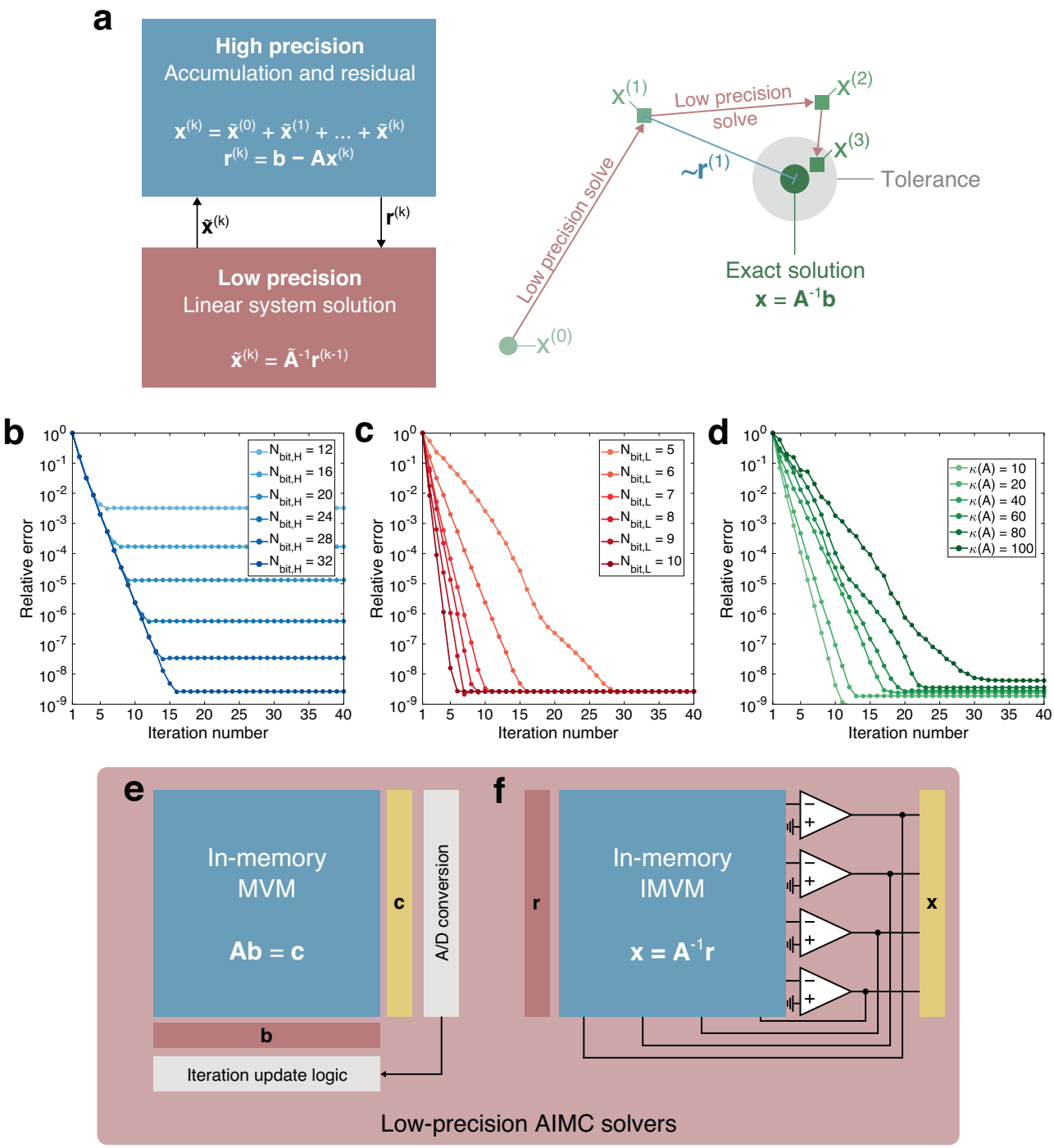

Fig. 6 | Iterative refinement for linear systems solutions. a At each iteration, an efficient low-precision unit computes a correction of the solution $\tilde{x}^{(k)}$ by solving the linear system associated with the residual which is, instead, computed by a high-precision unit. The algorithm takes advantage of the high-precision unit to increase the precision of the numerical solution, whereas the demanding computation is delegated to the low-precision unit. The process stops when the target tolerance is achieved on the residual value. **b**–**d** Relative error as a function of the iteration number for 50 × 50 linear systems for (**b**) 6-bit linear system solver and increasing high-precision residual computation from 12-bit to 32-bit ($\kappa = 50$), (**c**) 32-bit residual computation and increasing low-precision solver from 5-bit to 10-bit ($\kappa = 50$), and (**d**) 6-bit linear system solver, 32-bit residual computation unit and increasing linear system condition number from $\kappa = 10$ to $\kappa = 100$. **e**–**f** Possible implementations of the LPU by (**e**) iterative open-loop AIMC-based MVM, or (**f**) one-shot closed-loop AIMC-based IMVM.

realize the HP unit with a digital floating-point processor, whereas the LP IMVM solver is composed of a CBA-based MVM accelerator iteratively operated under a Krylov subspace linear solver as shown in Fig. 6e, namely conjugate gradient (CG). By combining an inner LP iterative solver based on low-power, highly-parallel CBAs and outer digital-based HP residual computation, an accuracy of 10^{-6} was demonstrated in solving well-conditioned linear systems described by prototype covariance matrices. By implementing the iterative update circuitry for the inner solver at the analog level, Li et al.[124] further demonstrated up to 10^{-12} solution accuracy for time-evolving partial differential equations. Feinberg et al.[125] further propose replacing the iterative inner solver with the one-step, IMC-based IMVM solver[25] shown in Fig. 6f to further increase the solution throughput.

