Memory devices and applications for in-memory computing

Abu Sebastian, Manuel Le Gallo, Riduan Khaddam-Aljameh, and Evangelos Eleftheriou
IBM Research – Zurich, Säumerstrasse 4, 8803 Rüschlikon, Switzerland

(Dated: February 6, 2020)

Traditional von Neumann computing systems involve separate processing and memory units. However, data movement is costly in terms of time and energy and this problem is aggravated by the recent explosive growth in highly data-centric applications related to artificial intelligence. This calls for a radical departure from the traditional systems and one such non-von Neumann computational approach is in-memory computing. Hereby certain computational tasks are performed in place in the memory itself by exploiting the physical attributes of the memory devices. Both charge-based and resistance-based memory devices are being explored for in-memory computing. In this Review, we provide a broad overview of the key computational primitives enabled by these memory devices as well as their applications spanning scientific computing, signal processing, optimization, machine learning, deep learning and stochastic computing.

Today's computing systems are primarily built based on the von Neumann architecture where data must be moved to a processing unit. During the execution of various computational tasks, large amounts of data need to be shuttled back and forth between the processing and memory units and this incurs significant costs in latency and energy. The latency associated with accessing data from the memory units is a key performance bottleneck for a range of applications, in particular for the increasingly prominent artificial intelligence (AI) related workloads. There is an increasing disparity between the speed of the memory and processing units, typically referred to as the memory wall[1]. The energy cost of moving data is another significant challenge given that the computing systems are severely power limited due to cooling constraints as well as the proliferation of mobile computing devices. Even at the relatively old 45 nm complementary metal oxide semiconductor (CMOS) node, the cost of multiplying two numbers is orders of magnitude lower than that of accessing them from memory [2]. The current approaches, such as the use of hundreds of processors in parallel (eg. graphics processing units [3]) or application-specific processors [4, 5] that are custom designed for specific applications, are not likely to fully overcome the challenge of data movement. Hence, it is becoming increasingly evident that novel architectures need to be explored where memory and processing are better collocated. One prominent idea that dates to the 1990's is that of physically placing monolithic compute units closer to a monolithic memory [6]. This concept known as near-memory computing has benefitted significantly from recent advances in die stacking technology[7] and the commercialization of advanced memory modules such as the Hybrid Memory Cube (HMC)[8] and High Bandwidth Memory (HBM)[9]. To achieve a denser and more fine-grained connectivity between memory and processing units, even 3-D monolithic integration has been proposed [10]. However, in all of these approaches that aim to reduce the time and distance to memory access, there still exists a physical separation between the memory and the compute units.

In-memory computing is an alternate approach where certain computational tasks are performed in place in the memory itself organized as a computational memory unit. As schematically illustrated in Figure 1, this is achieved by exploiting in tandem the physical attributes of the memory devices, their array-level organization, the peripheral circuitry as well as the control logic. Any computational task that is realized within the confines of a computational memory unit could be referred to as in-memory computing. However, the key distinction is that at no point during computation the memory content is read back and processed at the granularity of a single memory element. This latter scenario, where in addition the processing is performed in close proximity to the memory array, could instead be viewed as near-memory computing. Besides alleviating the costs in latency and energy associated with data movement, in-memory computing also has the potential to significantly improve the computational time complexity associated with certain computational tasks. This arises mostly from the massive parallelism afforded by a dense array of millions of memory devices performing computation. It is also likely that by introducing physical coupling between the memory devices, we can further reduce the computational time complexity [11]. By blurring the boundary between processing and memory units (an attribute that is also shared with the highly energy-efficient mammalian brain where memory and processing are deeply intertwined [12]), we gain significant improvements in computational efficiency. However, this is at the expense of the generality afforded by the conventional approach where memory and processing units are functionally distinct from each other. In this Review, we first give an overview of the memory devices that facilitate in-memory computing as well as the key in-memory computational primitives that are enabled. Subsequently, we present a range of applications that exploit these primitives. Finally, we present an outlook on the opportunities and challenges.

I. MEMORY DEVICES

Memory is at the heart of in-memory computing. One of the primary means to store information to date is through the presence or absence of charge such as in dynamic random access memory (DRAM), static random access memory (SRAM) and Flash memory[13]. There is also an emerging class of memory devices where information is stored in terms of differences in the atomic arrangements or orientation of ferromagnetic metal layers. Such differences manifest as a change of resistance and these devices are thus termed resistive memory devices[14]. Sometimes they are also referred to as memristive devices due to their relation to the circuit theoretic concept of memristive systems [15].

