Turning to Information Theory to Bring In-Memory Computing Into Practice

Elsa Dupraz, François Leduc-Primeau, Kui Cai, Lara Dolecek

Turning to Information Theory to Bring In-Memory Computing Into Practice

Elsa Dupraz, François Leduc-Primeau, Kui Cai, and Lara Dolecek

***Abstract*—This paper explores the emerging field of in-memory computing, which has the potential to significantly improve energy efficiency in many signal processing and machine learning applications. In-memory computing systems are impacted by various sources of noise and perturbations that can affect their computation accuracy. Therefore, this paper aims to identify the key challenges to be addressed from an information-theoretic point of view in this area. It first identifies relevant computation structures and noise models from the literature of hardware implementation, and then reviews existing works in information theory and error-correction for in-memory computing. Finally, it identifies key open avenues for establishing information-theoretic foundations of in-memory computing systems, and for providing insightful design rules leading to highly energy-efficient computing systems.**

I. Introduction

In the past, the performance and energy efficiency of electronic computing systems were mainly improved through advancements in CMOS integrated circuit technology, with the number of transistors per chip increasing from around a thousand in 1970 to more than 50 billion in 2022 [1]. However, as it is unlikely that CMOS technology will continue to make significant improvements in the future, computer designers are exploring other methods to achieve further enhancements, such as specialized architectures and new styles of computing circuits. In particular, the conventional von Neumann architecture involves significant data transfers between memories and processing units, which is costly in terms of latency and energy consumption. In-memory computing (IMC) architectures are seen as particularly promising alternatives since they simultaneously solve the von Neumann bottleneck while offering good storage densities. Among other applications, in-memory computing is seen as having great potential in the field of artificial intelligence (AI), as it could significantly reduce the energy consumption of AI circuits [2].

Emerging non-volatile memories (eNVMs), such as spin-transfer torque magnetic random access memory (STT-MRAM), phase change memory (PCM), resistive random access memory (ReRAM), and ferroelectric field-effect transistors (FeFETs) are currently being intensively explored as the key building block for in-memory computing. Their memory cells can be programmed into different resistance/conductance states, through which both storage and analog multiply-and-accumulate (MAC) operations can be done [3]. In-memory computing architectures have been developed for a variety of applications in the fields of signal processing, optimization [4], computer science [5], and AI [2], [6]. Although in-memory computing can also be achieved with conventional memory circuits such as SRAMs [7], in this paper we choose to focus on eNVMs since their non-volatile property enables an additional source of energy efficiency from the ability to quickly power on or off the device.

Memristive devices, also referred to as memristors, are typically used to build memory cells in eNVMs. Memristors are characterized by their distinctive physical property, namely, the alteration of their internal conductance in response to the current passing through them. This characteristic not only facilitates data storage but also, when harnessed in conjunction with a two-dimensional (2D) crossbar configuration of memristors, enables the construction of various computational structures, from dot-product engines [8] to logic circuits [9] or shortest path computation [5]. The prevalent methodology for building such computational structures typically entails an initial step wherein an idealized electrical circuit is designed, employing memristors in conjunction with other electronic components such as resistors and amplifiers, to execute the desired computational task. Subsequently, this abstracted circuit is mapped onto the aforementioned

Elsa Dupraz is with IMT Atlantique, Lab-STICC, UMR CNRS 6285, 29238 Brest, France (email: elsa.dupraz@imt-atlantique.fr).

François Leduc-Primeau is with the Department of Electrical Engineering, Polytechnique Montreal, QC, Canada (email: francois.leduc-primeau@polymtl.ca).

Kui Cai is with the Science, Mathematics and Technology Cluster, Singaproe Unversity of Technology and Design (SUTD), Singapore 487372 (email: cai_kui@sutd.edu.sg).

Lara Dolecek is with the Electrical and Computer Engineering Department, University of California Los Angeles (UCLA), Los Angeles, CA, 90095, USA (email: dolecek@ee.ucla.edu).

Kui Cai's work was supported by the Singapore Ministry of Education Academic Research Fund Tier 2 T2EP50221-0036

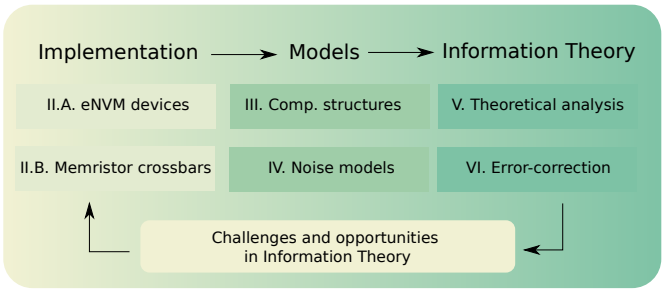

Figure 1. Outline of the paper. Extracting computation structures and error models from the physics and implementation of eNVM devices will hopefully lead to insightful information theory analysis of in-memory computing and to more efficient implementations.

crossbar architecture.

These IMC circuits are impacted by various sources of perturbations that can be modeled as a combination of noise and errors introduced into the values stored and during the computations performed on those values. Careful management of this noise can bring significant opportunities into the design by addressing the inherent tradeoff between the amount of noise and the computation energy. It is thus beneficial to study these systems from an information-theoretic perspective in order to establish a theoretical foundation for their efficient design. The joint investigation of architecture design and information theory gives rise to fresh challenges and questions for information theory, while also contributing valuable insights and analytical tools to the design process.

The primary objectives of this paper are to extract computation structures and noise models from the existing body of literature on IMC hardware implementation, and to identify information-theoretic problems of interest for in-memory computing. When discussing such problems, the following three aspects will be crucial: (i) the need for the development of accurate yet simple noise models to capture memory device non-idealities while making the analysis tractable, (ii) the importance of analyzing and designing in-memory computing systems from the perspective of the final application (such as matrix-vector multiplication, deep neural networks, etc.), given that each application has its own performance metrics and constraints, and (iii) the crucial role of hardware efficiency in terms of power and latency for developing energy-efficient in-memory computing solutions, as this is a key benefit of this technology.

The paper outline, summarized in Figure 1, is as follows. Section II describes the physical background of in-memory computing devices. Section III presents computation structures for key applications. Section IV extracts relevant noise models from the literature. Section V introduces existing information-theoretic analyses for in-memory computing. Section VI describes existing error-correction solutions.

II. Physical Background

In this part, after discussing key memory technologies that allow for in-memory computing, we present a high-level physical model of memristors that captures the relationship between the memristor conductance value and the current that flows accross it. We next describe the 2D memristor crossbar arrangement which will then allow for implementing various IMC structures.

A. Existing Memory Devices for In-Memory Computing

Currently, a variety of nanoscale memory devices are being considered for in-memory computing [3], [7]. Conventional charge-based memories such as static random access memory (SRAM), dynamic random access memory (DRAM), and Flash memory, store information based on the presence or absence of charge in the memory components. SRAM and DRAM, being conventional volatile memories, feature excellent read/write speed with mature fabrication process. Both SRAM and DRAM can implement Boolean functions and other discrete arithmetic operations in memory [7]. On the other hand, NAND Flash memory dominates the current non-volatile memory (NVM) market and supports sequential data accesses. Both NAND and NOR Flash memories can implement analog arithmetic operations in memory [10].

eNVMs refer to resistance-based memories that utilize specific resistive components with variable internal resistance states to store information. The internal resistance level of each elementary memory component can change based on the voltage drop applied across it. Among eNVMs, STT-MRAM, PCM, and ReRAM possess appealing features such as excellent scalability and near-zero leakage power [3]. STT-MRAM [11] can achieve the shortest access latency and the lowest energy consumption while having a relatively larger cell size. STT-MRAM components can only store binary values, due to their limited resistance range. Both PCM [12] and ReRAM [13] have demonstrated multilevel resistance states, which allows them to store multiple bits per cell for storage and computation purposes. Another promising eNVM device is FeFET [14], which exhibits low read latency, but a limited number of levels.