IRF is known to exhibit a slow convergence rate or divergence for ill-conditioned linear systems[126]. Kalantzis et al.[127,128] proposed a

Table 2 | Representative works on error mitigation in AIMC systems and quantitative benchmarks

Authors	Year	Technique	Main result
Hu et al.[77]	2016	Slicing	7-bit computation on 4-bit primitive by binary slicing (+3 bits)
Shafiee et al.[76]	2016	Slicing	16-bit computation by binary matrix and input slicing (+15 bits)
Ankit et al.[80]	2019	Slicing	16-bit computation by binary matrix and input slicing (+15 bits)
Pedretti et al.[79]	2021	Slicing	11-bit weight precision on 4-bit memory by analog slicing (+7 bits)
Le Gallo et al.[75]	2022	Slicing	Binary slicing optimization for bit-width and drift mitigation
Khaddam-Aljameh et al.[13]	2022	Slicing	7-bit input slicing by PWM with time-to-digital readout
Jiang et al.[74]	2022	Slicing	Analog input slicing with capacitive readout and PWM generation
Song et al.[21]	2024	Slicing	Arbitrary precision on 3-bit memory by analog slicing (≈ +13 bits)
Salamat et al.[94]	2018	RNS	Simulated 16-bit accuracy on 6-bit analog CBA (+10 bits)
Roohi et al.[95]	2021	RNS	Simulated 16-bit accuracy on 5-bit digital CBA (+11 bits)
Demirkiran et al.[85]	2024	RNS	FP32 inference accuracy on 6-bit analog primitives (≈ +18 bit)
Niu et al.[107]	2012	ECC	Bit-level CBA protection by Hamming code-based ECC
Feinberg et al.[109]	2018	ECC	16-bit compute-level protection by ABN ECC
Li, Roth et al.[66]	2020	ECC	Correction of outlying errors by fully-analog ECC
Li, Read et al.[110]	2022	ECC	In situ analog/digital AN code-based ECC for MSB correction
Parrini et al.[136]	2024	ECC	4-bit weight protection by analog ECC
Le Gallo et al.[49]	2018	IRF	10^{-6} solution accuracy with 4-bit AIMC primitive (+16 bits)
Feinberg et al.[125]	2021	IRF	In situ preconditioning by AIMC IMVM
Kalantzis, Gupta et al.[127]	2021	IRF	AIMC-preconditioned linear system solution by Richardson iteration
Wu et al.[129]	2023	IRF	AIMC-tailored iterative update algorithm
Kalantzis, Squillante et al.[128]	2023	IRF	AIMC-preconditioned linear system solution by GMRES algorithm
Li, Xue et al.[124]	2025	IRF	10^{-10} accuracy with fully-analog iterative AIMC solver

preconditioned Richardson iterator for the outer IRF algorithm, demonstrating improved robustness in the solution of sparse linear systems. Preconditioners may also be embedded directly within the AIMC primitives[41] to mitigate the impact of ill-conditioning on the low-precision solver. Most recently, Wu et al.[129] formulated a stable IRF outer algorithm with improved robustness and superior convergence performance with respect to vanilla IRF.

Discussion

While each of the presented methodologies has been successfully implemented to achieve the results summarized in Table 2, precision improvement in AIMC systems exhibits tradeoffs in terms of complexity, area, and digital overhead. Slicing methods allow the representation of high-bit-precision operands by relatively low-precision memory devices and data converters. Partial results obtained by low-precision computations can be linearly combined in either the digital domain by S&A or in the analog domain by leveraging current integration on capacitive readout systems. Partial operations can be executed in parallel by replicating matrix slices onto multiple cores, thus preserving the throughput advantage of IMC systems. This approach, however, incurs a significant area overhead owing to the replication of both memory arrays and circuit peripherals, such as the high-precision ADCs for output conversion, as well as requiring frequent tile-to-tile data movement for the accumulation of partial results. Area occupation and tile-to-tile communication may be reduced by spatially mapping tiles onto the same memory array, at the cost of either operating on smaller target matrices if the same memory array size is considered, or imposing stricter requirements on array nonidealities such as IR drop and driving capabilities of the DAC peripherals to retain operability on the same matrix size. Intermediate approaches, where multiple physical arrays are used, each storing multiple weight slices, may allow for the optimal balancing of inter-tile communication and tile accuracy constraints.

Area occupation can be reduced by adopting RNS-based decomposition, where the required precision of DACs, memory, and ADCs is determined by the integer moduli. Partial operations can be executed in parallel on multiple cores, each performing computations with respect to a modulo of the selected set. However, the digital circuitry must address the modular arithmetic to perform reconstruction by CRT with medium-high complexity. With respect to slicing, RNS displays a higher sensitivity to noise and perturbations, leading to correlation loss between the computed output and the expected output. RNS robustness can be improved by introducing redundant moduli, allowing detection and correction of shot-like noise events, or by selecting non-pairwise-coprime integers as moduli at the cost of increased reconstruction overhead. Noise rejection may be further enhanced by ECCs for the detection and correction of outlying noise events, the identification of highly variable devices, and the restoration of nominal precision.