B. Memristor Physical Model

In eNVMs, resistive memory components are also referred to as memristive devices or memristors. The

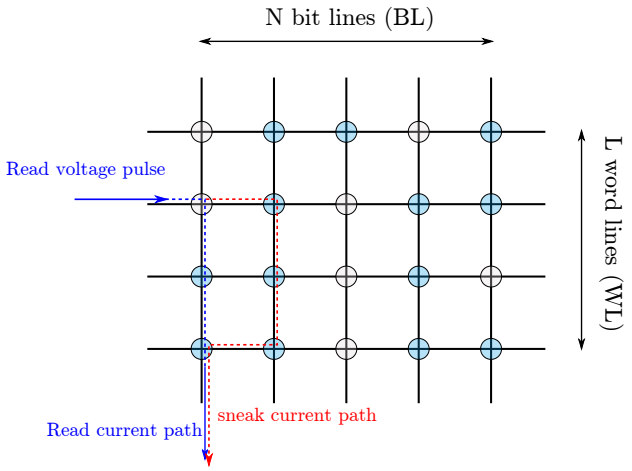

Figure 2. Example of a memristor crossbar array composed of L row WLs and N column BLs, with SLC. Each intersection between a WL and a BL contains a memory cell. The blue cells contain logical value 1 (low resistance value r_{ON}) and the gray cells contain logical value 0 (high resistance value r_{OFF}). To read one cell value, a read voltage pulse is applied at the input WL, and the current is read at the output BL. A sneak-path is a concurrent current flow which affects more the reading of cells in high resistance states, while cells in the concurrent path are in low resistance states.

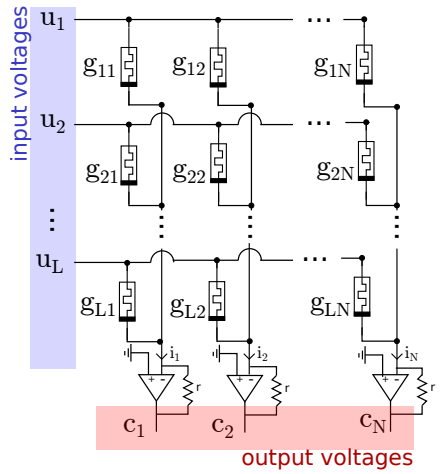

Figure 3. Electrical circuit of the DPE built from a $L \times N$ memristor crossbar, where memory cell (j,k) is built from a memristor of conductance $g_{j,k}$. By applying Ohm's law (1) and Kirchhoff's current law on each BL, we show that the current i_k has expression $i_k = \sum_{j=1}^{L} g_{j,k} u_j$. The output TIA with feedback resistance r transforms the current i_k into a voltage c_k as $c_k = r i_k$, which permits to read the DPE output.

first physical model of the memristors voltage-current characteristic was expressed in [15] as

$$v(t) = r(t) i(t) \tag{1}$$

$$r(t) = r_{\text{ON}} \alpha q(t) + r_{\text{OFF}} \left(1 - \beta q(t)\right), \tag{2}$$

where $v(t)$, $q(t)$, $i(t) = \frac{dq}{dt}$, and $r(t)$, are the voltage, charge, current, and resistance of the memristor at time instant t, respectively, and α, β are parameters that depend on internal characteristics of the memristors. The first equation is the conventional Ohm's law, while the second one is specific to memristors, where the internal resistance value varies with its charge. Although the model in (2) is a rough approximation of the physics of the memristor, it allows us to understand its behavior from a high level perspective. Some more recent physical models were also proposed, which capture more accurately the device non-linearity [16].

C. Memristor Crossbars for In-Memory Computing

In eNVMs, memristors are typically arranged in 2D crossbar arrays [7], see Figure 2 for an example. Crossbar arrays are formed by the intersections of row *word* lines (WL) and column *bit* lines (BL), and each intersection contains a memory cell. There exists different memory cell constructions. The 1M crossbar configuration (one memristor per cell) is subject to a sneak-path phenomenon, which corresponds to undesired paths for the current flow in the crossbar. The 1T1M configuration (one transistor and one memristor per cell) can alleviate this issue but requires a larger area [17]. Since the 1M configuration is also harder to program, it is less considered in practice. Therefore, in the remaining of the paper, we always consider 1T1M configurations. However, in the figures, for simplicity and clarity, we represent each memory cell as a memristor only, omitting the transistor.

Another important aspect resides in the number of resistance/conductance levels that can be programmed within memristors. Widely used single-level cells (SLC) only support two possible values: one low-resistance state (LRS) with resistance value r_{ON} and conductance value g_{ON} (logical value 1), and one high-resistance state (HRS) with resistance value r_{OFF} and conductance value g_{OFF} (logical value 0) [15]. Alternatively, multi-level cells (MLC) and analog cells (AC) are very appealing in many applications. A MLC takes its value in a discrete alphabet, while an AC represents one real value.

D. Memory Reads and Writes

As described in Figure 2, one can read a specific conductance value at position (j,k) by applying a read voltage pulse on WL j and reading the current on BL k, according to (1). In addition, to write a specific value to the memristor at position (j,k), one can apply a write voltage pulse of higher amplitude at WL j, relying on (2). For instance, in SLC, each memristor can be set to either r_{ON} or r_{OFF} by applying a voltage pulse V_{SET} for HRS or V_{RESET} for LRS. In the 1T1M configuration,

the transistors permit to select the cell to be updated in the WL.

Note that the terms WL and BL originate from the SLC random access memories, and by convention remain in use when discussing crossbar arrays. However, crossbar arrays allow activating all rows (WLs) simultaneously, while cells may store more than one bit, with the result that BLs indeed may carry much more than one bit of information. Unfortunately, the terminology is not uniform in the literature. For instance some authors use "bit lines" to refer to the input lines and "source lines" to refer to the output lines.

III. Computation Structures

The objective of this section is to describe in-memory computation structures that can be implemented within a memristor crossbar. These architectures have been developed by leveraging inherent physical characteristics of memristors captured by equations (1) and (2). In what follows, we focus on three fundamental operations: dot-product computation, Boolean logic computation, and Hamming distance computation, which will then allow to address a large variety of applications in the area of optimization, signal processing, AI, and computer sciences.

A. Dot-Product Computation

Memristor crossbars allow for the implementation of in-memory analog dot-product engines (DPE) represented by the equation $\mathbf{c} = \mathbf{u}A$, where the matrix A is of size $L \times N$, and the vectors $\mathbf{c}$ and $\mathbf{u}$ are of length N and L, respectively. The DPE is realized with AC using an idealized circuit shown in Figure 3. In this crossbar, each coefficient $a_{j,k}$ of the matrix A is mapped as a conductance value $g_{j,k}/r$ at the intersection of WL j and BL k [8], where r is the feedback resistance of the transimpedance amplifier (TIA) at the end of each BL. Each element of the input vector $\mathbf{u}$ is fed into a digital-to-analog converter (DAC) to produce a voltage level. The product output $A\mathbf{u}$ is then reflected in the voltages $\mathbf{c}$ after the TIAs at the end of each column. Finally, these voltages are fed into analog-to-digital converters (ADCs) to generate the digital outputs. Since conductance values must be positive, the dot-product operation is often performed by using two memristors for each matrix element, one for positive values and one for negative values [4]. In this case, the final output can be obtained by performing current subtraction in the analog domain, or performing the subtraction in the digital domain.

The DPE shown in Figure 3 is considered in many applications. First, it can be used to implement the linear parts of deep neural network (DNN) layers, with each conductance storing a neuron weight [2], [6], [18]. The non-linear parts of each DNN layer are then implemented either in CMOS [2], [18], or with a dedicated analog circuit [6]. In addition, the DPE can be used to compute the fixed-points of the matrix A by re-injecting output voltages c_k as inputs u_j through several iterations. This observation comes from the fact that the fixed-points of a given function $f : \mathbb{R}^N \to \mathbb{R}^N$ can be calculated iteratively as $\mathbf{x}^{(\ell+1)} = f(\mathbf{x}^{(k)})$. This iterative procedure converges locally (*e.g*, when initialized close to the fixed-point) to a fixed point $\mathbf{x}^\star$ satisfying $\mathbf{x}^\star = f(\mathbf{x}^\star)$, given that the Jacobian of the function f satisfies certain conditions [19].

Note that some other DPE architectures consider the circuit of Figure 3 but replace the TIA by a simple resistance with internal conductance level g_s [4]. In this case, the relation between the matrix component $a_{j,k}$ and the conductance levels $g_{j,k}$ is given by $a_{j,k} = \frac{g_{j,k}}{g_s + \sum_{j=1}^{L} g_{j,k}}$ [4]. When considering $N = L$, this architecture also enables to implement the inverse operation $\mathbf{u} = \mathbf{c}A^{-1}$, by first applying specific output voltages c_k and then reading the input voltages u_j [4]. This property permits us to solve a variety of optimization and signal processing problems, by recasting them, either exactly or approximately, as combinations of dot-product computation, inverse dot-product computation, and fixed-point computation [4].

B. Logic-in-Memory

In-memory computing can also implement logic circuits by considering SLC with two conductance levels g_{ON} and g_{OFF}. As one of the first proposed solutions, the memristor-based material implication (IMPLY) logic [9] implements in-memory NOR operations over K inputs; see Figure 4 (a). The NOR-gate output $z = \overline{x_1 + x_2 + \cdots x_N}$ is equal to 1 only if all binary values x_k are set to 0. To evaluate z, conductance values g_1 to g_K are set to logic input gate values x_k either 0 or 1 (through g_{ON} and g_{OFF}) and the NOR gate output is then read in the internal conductance state h of the rightmost memristor. However, the IMPLY architecture requires each memristor to be connected to one resistor R_S, making it challenging to integrate many logic gates into the same crossbar. Alternatively, the memristor-aided logic (MAGIC) architecture [20] shown in Figure 4 (b) can also implement a K-input NOR gate but without using resistors R_S. It is worth noting that the NOR gate operation has the functional completeness property, which means that any logic function can be implemented from NOR gates only. However,

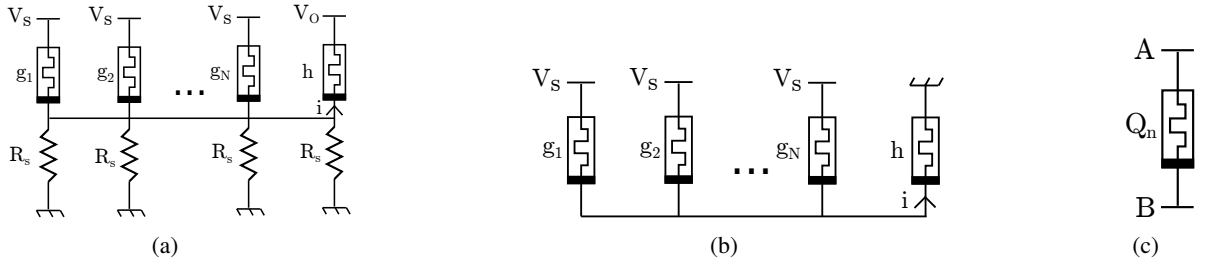

Figure 4. Logic gates implemented in memory. (a) NOR gate from IMPLY architecture [9]. Conductance levels g_1 to g_K store the K gate inputs, while conductance value h provides the gate output. The architecture also uses $K+1$ resistors R_s. The conductance h is first initialized at HRS g_{OFF}. Next, if all g_k are set to HRS g_{OFF}, h will switch to LRS g_{ON} according to (2) (under proper choice of parameters V_s, V_0, g_{ON}, g_{OFF}, R_s). (b) NOR gate from MAGIC architecture [20]. Conductance levels g_1 to g_K store the K gate inputs, while conductance value h provides the gate output. Given that the conductance h is initialized at LRS g_{ON}, h will switch to HRS g_{OFF} only if all g_k are set to g_{ON}, according to (2) (under proper choice of parameters V_s, g_{ON}, g_{OFF}). (c) Generic logic gate from PLiM [21] and MOL architectures [21].

this solution often results in large circuit sizes needed to implement the logic function. Additionally, one of the major challenges of IMPLY and MAGIC resides in mapping a large number of logic gates into a single memristor crossbar for performing massively parallel operations, as discussed in [9], [20].

PLiM [21] and MOL [22] architectures also implement logic gates, but using only one memristor, as shown in Figure 4 (c). Both approaches utilize the Boolean formula $Q_{n+1} = Q_n \cdot A + Q_n \cdot \bar{B} + A \cdot \bar{B}$, which enables the implementation of any logic function by adjusting the logic values of A, B, and Q_n [21], [22]. This equation is computed in memory by converting Q_n into the memristor conductance value, where $Q_n = 1$, $Q_n = 0$, are mapped to conductance levels g_{ON}, g_{OFF}, respectively, while A and B are assigned as voltage levels $0V$ (for logic value 0) or uV (for logic value 1). According to (2), the voltage drop accross the memristor may update its internal conductance value, which is then read as Q_{n+1} in accordance with the previous Boolean equation. These architectures allow for the implementation of a large number of gates in one crossbar, where memristors in each WL share the same value A, and memristors in each BL share the same value B [22]. However, difficulties in this architecture reside in the dynamic aspect of evaluating output gate values Q_{n+1} from previous values Q_n, and the need for a secondary memristor crossbar to store useful input values that are erased in the computation process.

C. Hamming Distance Computation

Hamming distance computation is a fundamental operation on a pair of binary vectors, and has found broad use in large-scale data processing. Being able to execute this calculation and similar ones directly in memory could have substantial positive impact on the practical realization of in-memory computing. It was first shown

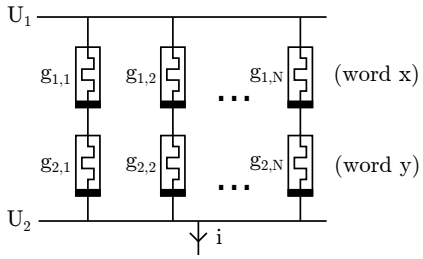

Figure 5. Hamming distance computation between two binary words $\mathbf{x}$ and $\mathbf{y}$ of length N. The bit values x_k and y_k are mapped into the conductance values $g_{1,k}$ and $g_{2,k}$, respectively, with $x_k = 1$, $y_k = 1$ correspond to g_{OFF} and $x_k = 0$, $y_k = 0$ correspond to g_{ON}. The Hamming distance $d_H(\mathbf{x}, \mathbf{y})$ is found from measuring the equivalent conductance $G_{1,2}$ of this circuit using the current-voltage relation $i = G_{1,2}(U_1 - U_2)$.

in [23] that the Hamming distance $d_H(\mathbf{x}, \mathbf{y})$ between two binary words $\mathbf{x}$ and $\mathbf{y}$ can be computed within an SLC crossbar, using *e.g.*, an idealized circuit shown in Figure 5. This circuit corresponds to two WLs of the crossbar, the first WL storing the bit values of $\mathbf{x}$, and the second one the bit values of $\mathbf{y}$. The caveat is that in memristors, one cannot directly compare the values of two bit cells in two different vectors. Rather, the access is permitted only at the vector level. It was recognized by Cassuto and Crammer in [23] that the Hamming distance between two binary vectors can be computed using a single measurement (or a small number of measurements) of the equivalent conductance $G_{1,2}$ of this circuit, provided that the conductance parameter (dictated by the memristor circuitry and design) is within a certain range. This measurement is then mathematically manipulated to yield the Hamming distance – the key is to keep track of how many positions the two vectors have pair-wise bit values ab, for $a, b \in \{0, 1\}$.

IV. Error Models

While the previous section introduced in-memory computation structures for several applications, the objective of this section is to present error models that can be utilized in the information-theoretic analysis of each computation structure, as well as to develop error-correction solutions relevant to each model.

A. Sources of Impairments

Despite the recent progress of in-memory computing, significant challenges remain in implementing large-scale IMC units with high accuracy, primarily due to the non-ideal properties of memory devices and circuits. First, fabrication process imperfection causes device-to-device variations within a single chip or across different chips in terms of the conductance ranges, device programmability, endurance, as well as data retention. Meanwhile, achieving a high level of precision in writing conductance values proves to be a challenging endeavor. This challenge stems from the dynamic nature of resistance variations, as described by equation (2). This issue becomes even more pronounced when considering in-memory computing with AC, as in the DPE computation discussed in Section III-A. In such scenarios, the difficulty to write each conductance at arbitrary real values introduces inherent noise into the computation. Even in the case of SLC implementations, such as those employed in logic-in-memory computing detailed in Section III-B, there exists a non-negligible risk of conductance failing to transition from one state to another, thereby introducing computational errors. In addition, there are also important variations in the time for switching from one state to another. Furthermore, the resistance drift over time and temperature, the read errors caused by random telegraph noise (RTN) [24], and the sneak path effect illustrated in Figure 2 can also affect computing accuracy. Finally, stuck-at-faults correspond to errors in which conductance values are permanently stuck in a certain state.