By suitably combining slicing, RNS, and ECC, high-precision in-memory MVM units can be obtained from low-precision memory and peripherals. High-precision MVM represents a fundamental cornerstone for the iterative solution of linear systems, typically relying on the construction of fixed-point sequences converging to the desired IMVM result. While performing all the MVMs required by the iterative solver in a high-precision unit may result in significant energy consumption, IRF can be used to delegate a significant fraction of the required computations to a lower-precision unit with reduced latency and energy consumption. In this scheme, closed-loop AIMC may play a pivotal role by acting as a one-step, low-precision solver, with the enhanced-precision open-loop AIMC primitive providing high-precision refinement.

Table 3 summarizes the reviewed error mitigation techniques and their impact on computation precision, as well as some qualitative benchmarks in terms of latency, area consumption, and required digital overhead. Slicing, RNS, and IRF allow high-precision computation outputs by suitably combining low-precision partial results. Conversely, ECCs are typically limited to restoration of the nominal primitive precision by improving its robustness to noise and perturbations. Slicing, RNS, and ECC can be implemented with minimal latency by adopting spatial parallelism for slicing and RNS, and MVM-based encoders and decoders for ECC. Conversely, the sequential nature of IRF may result in increased latency for the high-

Table 3 | Qualitative benchmarks for error mitigation techniques

Technique	Impact on precision	Latency	Area	Digital overhead	Data movement
Slicing	Increase (MVM)	Low	High	Low	High
RNS	Increase (MVM)	Low	Low	High	Medium
ECC	Recovery (MVM)	Low	Low	Medium	Low
IRF	Increase (IMVM)	High	Medium	Medium	High

precision computation with respect to the low-precision IMVM solution. The spatial parallelism of RNS and slicing leads to significant area consumption, with RNS typically being advantageous with respect to slicing owing to the reduced bit-precision of ADCs. The area overhead of IRF can be reduced by reusing high-precision in-memory MVM accelerators for residual computation, avoiding the need for digital coprocessors. Implementation of modular arithmetic for reconstruction by CRT leads to a significant digital overhead required for RNS, whereas slicing can exploit S&A with reduced digital complexity. Syndrome computation in ECC may similarly require the implementation of modular arithmetic, although with a vastly reduced number of operands and, thus, a more lightweight circuitry requirement.

While this work focuses primarily on resistive AIMC, similar precision challenges arise in all analog computing contexts based on other physical domains. For instance, Zhou et al. demonstrated a photonic-electronic in-memory dot-product engine with 4-bit weight encoding, mainly limited by photodetector, shot, and thermal noise[130]. On the ferroelectric side, Soliman et al. showed a multi-level FeFET crossbar IMC implementation capable of multi-bit MAC operations, where bit-precision of stored weights is limited by device variability[131]. More recently, dual-design FeFET AIMC architectures with inherent shift-add mechanisms have been proposed to enhance precision and reduce overhead[132]. In quantum computing, decoherence and imperfect gate operations necessitate extensive use of quantum ECCs[133] and noise mitigation techniques[134,135]. While these works do not yet employ full precision-enhancement techniques, they underscore the cross-platform relevance of error mitigation in in-memory computing and suggest fertile ground for applying our surveyed approaches more broadly.

Conclusions

This work provides an overview of error mitigation techniques aimed at elevating the computing precision of inherently low-precision AIMC primitives, with a specific focus on MVM and IMVM operations. The main sources of error affecting analog computing primitives are first identified, including memory, DAC, and ADC quantization, variability, and noise, as well as circuit-level nonidealities. Slicing and RNS are introduced to relax the precision requirements for the memory and converters by decomposing high-bit-width inputs and matrices into low-bit-width operands. To reject noise and allow identification of faulty devices, three different ECC implementations are reviewed, namely Hamming and ABN codes, as well as fully-analog, AIMC-tailored ECCs. Finally, IRF is proposed as an IMVM-specific algorithm to allow high-precision solution of matrix equations by leveraging low-precision in-memory linear system solution, either by iterative open-loop AIMC or one-shot closed-loop AIMC. While the techniques reviewed in this work have been individually demonstrated in real AIMC prototypes, we argue that combining different error mitigation and precision improvement strategies within the same AIMC core can yield greater precision gains while offsetting the limitations of each individual approach. A promising direction for future work will be the development of systematic numerical studies, combining different modulation schemes, noise sources, and input distributions, to quantitatively assess the range of achievable precision improvements and trade-offs across techniques. A careful co-integration of these frameworks can enable harnessing the intrinsic energy and area efficiency of AIMC while achieving the high precision typically required by advanced data processing, thereby advancing AIMC systems toward real-world applicability.

Data availability

No datasets were generated or analysed during the current study.

Received: 24 July 2025; Accepted: 10 November 2025;
Published online: 07 January 2026