In addition to the noise affecting each memory cell, the inputs and outputs of the computation can also be subject to perturbations. Typically, in-memory computing will be required to have a digital interface. As such, the input will be driven by a DAC converter that is affected by mismatch, and the output will be obtained through an ADC converter that introduces quantization and clipping errors [25].

While most previous works have experimentally evaluated the impact of noise through noise injection on software models, hardware models, and on-board implementations, they have often utilized specific noise models that could also be incorporated in theoretical analyses. In this section, we present and discuss two sets of models that have been widely considered in the literature. The first set of models characterizes the noise introduced in the conductance values, which can capture transient effects that come from several factors such as device-to-device variations, inaccurate write or read operations, or resistance drift over time. Interestingly, these models are applicable to all SLC, MLC, and AC. The second set of models is specific to SLC and aims to capture switching errors that can occur in this type of device. Finally, we present error models for the interface of a DPE that is integrated in a digital system.

B. Continuous Noise Models

In what follows, we use $G_{j,k}$ to denote the stochastic version of the conductance value $g_{j,k}$. In [11], [26], empirical probability densities of current-voltage characteristics indicate that an additive Gaussian model is an appropriate representation of the noise affecting actual conductance values. The model is centered around the target conductance value, but the variance is shown to vary with the target conductance value as well. This additive Gaussian noise model is considered in [2] for analog DPE described in Section III-A, and in [27] for Hamming distance computation presented in Section III-C, where in both works each possible conductance state has its own variance value. For instance, in the SLC case considered in [27], the conductance values for G_{ON} and G_{OFF} are modeled as $\mathcal{N}(g_{\text{ON}}, \sigma^2_{\text{ON}})$ and $\mathcal{N}(g_{\text{OFF}}, \sigma^2_{\text{OFF}})$, respectively. On the other hand, a simplified Gaussian model with only one variance value for any possible conductance level is considered in [28], [29] for analog DPE with MLC described in Section III-A. This model represents the conductance values $G_{j,k}$ as $\mathcal{N}(g_{j,k}, \sigma^2)$.

Furthermore, [30] simulates conductance values with a more complex model that takes into account programming noise, conductance drift over time, and read noise. The three sources of noise are combined as follows. First, the programming noise produces a random variable $G^{(p)}_{j,k} = g_{j,k} + U_{j,k}$, where $U_{j,k}$ is a Gaussian random variable with mean 0 and variance σ^2_p. The conductance drift over time is modeled as $G^{(d)}_{j,k}(t) = G^{(p)}_{j,k}\left(\frac{t_0}{t}\right)^{\nu}$, where ν is a Gaussian random variable with mean μ_ν and variance σ^2_ν. Finally, the actual conductance value is computed as $G_{j,k}(t) = G^{(d)}_{j,k}(t) + V_{j,k}$, where the read noise $V_{j,k}$ is modeled as a Gaussian random variable with mean zero and variance σ^2_r. Although the three noise sources are considered to be statistically independent from one another, they are all taken into account in the resulting $G_{j,k}(t)$. Note that if the conductance drift over

time is instead modeled as a deterministic scaling by $\alpha(t)$, this model reduces to representing the conductance values $G_{j,k}$ as $\mathcal{N}(\alpha(t)g_{j,k},\ \alpha^2(t)\sigma_p^2 + \sigma_r^2)$.

C. SLC Error Models

The previous Gaussian models assume that the actual conductance value varies around its target value. However, sometimes memristors fail to switch to the target value. When considering SLC, the switching time is known to follow a log-normal distribution [31]. In addition, for Hamming distance computation [27] described in Section III-C, unsuccessful write operations are modeled at a higher level as a binary symmetric channel (BSC), which assumes that errors are not only symmetric but also independent and identically distributed from one memristor to another. Nevertheless, [32] has shown that errors are not identically distributed within the crossbar, meaning that the position of a memristor in the crossbar has an influence on the error distribution. Moreover, [33] assumes a correlation between the error realizations inside a crossbar. Of course, these generalizations could also be introduced in the continuous models. Finally, [34] demonstrates that it is more relevant to consider asymmetric errors. Following these concepts, [35] models read and write errors as binary asymmetric channels (BAC) with non-uniform transition probabilities across the crossbar. It is worth noting that [32], [34], [35] discuss error models and assumptions for storage applications, although these models are also relevant for in-memory computing.

D. Noise Models for the DPE Interface

When considering specifically the DPE structure illustrated in Figure 3, the noise model can be extended to the conversion circuits that are required to integrate the DPE into a digital system. The most important aspect to be modeled is the fact that the digital-to-analog conversion at the WL inputs may be inaccurate, which can be modeled as additive Gaussian noise with input-dependent variance [25]. This input noise has a different impact on the computation compared to conductance variations, since it affects all the elements of the output vector. However, it is interesting to note that DAC circuits are not essential to implement a DPE. Instead, each bit of the quantized input elements can be presented to the crossbar one at a time, allowing to perform the dot-product over repeated uses of the crossbar [14].

On the other hand, the analog-to-digital conversion introduces the usual quantization and clipping errors. A detailed error model should also consider that the quantization range may be narrower or wider than intended, potentially introducing additional clipping or quantization errors.

V. Information-Theoretic Analysis

In information theory, the key notion of capacity provides the fundamental limit of reliable communication over a noisy channel, where reliability is defined as vanishing error probability under asymptotic code length. This fundamental result has long motivated the search for practical and efficient channel codes with performance close to the capacity.

The information-theoretic analysis of the capacity can be straightforwardly applied to noisy storage systems. This is because the memory noise can be factored into a standalone channel model, which functions independently of both encoding and decoding processes. Yet, when it comes to in-memory computing, this method is less applicable. Take logic-in-memory as an example, where each logic gate is subject to noise. To improve reliability, one may expand the circuit, by incorporating error-correction mechanisms. However, these components might bring along their own inherent noise during processing, resulting in a tradeoff between noise introduction and noise correction. Given such strong interplay between noise and computation, the notion of capacity cannot be used as is. There is a need for alternative concepts which we discuss in this section.

It is worth noting that for in-memory computing, each computational structure outlined in Section III might necessitate its own unique analytical approach. Particularly, the very different nature of these computation structures may require different analysis tools. In addition, the evaluation criterion may differ: while error probability is appropriate for logic-in-memory or Hamming distance computation, each specific application of dot-product computation may need its own specific metrics. For instance, [36] proposed a method to significantly improve the error-correction performance of a noisy DNN, by optimizing the final classification accuracy instead of the usual bit error rate. Such results motivate us to also discuss the evaluation of relevant reliability criteria for the computation structures described in Section III.

A. An Equivalent of the Capacity for IMC

In this section, we present appropriate definitions of the capacity in order to investigate the fundamental limits of noisy IMC systems.

1) Redundancy of IMC Systems: The redundancy of a noisy circuit is defined as the number of noisy gates needed to perform the computation within the target

error probability, divided by the number of gates in the noiseless circuit [37]. This definition was introduced for noisy circuits in general, not necessarily those implemented in memory, and it allows to account for an error-correction mechanism which would increase the size of the circuit. But one key difficulty is that the circuit implementation has an influence both on the number of gates in the circuit, and on the robustness to noise [38]. This observation largely explains why there only exist lower bounds on the redundancy of noisy logic circuits [39], [40].

For noisy in-memory computing, redundancy can be regarded as a first proxy in comparing different error-correction solutions [36], but it fails to take into account some specific mechanisms, like increasing the conductance values to improve the robustness to noise [29], or including golden gates with a lower error probability than standard ones in order to improve the overall computation performance [38]. In addition, it does not account for the cost of the accompanying CMOS operations, including DACs and ADCs, which are often used together with in-memory computing. Therefore, the information theory analysis could go one step further in providing insightful design rules, by considering more practical concepts such as the power consumption of the in-memory computing units.

2) Power Consumption of IMC Systems: Outside of in-memory computing, some works have already proposed to use information theory to investigate the power consumption of computing systems. For instance, while the capacity of an additive white Gaussian noise (AWGN) channel depends on the transmit power, [41] proposed to also investigate the decoder power supply, which for short-distance communications becomes non-negligible compared to the transmit power. For the special case of LDPC codes, [41] provides a theoretical evaluation of the decoder power supply, by considering abstract models for the power requirements of wires and processors. It then derives scaling laws relating the power supply to the decoder error probability, and shows that under certain conditions, the decoding power is so high that uncoded transmission should be preferred.

Although the analysis of [41] is specific to LDPC decoders, it gives insights that using information theory to investigate the power consumption of IMC systems may be very relevant from a hardware implementation perspective. In the case of in-memory computing, an information-theoretic analysis of the power consumption would allow to take into account everything at once: not only the additional redundancy, but also all accompanying operations (DACs/ADCs, etc.). Such analysis would also permit to fairly compare different strategies to improve reliability, such as error-correction mechanisms, increase in conductance levels, etc. On the other hand, the power consumption of in-memory computing systems can be difficult to analyze, since it strongly depends on the computation structure, and on the way it is implemented.

Interestingly, the power required for analog DPE in memory described in Section III-A can be evaluated using conventional electrical power formulas [29]. For instance, the power consumption of a resistive device with internal conductance $G_{j,k}$ can be calculated using the formula $P_{j,k} = G_{j,k}u_j^2$, where u_j is the input voltage. When considering stochastic conductance values $G_{j,k}$ due to the noise, it is also useful to evaluate the average total power consumption by summing up the average power contributions of each resistive device in the crossbar.

The power consumption of a memristor crossbar depends on various factors including the crossbar size and the conductance levels. In particular, larger conductance values can improve the signal-to-noise ratio and make the system more robust to noise, at the price of increased power. Therefore, it is proposed in [29] to optimize the conductance values so as to achieve the best tradeoff between robustness to noise and power consumption.

The analysis of [29] only considers the power requirements of the crossbar memristors and TIAs shown in Figure 3. However, there is a need to also take into account the DACs and ADC needed for the computation. In addition, in many applications of in-memory computing, a part of the computation operation are still realized in CMOS, which should also be taken into account in the power analysis for a fair comparison of different error-correction mechanisms.

B. Evaluating Reliability Criteria

Investigating the capacity, or a similar notion, of IMC systems, requires to set a reliability criterion. We now discuss relevant reliability criteria for in-memory computing, depending on the considered computation structure and noise model.

1) Noisy Logic Computation: When considering noisy logic-in-memory, a relevant performance evaluation criterion is the error probability at the output of the circuit. Existing information-theoretic analysis of noisy logic circuits [37], [42], [43] can be adapted to evaluate such error probability in the case of in-memory computing. These works consider a logic circuit that implements a certain function $f(.)$ with k inputs and m outputs, and the noise is modeled as bit-flipping probabilities in each logic gate of the circuit. In this case, the function

output error probability can be calculated exactly by using the Gallager formula [43] which provides the probability of an odd number of gate flips. Note that it is often impossible to achieve zero error probability, even asymptotically, given that the final gates introduce a level of errors bounded away from zero.

The bit-flipping noise model used in previous works [37], [42], [43] is equivalent to the switching error model described by a BSC presented in Section IV-C. Therefore, the information-theoretic analysis of noisy logic circuits can be seen as a first proxy to investigate noisy logic-in-memory. Conversely, when considering continuous noise models described in Section IV-B, there is a need to update the previous theoretical analysis.

2) Hamming Distance Computation: For the Hamming distance computation described in Section III-C, the error probability is still a relevant criterion, although its theoretical evaluation differs. The work of [23] provided lower bounds on the minimum ratio $g_{\text{ON}}/g_{\text{OFF}}$ that allow to compute exactly the Hamming distance between two vectors, following the computation model of Section III-C. It was also recognized by [23] that channel coding in the form of adding a particular type of redundancy to the stored vectors, can enable a wider range of memristor parameters g_{ON} and g_{OFF}, thus enabling less restrictive hardware designs. This work was then subsequently expanded in [27], where a more realistic statistical model of memristor variability was considered. The work [27] considered two main - and complementary - sources of memristor variability: resistance variation, and the non-deterministic switching mechanism during the memristor write process. Leveraging prior literature on device physics (see Sections IV-B and IV-C), the former type of variation was modeled in [27] as a per-state (high or low) Gaussian random variable and the latter was modeled as a Bernoulli random variable. Statistical description of the resultant Hamming distance was provided, in the form of equivalent Gaussian noise, with known mean and variance, onto the true Hamming distance.

3) MSE of Noisy Analog Dot Products: When considering the analog DPE described in Section III-A, one may investigate the effect of noise by evaluating the mean squared error (MSE) between the noisy DPE outputs C_j and their noiseless counterparts c_j. For this problem, [28], [29], [44] proposed to theoretically evaluate the MSE from analytical formulas of the first and second-order moments $\mathbb{E}[C_j]$ and $\mathbb{V}[C_j]$ of the noisy outputs C_j. These works consider that the means $\mathbb{E}[G_{j,k}]$ and variances $\mathbb{V}[G_{j,k}]$ are known, but do not make any further assumption on the probability distributions of the stochastic conductance values $G_{j,k}$. The Gaussian model described in Section IV-B fits into these assumptions, and it is considered as a special case in [28], [29], [44]. The moments $\mathbb{E}[C_j]$ and $\mathbb{V}[C_j]$ were next either calculated exactly [44] or approximated from second-order Taylor expansions [28], [29].

This analysis was further extended to predict the MSE after fixed-point computation [28] or at the output of a DNN, by propagating the analytical moment expressions over the successive layers of the network [29]. In the latter case, moments after the non-linear computation parts were approximated from the second-order Taylor expansions. In addition, a key challenge is that the noisy outputs after each DNN layer are correlated, as they are calculated from the same inputs. Therefore, analytical expressions of the covariance terms $\mathbb{C}[C_j C_{j'}]$ must also be propagated over the network.

Interestingly, all previous works show that the theoretical expressions of the MSE predict the actual MSE measured from Monte-Carlo simulations with high accuracy, even when the successive moments are approximated from Taylor expansions. In addition, when considering DNNs, the non-linear activation functions are shown to reduce error propagation from one layer to another. Further, by comparing the results of [29] and [44], it can be observed that the widely considered DPE computation model in which $a_{j,k} = g_{j,k}$ is inherently less robust to noise than the computation model of [4] in which $a_{j,k} = \frac{g_{j,k}}{g_s + \sum g_{j,k}}$.

Finally, while the MSE gives a measure of the amount of noise that remains at the end of the computation, there is also a need to consider a criterion more specific to the targeted application, for instance the accuracy for classification from DNNs.

C. Additional Challenges

In addition to the previously mentioned computation structures (such as DPE, logic-in-memory, Hamming distance computation, and DNN inference), there is a need to explore additional applications of interest. These include tasks like calculating the shortest path [5], training deep neural networks (DNN training), and implementing binary neural networks (BNN). These applications may require specific information-theoretic analyses due to their unique computational structures or performance evaluation criteria. Furthermore, certain sources of impairment, such as the sneak-path effect or stuck-at-faults, may necessitate dedicated analyses. This is because they do not conform to the typical statistical assumptions required by most information theory analyses, such as the assumptions of independent and identically distributed (i.i.d.) noise and independence

with respect to inputs. Finally, an information-theoretic analysis of IMC systems may investigate their energy efficiency by not only considering the power consumption of the full system (including ADC and DAC, resistance programming, etc.), but also the computation latency.

VI. Advanced Robust Designs

The previous section discussed information-theoretic analysis of in-memory computing, and in particular the prediction of the power consumption of IMC systems. This section now addresses the practical robust design of power-efficient in-memory computing systems. While simple inversion coding was shown to be sufficient in the context of the Hamming distance computation [27], this section describes more advanced error-correction techniques that are necessitated by various applications.