References

1. von Neumann, J. First draft of a report on the EDVAC. *IEEE Ann. Hist. Comput.* **15**, 27–75 (1993).
2. Aguirre, F. et al. Hardware implementation of memristor-based artificial neural networks. *Nat. Commun.* **15**, 1974 (2024).
3. Geens, R., Shi, M., Symons, A., Fang, C. & Verhelst, M. Energy cost modelling for optimizing large language model inference on hardware accelerators. in *Proc. IEEE 37th International System-on-Chip Conference (SOCC)* 1–6, https://doi.org/10.1109/SOCC62300.2024.10737844 (IEEE, 2024).
4. Jones, N. How to stop data centres from gobbling up the world's electricity. *Nature* **561**, 163–166 (2018).
5. Horowitz, M. 1.1 Computing's energy problem (and what we can do about it). in *Proc. IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)* 10–14, https://doi.org/10.1109/ISSCC.2014.6757323 (IEEE, 2014).
6. Zidan, M. A., Strachan, J. P. & Lu, W. D. The future of electronics based on memristive systems. *Nat. Electron.* **1**, 22–29 (2018).
7. Ielmini, D. & Wong, H.-S. P. In-memory computing with resistive switching devices. *Nat. Electron.* **1**, 333–343 (2018).
8. Sebastian, A., Le Gallo, M., Khaddam-Aljameh, R. & Eleftheriou, E. Memory devices and applications for in-memory computing. *Nat. Nanotechnol.* https://doi.org/10.1038/s41565-020-0655-z (2020).
9. Mittal, S., Verma, G., Kaushik, B. & Khanday, F. A. A survey of SRAM-based in-memory computing techniques and applications. *J. Syst. Architect.* **119**, 102276 (2021).
10. Lepri, N. et al. In-memory computing for machine learning and deep learning. *IEEE J. Electron Devices Soc.* **11**, 587–601 (2023).
11. Mannocci, P. et al. In-memory computing with emerging memory devices: Status and outlook. *APL Mach. Learn.* **1**, 010902 (2023).
12. Liu, Q. et al. 33.2 a fully integrated analog ReRAM based 78.4TOPS/W compute-in-memory chip with fully parallel MAC computing. in *Proc. IEEE International Solid- State Circuits Conference - (ISSCC)* 500–502, https://doi.org/10.1109/ISSCC19947.2020.9062953 (IEEE, 2020).
13. Khaddam-Aljameh, R. 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* **57**, 1027–1038 (2022).
14. Zhang, X., Jo, Y.-J. & Kim, T. T.-H. A 65-nm 55.8-TOPS/W Compact 2T eDRAM-Based Compute-in-Memory Macro With Linear Calibration. in *Proc. IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 1–5 500–502, https://doi.org/10.1109/ISSCC19947.2020.9062953 (IEEE, 2025).
15. Wang, H. et al. A charge domain SRAM compute-in-memory macro with C-2C ladder-based 8-bit MAC unit in 22-nm FinFET process for edge inference. *IEEE J. Solid-State Circuits* **58**, 1037–1050 (2023).