It is well known that error correction coding techniques have found broad use in data storage and communication systems for many decades. What is perhaps less known is that they have an equally long history in being used for reliable computation. For example, arithmetic codes named AN codes [45] were proposed back in 1950s for correcting computational errors. As in communication systems, error-correction codes (ECCs) may allow to further reduce the power consumption of in-memory computing [24], [46]. However, as for the theoretical analysis, the code design should depend on the considered computation structure and error model. Especially, the sneak-path effect may necessitate its own specific code design.

A. ECCs Defined Over the Integer Set

In [47], algebraic coding schemes defined over the subset of integer set $\mathbb{Z}$ were proposed. These codes can detect and correct computational errors occurring for computing integer vector-matrix multiplication (VMM) $\mathbf{c} = \mathbf{u}A$ where all the components of $\mathbf{u}$ and A are integers [48]. Such integer VMMs can be implemented in the same way as analog DPE described in Section III-A.

The accuracy of the computation can be affected by several factors, such as inaccuracies of programming the resistors in the crossbar and noise in reading the currents (see Section IV-A). Therefore, the error vector $\mathbf{e}$ is defined as $\mathbf{e} = \mathbf{y} - \mathbf{c}$, with $\mathbf{y}$ being the actual integer-valued vector that was read. It is natural to consider the L_1-norm to characterize the errors, where the L_1-norm of a vector is the sum of the absolute values of its elements. Alternatively, errors could be caused by the shorted or stuck cells in the memory array. In this case, it is more convenient to use the Hamming metric to model these errors, where the Hamming distance between the erroneous reading and the correct output vector equals the number of positions in which their values differ. Therefore, the code constructions in [47] make use of either the L_1-metric or the Hamming metric, for both single and multiple error correction. For correcting τ errors within a codeword of length n, the required redundancy is on the order of $\tau \cdot \log_q(n)$ for the case with the L_1-metric, and it is approximately $2\tau \cdot \log_q(n)$ for the case with the Hamming metric. In [47], a lower bound is provided on the code redundancy needed to correct a single error in the L_1 metric. It is then shown that the redundancy of the proposed code construction is within one symbol from this lower bound. However, the optimal redundancy of codes correcting multiple errors is still an open problem.

B. Analog Error-Correcting Codes

Different from the exact integer VMM model adopted by [47], the coding schemes proposed by [46] considers the computation model for analog DPE introduced in Section III-A. In this case, both the ideal VMM computation $\mathbf{c} = \mathbf{u}A$ and the actual read vector $\mathbf{y}$ are in the analog domain. Then, [46] considers an approximate computation model in which entries of $\mathbf{y}$ are considered "correct" if they are sufficiently close to the respective entries of $\mathbf{c}$. However, entries in $\mathbf{y}$ that are significantly different from those in $\mathbf{c}$ are called outlying errors and they must be corrected. More specifically, the read vector $\mathbf{y}$ is given by $\mathbf{y} = \mathbf{c} + \boldsymbol{\varepsilon} + \mathbf{e}$, where the entries in $\boldsymbol{\varepsilon}$ represent small computational errors or the circuit noise, which are tolerable, while the entries of $\mathbf{e}$ denote the outlying errors, which may be caused by the short cells or stuck cells in the memory array. It is assumed that the values of the tolerable errors are within the interval $[-\delta, \delta]$, while entries of the outlying errors that can be corrected fall outside the interval $[-\Delta, \Delta]$, where $\delta < \Delta$ are two positive real numbers. Such an approximate computation model is suitable for applications that can tolerate small errors, such as learning applications.

A linear $[n, k]$ code $\mathcal{C}$ defined over the real number domain $\mathbb{R}$ is said to be capable of correcting τ (outlying) errors and detecting σ additional (outlying) errors if there is a decoder of $\mathcal{C}$ which, for every read vector $\mathbf{y}$ with at most $\tau + \sigma$ outlying errors, returns a set of locations of outlying errors or an indication that errors have been detected.

The error correction capability of the above analog ECC $\mathcal{C}$ can be characterized by its height profile, which is defined as follows. Suppose $\mathbf{c}$ is a codeword of $\mathcal{C}$ and it is a nonzero vector. The entries of $\mathbf{c}$ are sorted according to descending absolute values as $|c_{\pi(0)}| \geq$

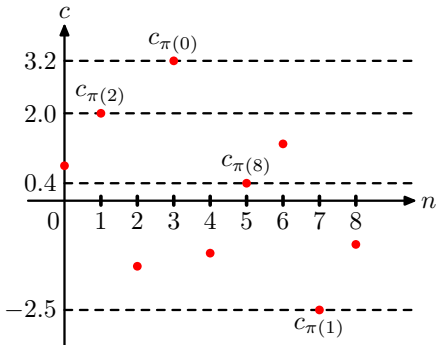

Figure 6. An example of the m-height of a real vector $\mathbf{c}$ of length 9: We can see that $|c_3| > |c_7| > |c_1| > \cdots > |c_5|$, so $\pi(0) = 3$, $\pi(1) = 7$, $\pi(2) = 1, \cdots$, and $\pi(8) = 5$. According to the definition, $\mathsf{h}_0(\mathbf{c}) = |\frac{c_3}{c_3}| = 1$, $\mathsf{h}_1(\mathbf{c}) = |\frac{c_3}{c_7}| = \frac{3.2}{2.5} = 1.28$, $\mathsf{h}_2(\mathbf{c}) = |\frac{c_3}{c_1}| = \frac{3.2}{2.0} = 1.6, \cdots, \mathsf{h}_8(\mathbf{c}) = |\frac{c_3}{c_5}| = \frac{3.2}{0.4} = 8$.

$|c_{\pi(1)}| \geq |c_{\pi(2)}| \geq \cdots \geq |c_{\pi(n-1)}|$, where π denotes the corresponding permutation of the coordinates of $\mathbf{c}$ (see Fig. 6 for an example). For each $m \in \{0, 1, \cdots, n-1\}$, the m-height of $\mathbf{c}$, denoted by $\mathsf{h}_m(\mathbf{c})$, is defined as the ratio of $|c_{\pi(0)}|$ to $|c_{\pi(m)}|$, i.e., $\mathsf{h}_m(\mathbf{c}) = |\frac{c_{\pi(0)}}{c_{\pi(m)}}|$. The m-height of $\mathcal{C}$, denoted by $\mathsf{h}_m(\mathcal{C})$, is defined as the largest value of $\mathsf{h}_m(\mathbf{c})$ among all $\mathbf{c} \in \mathcal{C}$, where $\mathsf{h}_m(\mathbf{0}) = 0$ for every $m \geq 0$. In [46], a necessary and sufficient condition for the existence of the decoder of $\mathcal{C}$ is presented. That is, the decoder of $\mathcal{C}$ can correct τ errors and detect σ additional errors with respect to the threshold pair (δ, Δ) if and only if $2(\mathsf{h}_m(\mathcal{C}) + 1) \leq \frac{\Delta}{\delta}$, with $m = 2\tau + \sigma$. Given n, k and $m = 2\tau + \sigma$, a basic problem is to design an $[n, k]$ code $\mathcal{C}$ with $\mathsf{h}_m(\mathcal{C})$ as small as possible.

The analog ECC proposed in [46] can detect or correct a single outlying error. For encoding, the programmed matrix A is designed in the form $A = (A', A'')$, and hence the encoded codeword is given by $\mathbf{c} = (\mathbf{c}', \mathbf{c}'')$, where $\mathbf{c}' = \mathbf{u}A'$ is the expected result (i.e. the target computation), while $\mathbf{c}'' = \mathbf{u}A''$ is the redundancy part for detecting or correcting the outlying errors. Given any linear $[n, k]$ code $\mathcal{C}$ over $\mathbb{R}$, if $G = (I_k, G')$ is a systematic generator matrix of $\mathcal{C}$ with I_k being the identity matrix of order k, A'' can be determined as $A'' = A'G'$. That is, the encoding is done by adding in a sub-matrix A'' next to the matrix A' for VMM operation, which can be implemented using the memory crossbar array in parallel to the array for VMM. Therefore, the encoding can be carried out simultaneously with the VMM operation and it does not incur additional latency. For decoding, single error detection or correction can be achieved by examining the entries of the syndrome $\mathbf{s} = H\mathbf{y}^{\mathrm{T}}$ of the read vector based on a specifically designed parity-check matrix H of $\mathcal{C}$ such that the necessary and sufficient condition for the existence of its decoder is satisfied.