16. Wang, J. et al. A 28-nm compute SRAM with bit-serial logic/arithmetic operations for programmable in-memory vector computing. *IEEE J. Solid-State Circuits* **55**, 76–86 (2020).
17. Krishnan, G. et al. SIAM: chiplet-based scalable in-memory acceleration with mesh for deep neural networks. *ACM Trans. Embed. Comput. Syst.* **20**, 1–24 (2021).
18. Le Gallo, M. et al. A 64-core mixed-signal in-memory compute chip based on phase-change memory for deep neural network inference. *Nat. Electron.* **6**, 680–693 (2023).
19. Wan, W. et al. A compute-in-memory chip based on resistive random-access memory. *Nature* **608**, 504–512 (2022).
20. Li, C. et al. Analogue signal and image processing with large memristor crossbars. *Nat. Electron.* **1**, 52–59 (2018).
21. Song, W. et al. Programming memristor arrays with arbitrarily high precision for analog computing. *Science* **383**, 903–910 (2024).
22. Ambrogio, S. et al. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature* **558**, 60–67 (2018).
23. Cai, F. et al. Power-efficient combinatorial optimization using intrinsic noise in memristor Hopfield neural networks. *Nat. Electron.* **3**, 409–418 (2020).
24. Bhattacharya, T. et al. Computing high-degree polynomial gradients in memory. *Nat. Commun.* **15**, 8211 (2024).
25. Sun, Z. et al. Solving matrix equations in one step with cross-point resistive arrays. *Proc. Natl. Acad. Sci.* **116**, 4123–4128 (2019).
26. Sun, Z., Pedretti, G., Bricalli, A. & Ielmini, D. One-step regression and classification with cross-point resistive memory arrays. *Sci. Adv.* **6**, eaay2378 (2020).
27. Mannocci, P. & Ielmini, D. A generalized block-matrix circuit for closed-loop analog in-memory computing. *IEEE J. Explor. Solid-State Comput. Devices Circuits* **9**, 47–55 (2023).
28. Mannocci, P., Melacarne, E. & Ielmini, D. An analogue in-memory ridge regression circuit with application to massive MIMO acceleration. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **12**, 952–962 (2022).
29. Mannocci, P. et al. Accelerating massive MIMO in 6G communications by analog in-memory computing circuits. in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)* 1–5, https://doi.org/10.1109/ISCAS46773.2023.10181941 (IEEE, 2023).
30. Mannocci, P. et al. An SRAM-based reconfigurable analog in-memory computing circuit for solving linear algebra problems. in *Proc. International Electron Devices Meeting (IEDM)* 1–4, https://doi.org/10.1109/IEDM45741.2023.10413724 (IEEE, 2023).
31. Zeng, Q. et al. Realizing in-memory baseband processing for ultrafast and energy-efficient 6G. *IEEE Internet Things J.* **11**, 5169–5183 (2024).
32. Sun, Z., Ambrosi, E., Pedretti, G., Bricalli, A. & Ielmini, D. In-memory pagerank accelerator with a cross-point array of resistive memories. *IEEE Trans. Electron Devices* **67**, 1466–1470 (2020).
33. Korkmaz, A. et al. Spectral ranking in complex networks using memristor crossbars. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **13**, 357–370 (2023).
34. Chen, X. & Han, Y. Solving Least-Squares Fitting in $O(1)$ Using RRAM-based Computing-in-Memory Technique. in *Proc. 27th Asia and South Pacific Design Automation Conference (ASP-DAC)* 339–344, https://doi.org/10.1109/ASP-DAC52403.2022.9712568 (IEEE, 2022).
35. Zoppo, G. et al. Gaussian process for nonlinear regression via memristive crossbars. in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)* 1–5, https://doi.org/10.1109/ISCAS46773.2023.10181785 (IEEE, 2023).
36. Mannocci, P., Giannone, E. & Ielmini, D. In-memory principal component analysis by analogue closed-loop eigendecomposition. *IEEE Trans. Circuits Syst. II Express Briefs* **71**, 1839–1843 (2024).
37. Wang, S. et al. In-memory analog solution of compressed sensing recovery in one step. *Sci. Adv.* **9**, eadj2908 (2023).
38. Chen, Z. et al. An ultrafast (<200ns) sparse solution solver made by $HfWO_x/VO_y$ threshold tunable neurons. in *Proc. International Electron Devices Meeting (IEDM)*https://doi.org/10.1109/IEDM45741.2023.10413700 (IEEE, 2023).
39. Bao, H. et al. Energy efficient memristive transiently chaotic neural network for combinatorial optimization. *IEEE Trans. Circuits Syst. I Regul. Pap.* **71**, 3708–3716 (2024).
40. Shang, L., Adil, M., Madani, R. & Pan, C. Memristor-based analog recursive computation circuit for linear programming optimization. *IEEE J. Explor. Solid-State Comput. Devices Circuits* **6**, 53–61 (2020).
41. Mannocci, P. et al. A universal, analog, in-memory computing primitive for linear algebra using memristors. *IEEE Trans. Circuits Syst. I: Regul. Pap.* **68**, 4889–4899 (2021).
42. Lin, J., Barrows, F. & Caravelli, F. Memristive Linear Algebra. *Phys. Rev. Res.* **7**, 023241 (2025).
43. Aifer, M. et al. Thermodynamic linear algebra. *npj Unconv. Comput.* **1**, 13 (2024).
44. Bertuletti, M., Muñoz-Martín, I., Bianchi, S., Bonfanti, A. G. & Ielmini, D. A multilayer neural accelerator with binary activations based on phase-change memory. *IEEE Trans. Electron Devices* **70**, 986–992 (2023).
45. Carletti, F. et al. Low-energy, high-accuracy convolutional network inference in 3D crosspoint (3DXP) arrays. in *Proc. IEEE European Solid-State Electronics Research Conference (ESSERC)* 412–415. https://doi.org/10.1109/ESSERC62670.2024.10719497 (IEEE, 2024).
46. Ding, Z., Li, X. & Lin, L. Simulating open quantum systems using hamiltonian simulations. *PRX Quantum* **5**, 020332 (2024).
47. Sato, Y., Kondo, R., Hamamura, I., Onodera, T. & Yamamoto, N. Hamiltonian simulation for hyperbolic partial differential equations by scalable quantum circuits. *Phys. Rev. Res.* **6**, 033246 (2024).
48. Guo, Z. et al. Diffusion models in bioinformatics and computational biology. *Nat. Rev. Bioeng.* **2**, 136–154 (2023).
49. Le Gallo, M. et al. Mixed-precision in-memory computing. *Nat. Electron.* **1**, 246–253 (2018).
50. Selvadurai, A. P. S. *Partial Differential Equations in Mechanics 1: Fundamentals, Laplace's Equation, Diffusion Equation, Wave Equation* (Springer, 2000).
51. Lebrun, G., Gao, J. & Faulkner, M. MIMO transmission over a time-varying channel using SVD. *IEEE Trans. Wirel. Commun.* **4**, 757–764 (2005).
52. Veitschegger, W. & Wu, C.-H. Robot accuracy analysis based on kinematics. *IEEE J. Robot. Autom.* **2**, 171–179 (1986).
53. Verma, N. et al. In-memory computing: advances and prospects. *IEEE Solid-State Circuits Mag.* **11**, 43–55 (2019).
54. Li, Y., Wang, S. & Sun, Z. The maximum storage capacity of open-loop written RRAM is around 4 bits. in *Proc. IEEE 17th International Conference on Solid-State & Integrated Circuit Technology (ICSICT)* 1–3, https://doi.org/10.1109/ICSICT62049.2024.10831637 (2024).
55. Zahoor, F., Azni Zulkifli, T. Z. & Khanday, F. A. Resistive random access memory (RRAM): an overview of materials, switching mechanism, performance, multilevel cell (MLC) storage, modeling, and applications. *Nanoscale Res. Lett.* **15**, 90 (2020).
56. Milo, V. et al. Accurate program/verify schemes of resistive switching memory (RRAM) for in-memory neural network circuits. *IEEE Trans. Electron Devices* **68**, 3832–3837 (2021).
57. Mackin, C. et al. Optimised weight programming for analogue memory-based deep neural networks. *Nat. Commun.* **13**, 3765 (2022).
58. Pistolesi, L. et al. Drift compensation in multilevel PCM for in-memory computing accelerators. in *Proc. IEEE International Reliability Physics Symposium (IRPS)* 1–4, https://doi.org/10.1109/IRPS48228.2024.10529438 (IEEE, 2024).