The effectiveness of the constructed single-outlying-error-correction code was demonstrated at the circuit level by [49]. However, the construction of analog ECCs that can locate/correct more outlying errors, as well as the analysis of the bound of the magnitude of the outlying errors that can be corrected remain challenging open problems.

C. Additional Challenges

Ultimately, when developing ECC for IMC systems, one should target to improve not only the robustness, but also the energy efficiency of the computation. In addition to the previous code constructions, a key question is to determine whether modern codes, such as LDPC and Polar, which can correct a large amount of errors, are suitable for energy-efficient in-memory computing with strict latency and complexity constraints. A first conceptual error-correction solution based on LDPC codes for DPE was proposed in [24], although it does not yet satisfy these constraints.

Additionally, complementary to the hardware-based solutions are signal processing and communication methods that can help further optimize the underlying devices. For instance [50], [51] investigated the sneak-path effect from a classical communication theory viewpoint, proposing either a simple pilot construction or an optimal detector to mitigate this effect. More recently, [52] developed a new quantized channel model for the ReRAM array that incorporates both the sneak path interference and the random resistance variation caused by process variations. Next, by maximizing mutual information of the induced channel, [52] developed adaptive sneak path-aware detection and decoding schemes that can successfully operate with only a few bits of quantization. The previous solutions [50]–[52] were investigated in the context of data storage only, and it would be interesting to adapt them to in-memory computing.

VII. Conclusion

This paper provided an overview of in-memory computing systems, with the aim of connecting the fields of information theory and hardware implementation. The paper emphasizes the importance of considering accurate noise models, as well as hardware implementation constraints such as latency and power, in the theoretical analysis of in-memory computing systems. Additionally, the analysis and design of in-memory computing systems should be done from the perspective of the target applications, an idea which is now being intensively explored

in the field of goal-oriented communications, where the communication objective is to address a given learning task (decision making, classification, etc.), rather than data reconstruction.

REFERENCES

[1] G. E. Moore, "No exponential is forever: but forever can be delayed!" in *IEEE Int. Solid-State Circuits Conf. (ISSCC). Digest of Technical Papers.*, 2003, pp. 20–23.

[2] V. Joshi, M. Le Gallo, S. Haefeli, I. Boybat, S. R. Nandakumar, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian, and E. Eleftheriou, "Accurate deep neural network inference using computational phase-change memory," *Nature comm.*, vol. 11, no. 1, pp. 1–13, 2020.

[3] B. Li, B. Yan, and H. Li, "An overview of in-memory processing with emerging non-volatile memory for data-intensive applications," in *Proc. of the Great Lakes Symp. on VLSI*, 2019, pp. 381–386.

[4] S. Liu, Y. Wang, M. Fardad, and P. K. Varshney, "A memristor-based optimization framework for artificial intelligence applications," *IEEE Circ. and Syst. Mag.*, vol. 18, no. 1, pp. 29–44, 2018.

[5] D. Stathis, I. Vourkas, and G. C. Sirakoulis, "Shortest path computing using memristor-based circuits and cellular automata," in *Int. Conf. on Cellular Automata*. Springer, 2014, pp. 398–407.

[6] J. Kendall, R. Pantone, K. Manickavasagam, Y. Bengio, and B. Scellier, "Training end-to-end analog neural networks with equilibrium propagation," *arXiv preprint arXiv:2006.01981*, 2020.

[7] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature nanotech.*, vol. 15, no. 7, pp. 529–544, 2020.

[8] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, "Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication," in *53nd ACM/EDAC/IEEE design autom. conf. (DAC)*, 2016, pp. 1–6.

[9] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (IMPLY) logic: design principles and methodologies," *IEEE Trans. on Very Large Scale Int. (VLSI) Syst.*, vol. 22, no. 10, pp. 2054–2066, 2013.

[10] F. Merrikh-Bayat, X. Guo, M. Klachko, M. Prezioso, K. K. Likharev, and D. B. Strukov, "High-performance mixed-signal neurocomputing with nanoscale floating-gate memory cell arrays," *IEEE Trans. on Neural Net. and Learning syst.*, vol. 29, no. 10, pp. 4782–4790, 2017.

[11] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan, "Computing in memory with spin-transfer torque magnetic RAM," *IEEE Trans. on Very Large Scale Int. (VLSI) Syst.*, vol. 26, no. 3, pp. 470–483, 2018.

[12] M. Le Gallo and A. Sebastian, "An overview of phase-change memory device physics," *Journal of Physics D: Applied Physics*, vol. 53, no. 21, p. 213002, 2020.

[13] L. Ni, H. Huang, Z. Liu, R. V. Joshi, and H. Yu, "Distributed in-memory computing on binary RRAM crossbar," *ACM Journal on Emerging Tech. in Comp. Syst. (JETC)*, vol. 13, no. 3, pp. 1–18, 2017.

[14] T. Soliman, F. Müller, T. Kirchner, T. Hoffmann, H. Ganem, E. Karimov, T. Ali, M. Lederer, C. Sudarshan, T. Kämpfe, A. Guntoro, and N. Wehn, "Ultra-low power flexible precision fefet based analog in-memory computing," in *IEEE Inte. Electron Devices Meeting (IEDM)*, 2020, pp. 29.2.1–29.2.4.

[15] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[16] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Team: Threshold adaptive memristor model," *IEEE Trans. on Circ. and Syst. I: reg. papers*, vol. 60, no. 1, pp. 211–221, 2012.

[17] H. Lv, X. Xu, H. Liu, R. Liu, Q. Liu, W. Banerjee, H. Sun, S. Long, L. Li, and M. Liu, "Evolution of conductive filament and its impact on reliability issues in oxide-electrolyte based resistive random access memory," *Scientific reports*, vol. 5, no. 1, pp. 1–6, 2015.

[18] S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. Di Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. Farinha *et al.*, "Equivalent-accuracy accelerated neural-network training using analogue memory," *Nature*, vol. 558, no. 7708, pp. 60–67, 2018.

[19] S. Shukla, S. Balasubramanian, and M. Pavlović, "A generalized banach fixed point theorem," *Bulletin of the Malaysian Mathematical Sciences Society*, vol. 39, pp. 1529–1539, 2016.

[20] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using memristor-aided logic (MAGIC)," *IEEE Trans. on Nanotech.*, vol. 15, no. 4, pp. 635–650, 2016.

[21] P.-E. Gaillardon, L. Amarú, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. De Micheli, "The programmable logic-in-memory (PLiM) computer," in *Design, Autom. & Test in Europe Conf. & Exhibit. (DATE)*. IEEE, 2016, pp. 427–432.

[22] K. A. Ali, M. Rizk, A. Baghdadi, J.-P. Diguet, J. Jomaah, N. Onizawa, and T. Hanyu, "Memristive computational memory using memristor overwrite logic (MOL)," *IEEE Trans. on Very Large Scale Int. (VLSI) Syst.*, vol. 28, no. 11, pp. 2370–2382, 2020.

[23] Y. Cassuto and K. Crammer, "In-memory Hamming similarity computation in resistive arrays," in *IEEE Int. Symp. on Inf. Th. (ISIT)*, 2015, pp. 819–823.

[24] Q. Lou, T. Gao, P. Faley, M. Niemier, X. S. Hu, and S. Joshi, "Embedding error correction into crossbars for reliable matrix vector multiplication using emerging devices," in *Proc. of the ACM/IEEE Int. Symp. on Low Power Elec. and Design*, 2020, pp. 139–144.

[25] S. K. Roy, A. Patil, and N. R. Shanbhag, "Fundamental limits on the computational accuracy of resistive crossbar-based in-memory architectures," in *2022 IEEE Int. Symp. on Circ. and Syst. (ISCAS)*, 2022, pp. 384–388.

[26] K. V. Pham, S. B. Tran, T. V. Nguyen, and K.-S. Min, "Asymmetrical training scheme of binary-memristor-crossbar-based neural networks for energy-efficient edge-computing nanoscale systems," *Micromachines*, vol. 10, no. 2, p. 141, 2019.