59. Ambrogio, S., Balatti, S., McCaffrey, V., Wang, D. C. & Ielmini, D. Noise-induced resistance broadening in resistive switching memory—part I: intrinsic cell behavior. *IEEE Trans. Electron Devices* **62**, 3805–3811 (2015).
60. Chen, A. A comprehensive crossbar array model with solutions for line resistance and nonlinear device characteristics. *IEEE Trans. Electron Devices* **60**, 1318–1326 (2013).
61. Liu, B. et al. Reduction and IR-drop compensations techniques for reliable neuromorphic computing systems. in *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* 63–70, https://doi.org/10.1109/ICCAD.2014.7001330 (2014).
62. Fouda, M. E. et al. IR-QNN framework: an IR drop-aware offline training of quantized crossbar arrays. *IEEE Access* **8**, 228392–228408 (2020).
63. Lepri, N., Glukhov, A. & Ielmini, D. Mitigating read-program variation and IR drop by circuit architecture in RRAM-based neural network accelerators. in *Proc. IEEE International Reliability Physics Symposium (IRPS)* 3C.2–1–3C.2–6, https://doi.org/10.1109/IRPS48227.2022.9764486 (IEEE, 2022).
64. Lepri, N. et al. Modeling and compensation of IR drop in crosspoint accelerators of neural networks. *IEEE Trans. Electron Devices* **69**, 1575–1581 (2022).
65. Fouda, M. E., Eltawil, A. M. & Kurdahi, F. Modeling and analysis of passive switching crossbar arrays. *IEEE Trans. Circuits Syst. I: Regul. Pap.* **65**, 270–282 (2018).
66. Li, C., Roth, R. M., Graves, C., Sheng, X. & Strachan, J. P. Analog error correcting codes for defect tolerant matrix multiplication in crossbars. in *Proc. IEEE International Electron Devices Meeting (IEDM)* 36.6.1–36.6.4, https://doi.org/10.1109/IEDM13553.2020.9371978 (IEEE, 2020).
67. Yu, S., Jiang, H., Huang, S., Peng, X. & Lu, A. Compute-in-memory chips for deep learning: recent trends and prospects. *IEEE Circuits Syst. Mag.* **21**, 31–56 (2021).
68. Horn, R. A. & Johnson, C. R. *Matrix Analysis* 2nd edn (Cambridge University Press, 2012).
69. Murmann, B. ADC performance survey 1997–2024. https://github.com/bmurmann/ADC-survey (2024).
70. Milo, V. et al. Multilevel HfO_2-based RRAM devices for low-power neuromorphic networks. *APL Mater.* **7**, 081120 (2019).
71. Ambrogio, S. et al. An analog-AI chip for energy-efficient speech recognition and transcription. *Nature* **620**, 768–775 (2023).
72. Pistolesi, L. et al. Differential phase change memory (PCM) cell for drift-compensated in-memory computing. *IEEE Trans. Electron Devices* **71**, 7447–7453 (2024).
73. Rasch, M. J. et al. Hardware-aware training for large-scale and diverse deep learning inference workloads using in-memory computing-based accelerators. *Nat. Commun.* **14**, 5282 (2023).
74. Jiang, H., Li, W., Huang, S. & Yu, S. A 40 nm analog-input ADC-free compute-in-memory RRAM macro with pulse-width modulation between sub-arrays. in *Proc. IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)* 266–267, https://doi.org/10.1109/VLSITechnologyandCir46769.2022.9830211 (IEEE, 2022).
75. Le Gallo, M. et al. Precision of bit slicing with in-memory computing based on analog phase-change memory crossbars. *Neuromorphic Comput. Eng.* **2**, 014009 (2022).
76. Shafiee, A. et al. ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars. in *Proc. 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)* 14–26, https://doi.org/10.1109/ISCA.2016.12 (IEEE, 2016).
77. Hu, M. et al. Dot-product engine for neuromorphic computing: programming 1T1M crossbar to accelerate matrix-vector multiplication. in *Proc. 53rd Annual Design Automation Conference* 1–6, https://doi.org/10.1145/2897937.2898010 (ACM, 2016).
78. Xiao, T. P. et al. Analog fast Fourier transforms for scalable and efficient signal processing. ArXiv: http://arxiv.org/abs/2409.19071 (2024).
79. Pedretti, G. et al. Redundancy and analog slicing for precise in-memory machine learning—part II: applications and benchmark. *IEEE Trans. Electron Devices* **68**, 4379–4383 (2021).
80. Ankit, A. et al. PUMA: a programmable ultra-efficient memristor-based accelerator for machine learning inference. in *Proc. 24th International Conference on Architectural Support for Programming Languages and Operating Systems* 715–731, https://doi.org/10.1145/3297858.3304049 (ACM, 2019).
81. Rao, M. et al. Thousands of conductance levels in memristors integrated on CMOS. *Nature* **615**, 823–829 (2023).
82. Soderstrand, M. A. (ed.) *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing,* IEEE *Press Selected Reprint Series* (Institute of Electrical and Electronics Engineers, 1986).
83. Ding, C., Pei, T.-i. & Salomaa, A.*Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography* (World Scientific, 1996).
84. Xiao, L., Xia, X.-G. & Huo, H. Towards robustness in residue number systems. *IEEE Trans. Signal Process.* **65**, 1497–1510 (2017).
85. Demirkiran, C., Nair, L., Bunandar, D. & Joshi, A. A blueprint for precise and fault-tolerant analog neural networks. *Nat. Commun.* **15**, 5098 (2024).
86. Chang, C.-H., Molahosseini, A. S., Zarandi, A. A. E. & Tay, T. F. Residue number systems: a new paradigm to datapath optimization for low-power and high-performance digital signal processing applications. *IEEE Circuits Syst. Mag.* **15**, 26–44 (2015).
87. Chokshi, R., Berezowski, K. S., Shrivastava, A. & Piestrak, S. J. Exploiting residue number system for power-efficient digital signal processing in embedded processors. in *Proc. 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems* 19–28, https://doi.org/10.1145/1629395.1629401 (ACM, 2009).
88. Cardarilli, G. C., Nannarelli, A. & Re, M. Residue number system for low-power DSP applications. in *Proc. Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers* 1412–1416, https://doi.org/10.1109/ACSSC.2007.4487461 (IEEE, 2007).
89. Chaves, R. & Sousa, L. RDSP: a RISC DSP based on residue number system. in *Proc. Euromicro Symposium on Digital System Design*. 128–135, https://doi.org/10.1109/DSD.2003.1231911 (IEEE, 2003).
90. Samimi, N., Kamal, M., Afzali-Kusha, A. & Pedram, M. Res-DNN: a residue number system-based DNN accelerator unit. *IEEE Trans. Circuits Syst. I Regul. Pap.* **67**, 658–671 (2020).
91. Olsen, E. B. RNS hardware matrix multiplier for high precision neural network acceleration: "RNS TPU". in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)* 1–5, https://doi.org/10.1109/ISCAS.2018.8351352 (IEEE, 2018), (2018).
92. Nakahara, H. & Sasao, T. A deep convolutional neural network based on nested residue number system. in *Proc. 25th International Conference on Field Programmable Logic and Applications (FPL)* 1–6, https://doi.org/10.1109/FPL.2015.7293933 (IEEE, 2015).
93. Shen, S., Yang, H., Liu, Y., Liu, Z. & Zhao, Y. CARM: CUDA-accelerated RNS multiplication in word-wise homomorphic encryption schemes for internet of things. *IEEE Trans. Comput.* **72**, 1999–2010 (2023).
94. Salamat, S., Imani, M., Gupta, S. & Rosing, T. RNSnet: in-memory neural network acceleration using residue number system. in *Proc. IEEE International Conference on Rebooting Computing (ICRC)* 1–12, https://doi.org/10.1109/ICRC.2018.8638592 (IEEE, 2018).
95. Roohi, A., Taheri, M., Angizi, S. & Fan, D. RNSiM: efficient deep neural network accelerator using residue number systems. in *Proc. IEEE/ACM International Conference On Computer Aided Design*

(ICCAD) 1–9, https://doi.org/10.1109/ICCAD51958.2021.9643531 (IEEE, 2021).
96. Angizi, S., Roohi, A., Taheri, M. & Fan, D. Processing-in-memory acceleration of MAC-based applications using residue number system: a comparative study. in *Proc. of the 2021 Great Lakes Symposium on VLSI* 265–270, https://doi.org/10.1145/3453688.3461529 (ACM, 2021).
97. Demirkiran, C., Yang, G., Bunandar, D. & Joshi, A. Mirage: An RNS-Based Photonic Accelerator for DNN Training. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)* 73–87, https://doi.org/10.1109/ISCA59077.2024.00016 (IEEE, 2024).
98. Watson, R. & Hastings, C. Self-checked computation using residue arithmetic. *Proc. IEEE* **54**, 1920–1931 (1966).
99. James, J. & Pe, A. Error correction based on redundant Residue Number System. in *Proc. IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)* 1–5, https://doi.org/10.1109/CONECCT.2015.7383940 (IEEE, 2015).
100. Lie-Liang, Y. & Hanzo, L. Redundant residue number system based error correction codes. in *Proc. IEEE 54th Vehicular Technology Conference. VTC Fall 2001. Proceedings (Cat. No.01CH37211)*, Vol. 3 1472–1476, https://doi.org/10.1109/VTC.2001.956442 (IEEE, 2001).
101. Hamming, R. W. Error Detecting and error correcting codes. *Bell Syst. Tech. J.* **29**, 147–160 (1950).
102. Huffman, W. C. & Pless, V. (eds) *Fundamentals of Error-correcting Codes* (Cambridge University Press, 2003).
103. Johannesson, R. & Zigangirov, K. S. *Fundamentals of Convolutional Coding* IEEE Series on Digital & Mobile Communication (IEEE Press, 1999).
104. Reed, I. S. & Solomon, G. Polynomial codes over certain finite fields. *J. Soc. Ind. Appl. Math.* **8**, 300–304 (1960).
105. Bose, R. & Ray-Chaudhuri, D. On a class of error correcting binary group codes. *Inf. Control* **3**, 68–79 (1960).
106. Reed, I. S. Error-control coding for data networks No. v.508. in *The Springer International Series in Engineering and Computer Science Ser* (Springer, 1999).
107. Niu, D., Xiao, Y. & Yuan, X. Low power memristor-based ReRAM design with Error Correcting Code. in *Proc. 17th Asia and South Pacific Design Automation Conference* 79–84, https://doi.org/10.1109/ASPDAC.2012.6165062 (IEEE, 2012).
108. Bae, W., Han, J.-W. & Yoon, K. J. In-memory hamming error-correcting code in memristor crossbar. *IEEE Trans. Electron Devices* **69**, 3700–3707 (2022).
109. Feinberg, B., Wang, S. & Ipek, E. Making memristive neural network accelerators reliable. in *Proc. IEEE International Symposium on High Performance Computer Architecture (HPCA)* 52–65, https://doi.org/10.1109/HPCA.2018.00015 (IEEE, 2018).
110. Li, W., Read, J., Jiang, H. & Yu, S. MAC-ECC: in-situ error correction and its design methodology for reliable NVM-based compute-in-memory inference engine. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **12**, 835–845 (2022).
111. Roth, R. M. Fault-tolerant dot-product engines. *IEEE Trans. Inf. Theory* **65**, 2046–2057 (2019).
112. Roth, R. M. Analog error-correcting codes. *IEEE Trans. Inf. Theory* **66**, 4075–4088 (2020).
113. Jiang, A. Analog error-correcting codes: designs and analysis. *IEEE Trans. Inf. Theory* **70**, 7740–7756 (2024).
114. Wei, H. & Roth, R. M. Multiple-error-correcting codes for analog computing on resistive crossbars. *IEEE Trans. Inf. Theory* **70**, 8647–8658 (2024).
115. Sau, C. et al. Feasibility study and porting of the damped least square algorithm on FPGA. *IEEE Access* **8**, 175483–175500 (2020).
116. Golub, G. H. & Van Loan, C. F.*Matrix Computations* (The Johns Hopkins University Press, 2013).
117. Wilkinson, J. H. *Rounding Errors in Algebraic Processes* (Dover, 1994).
118. Moler, C. B. Iterative refinement in floating point. *J. ACM* **14**, 316–321 (1967).
119. Bjorck, A. Iterative refinement of linear least squares solutions I. *BIT* **7**, 257–278 (1967).
120. Bjorck, A. Iterative refinement of linear least squares solutions II. *BIT* **8**, 8–30 (1968).
121. Ogita, T. & Aishima, K. Iterative refinement for symmetric eigenvalue decomposition. *Jpn. J. Ind. Appl. Math.* **35**, 1007–1035 (2018).
122. Carson, E. & Higham, N. J. A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems. *SIAM J. Sci. Comput.* **39**, A2834–A2856 (2017).
123. Richter, I. et al. Memristive accelerator for extreme scale linear solvers. in *Proc. Government Microcircuit Applications & Critical Technology Conference (GOMACTech)* 1–4 (St. Louis, 2015).
124. Li, J. et al. Fully analog iteration for solving matrix equations with in-memory computing. *Sci. Adv.* **11**, eadr6391 (2025).
125. Feinberg, B. et al. An analog preconditioner for solving linear systems. in *Proc. IEEE International Symposium on High-Performance Computer Architecture (HPCA)* 761–774. https://doi.org/10.1109/HPCA51647.2021.00069 (IEEE, 2021).
126. Higham, N. J. Numerical analysis of a quadratic matrix equation. *IMA J. Numer.* **20**, 499–519 (2000).
127. Kalantzis, V. et al. Solving sparse linear systems with approximate inverse preconditioners on analog devices. in *Proc. IEEE High Performance Extreme Computing Conference (HPEC)* 1–7. https://doi.org/10.1109/HPEC49654.2021.9622816 (IEEE, 2021).
128. Kalantzis, V. et al. Solving sparse linear systems via flexible GMRES with in-memory analog preconditioning. in *Proc. IEEE High Performance Extreme Computing Conference (HPEC)* 1–7. https://doi.org/10.1109/HPEC58863.2023.10363502 (IEEE, 2023).
129. Wu, C. W., Squillante, M. S., Kalantzis, V. & Horesh, L. Stable iterative refinement for solving linear systems with inaccurate computation. *J. Comput. Appl. Math.* **471**, 116746 (2026).
130. Zhou, W. et al. In-memory photonic dot-product engine with electrically programmable weight banks. *Nat. Commun.* **14**, 2887 (2023).
131. Soliman, T. et al. First demonstration of in-memory computing crossbar using multi-level Cell FeFET. *Nat. Commun.* **14**, 6348 (2023).
132. Yang, Z. et al. Energy Efficient Dual Designs of FeFET-Based Analog In-Memory Computing with Inherent Shift-Add Capability. in *Proc. 61st ACM/IEEE Design Automation Conference* 1–6, https://doi.org/10.1145/3649329.3655990 (ACM, 2024).
133. Google Quantum AI and Collaborators et al. Quantum error correction below the surface code threshold. *Nature* **638**, 920–926 (2025).
134. García-Molina, P., Martin, A., Garcia De Andoin, M. & Sanz, M. Mitigating noise in digital and digital-analog quantum computation. *Commun. Phys.* **7**, 321 (2024).
135. Quiroz, G. et al. Quantifying the impact of precision errors on quantum approximate optimization algorithms. *Phys. Rev. Res.* **7**, 023240 (2025).
136. Parrini, L. et al. Error detection and correction codes for safe in-memory computations. in *Proc. IEEE European Test Symposium (ETS)* 1–4, https://doi.org/10.1109/ETS61313.2024.10567894 (IEEE, 2024).

Acknowledgements

This article has received funding from the European Research Council (ERC) under the European Union's Horizon Europe Research and Innovation Programme (grant 101054098).

Author contributions

P.M. and D.I. conceived the idea and defined the overall organization of the work. P.M., G.L., and M.B. carried out the critical literature review. Simulations of bit-slicing and iterative refinement were performed by P.M. and G.L., while P.M. and M.B. conducted simulations of residue number systems. P.M. and D.I. wrote the manuscript with input and feedback from all authors. D.I. supervised the project and provided critical guidance throughout. All authors have read and approved the final version of the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to Piergiulio Mannocci or Daniele Ielmini.

Reprints and permissions information is available at http://www.nature.com/reprints

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.