[27] Z. Chen, C. Schoeny, and L. Dolecek, "Hamming distance computation in unreliable resistive memory," *IEEE Trans. on Comm.*, vol. 66, no. 11, pp. 5013–5027, 2018.

[28] E. Dupraz and L. R. Varshney, "Noisy in-memory recursive computation with memristor crossbars," in *IEEE Int. Symp. on Inf. Theory (ISIT)*, 2020, pp. 804–809.

[29] E. Dupraz, L. R. Varshney, and F. Leduc-Primeau, "Power-efficient deep neural networks with noisy memristor implementation," in *IEEE Inf. Th. Workshop (ITW)*, 2021, pp. 1–5.

[30] M. Le Gallo, S. Nandakumar, L. Ciric, I. Boybat, R. Khaddam-Aljameh, C. Mackin, and A. Sebastian, "Precision of bit slicing with in-memory computing based on analog phase-change memory crossbars," *Neuro. Comp. and Eng.*, vol. 2, no. 1, p. 014009, 2022.

[31] G. Medeiros-Ribeiro, F. Perner, R. Carter, H. Abdalla, M. D. Pickett, and R. S. Williams, "Lognormal switching times for titanium dioxide bipolar memristors: origin and resolution," *Nanotech.*, vol. 22, no. 9, p. 095702, 2011.

[32] A. Chen, "A comprehensive crossbar array model with solutions for line resistance and nonlinear device characteristics," *IEEE Trans. on Elec. Devices*, vol. 60, no. 4, pp. 1318–1326, 2013.
[33] M. Hu, H. Li, Q. Wu, G. S. Rose, and Y. Chen, "Memristor crossbar based hardware realization of BSB recall function," in *The 2012 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2012, pp. 1–7.
[34] M. Zorgui, M. E. Fouda, Z. Wang, A. M. Eltawil, and F. Kurdahi, "Non-stationary polar codes for resistive memories," in *IEEE Global Comm. Conf. (GLOBECOM)*, 2019, pp. 1–6.
[35] Z. Chen and L. Dolecek, "Coding schemes for crossbar resistive memory with high line resistance in SCM applications," in *Int. Symp. on Topics in Coding (ISTC)*, 2021, pp. 1–5.
[36] K. Huang, P. H. Siegel, and A. Jiang, "Functional error correction for robust neural networks," *IEEE Journal on Selected Areas in Inf. Th.*, vol. 1, no. 1, pp. 267–276, 2020.
[37] M. G. Taylor, "Reliable computation in computing systems designed from unreliable components," *Bell Syst. Tech. Journal*, vol. 47, no. 10, pp. 2339–2366, 1968.
[38] E. Dupraz, V. Savin, S. K. Grandhi, E. Popovici, and D. Declercq, "Practical LDPC encoders robust to hardware errors," in *IEEE Int. Conf. on Comm. (ICC)*, 2016, pp. 1–6.
[39] P. Gács and A. Gál, "Lower bounds for the complexity of reliable boolean circuits with noisy gates," *IEEE Trans. on Inf. Theory*, vol. 40, no. 2, pp. 579–583, 1994.
[40] N. Pippenger, "On networks of noisy gates," in *26th Ann. Symp. on Found. of Comp. Sci. (sfcs 1985)*. IEEE, 1985, pp. 30–38.
[41] K. Ganesan, P. Grover, J. Rabaey, and A. Goldsmith, "On the total power capacity of regular-ldpc codes with iterative message-passing decoders," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 2, pp. 375–396, 2015.
[42] N. Pippenger, G. D. Stamoulis, and J. N. Tsitsiklis, "On a lower bound for the redundancy of reliable networks with noisy gates," *IEEE Trans. on Inf. Theory*, vol. 37, no. 3, pp. 639–643, 1991.
[43] Y. Yang, P. Grover, and S. Kar, "Computing linear transformations with unreliable components," *IEEE Transactions on Information Theory*, vol. 63, no. 6, pp. 3729–3756, 2017.
[44] J. Kern, S. Henwood, G. Mordido, E. Dupraz, A. Aïssa-El-Bey, Y. Savaria, and F. Leduc-Primeau, "MemSE: Fast MSE prediction for noisy memristor-based DNN accelerators," in *IEEE 4th Int. Conf. on Artif. Intelligence Cir. and Syst. (AICAS)*, 2022, pp. 62–65.
[45] J. Diamond, "Checking codes for digital computers," in *Proc. of IRE*, 1995, pp. 487–488.
[46] R. M. Roth, "Analog error-correcting codes," *IEEE Trans. on Inf. Theory*, vol. 66, no. 7, pp. 4075–4088, 2020.
[47] ——, "Fault-tolerant dot-product engines," *IEEE Trans. on Inf. Th.*, vol. 65, no. 4, pp. 2046–2057, 2019.
[48] M. H. *et al.*, "Memristor-based analog computation and neural network classification with a dot product engine," *Advanced Materials*, vol. 30, no. 9, p. 1705914, 2018.
[49] C. Li, R. M. Roth, C. Graves, X. Sheng, and J. P. Strachan, "Analog error correcting codes for defect tolerant matrix multiplication in crossbars," in *2020 IEEE Int. Electron Devices Meet. (IEDM)*, 2020, pp. 1–4.
[50] Z. Chen, C. Schoeny, and L. Dolecek, "Pilot assisted adaptive thresholding for sneak-path mitigation in resistive memories with failed selection devices," *IEEE Trans. on Comm.*, vol. 68, no. 1, pp. 66–81, 2019.
[51] Y. Ben-Hur and Y. Cassuto, "Detection and coding schemes for sneak-path interference in resistive memory arrays," *IEEE Trans. on Comm.*, vol. 67, no. 6, pp. 3821–3833, 2019.
[52] P. Li, K. Cai, G. Song, and Z. Mei, "Sneak path interference-aware adaptive detection and decoding for resistive memory arrays," *IEEE Comm. Lett.*, vol. 26, no. 9, pp. 2032–2036, 2022.

Elsa Dupraz was born in Paris, France. She received the Master degree in advanced systems of radio-communications from ENS Cachan and University Paris Sud, France, in 2010, and the Ph.D. degree in physics from University Paris-Sud, France, in 2013. From January 2014 to September 2015, she held a Postdoctoral position with ETIS (ENSEA, University Cergy-Pointoise, CNRS, France) and ECE Department, University of Arizona, USA. Since October 2015, she has been an Assistant Professor with IMT Atlantique. Her research interests lie in the area of coding and information theory, with current interest on source/channel coding for Machine Learning applications, in-memory computing, and DNA data storage.

François Leduc-Primeau is an Associate Professor with the Department of Electrical Engineering at Polytechnique Montreal, Montreal, Canada. He received the B.Eng., M.Eng., and Ph.D. degrees in electrical & computer engineering from McGill University, Montreal, Canada. His research interests span digital system design, telecommunications, and machine learning, with applications in next-generation wireless communication systems, energy-efficient artificial intelligence, and other low-energy computing systems. He is an IVADO Professor, and he has served in the organization of various research events, such as the 2021 International Symposium on Topics in Coding (ISTC).

Kui Cai is an Associate Professor with the Science, Mathematics and Technology Cluster at Singapore University of Technology and Design (SUTD). She received her B.E. degree in information and control engineering from Shanghai Jiao Tong University, China and joint Ph.D. degree in electrical engineering from Technical University of Eindhoven, The Netherlands, and National University of Singapore. She received 2008 IEEE Communications Society Best Paper Award in Coding and Signal Processing for Data Storage and was listed in the 2020 Who's Who in Engineering Singapore. Her main research interests are in the areas of coding theory, information theory, and signal processing for emerging data storage systems and computing.

Lara Dolecek is a professor of Electrical and Computer Engineering and (by courtesy) of Mathematics at UCLA. Prof. Dolecek earned a B.S., M.S., Ph.D. degrees in EECS as well as an M.A. degree in Statistics from UC Berkeley. She is the 2019-21 Secretary, a 2021-22 Distinguished Lecturer, and a 2022-24 Board Member of the IEEE Information Theory Society. Prof. Dolecek received IBM Faculty Award, Intel Early Career Award, Okawa Research Grant, and NSF CAREER Award, among others, and with her research group and collaborators, she received over dozen best paper awards on topics in theory and applications of channel coding methods.