One of the primary characteristics of a memory device is the access time, i.e., how fast information can be stored (written) and retrieved (read). Another key characteristic is cycling endurance, which refers to the number of times a memory device can be switched from one state to the other. The memory devices in a computational memory unit are usually organized in a two-dimensional (2D) array with horizontal and vertical wires, typically referred to as the word line (WL) and the bit line (BL), used to access them. The memory array in a computational memory unit can be quite similar to that in a conventional memory unit but with certain differences in the read/write circuitry, the format of the input/output data as well as the control logic. For example, depending on the applications, multiple WLs need to be activated in parallel or analog output currents along BLs need to be sensed precisely.

A. Charge-based memory

An SRAM cell is a bi-stable transistor structure typically made of two CMOS inverters connected back to back, as shown in Figure 2**a**. The output potential of one inverter is applied as input to the other, forming a feedback loop that freezes the cell in a given logical state (0 or 1). Two additional field-effect transistors (FETs) serve as selectors, yielding a standard 6 transistor (6T) SRAM cell. SRAM is built entirely from FETs and has no dedicated storage element. However, one can view the charge as being confined within the barriers formed by the FET channels and the gate insulators. Due to the low FET barrier height (0.5eV), however, the charge constantly needs to be replenished from an external source and hence SRAM always needs to be connected to a power supply. A DRAM cell consists of a capacitor placed in series with a FET (Figure 2**b**). The charge is confined within the capacitor insulator, which forms a fixed-height barrier, and the FET. Since the maximum height of the FET barrier is limited by the band-gap of silicon ($\approx$ 1.1eV), the charge can be retained only for a fraction of a second and this necessitates periodic refresh. As shown in Figure 2**c**, in a Flash memory cell, the charge storage node is coupled to the gate of a FET with charge stored either on a conductive electrode surrounded by insulators (floating gate) or in discrete traps within a defective insulator layer (charge trapping layer). Unlike in DRAM, the barrier height of the storage node is sufficiently high for long-term data retention. However, the write operation requires high voltages (typically $> 10\,\mathrm{V}$) and entails significant latency ($> 10\,\mu\mathrm{s}$) due to the need to overcome the storage node barriers. Depending on how the Flash memory cells are organized, they are referred to as NOR or NAND Flash. In NOR Flash, every memory cell is connected to a BL, while in NAND Flash, several memory cells connected in series share a single connection to the BL. A Flash memory cell stores fewer electrons than DRAM and SRAM. Flash memory also has a substantially lower cycling endurance due to the gate oxide degradation under strong electric fields.

A range of in-memory logic and arithmetic operations can be performed using both SRAM and DRAM. Capacitive charge-redistribution serves as the foundation for many of them, in particular storing and sharing of charge across multiple storage nodes. In DRAM, simultaneous reading of devices along multiple rows can be used to execute basic Boolean functions within the memory array [16, 17]. Figure 2**d** shows a basic cell configuration that can be used to implement bit-wise `AND`/`OR` functions. Two memory cells, A and B, are used to store the operands. The logic state of the third cell, SEL, is set to 0 or 1 depending on whether an `AND` or an `OR` operation is realized, respectively. When all three cells are activated simultaneously, the bit-line voltage corresponds to the average voltage across the three capacitors. This voltage is sensed using a sense amplifier (SA) with a single decision threshold, which outputs the result of the logical operation. By using the negated output of the SA to also implement the `NOT` operation, a functionally complete set of Boolean functions is obtained. These bit-wise operations can be performed along the entire row of memory devices thus enabling parallel bulk bit-wise operations. Unlike DRAM, the SRAM cells do not contain a built-in capacitor and hence the parasitic BL capacitance is used instead to enable bulk in-memory logical operations [18, 19]. In Figure 2**e**, a basic construct for performing in-place bit-wise logical operations using SRAM is shown. Here, again, both of the WLs are activated simultaneously and by sensing the BL and $\overline{BL}$ with an SA, `AND` and `NOR` operations are performed, respectively. Besides realizing the logical primitives, it is also essential to efficiently cascade such operations. To perform cascadable logic operations using both DRAM and SRAM, additional cloning or duplication steps need to be enabled, allowing the construction of in-memory full adders and multipliers [17, 20]. The overhead of having to serially execute the cascaded operations is overcome by the ability to process several bit lines in parallel.

SRAM arrays can also be used for matrix-vector multiplication (MVM) operations, $Ax = b$, where A is the data matrix, x is the input vector, and b is the output vector [21–23]. If the elements of A and x are limited to signed binary values, the multiply operation is simplified to a combination of `XNOR` and `ADD` functions. Here, a 12T SRAM cell can be designed to execute `XNOR` operations within every memory cell [21]. In cases where x is non-binary, one approach is to employ capacitors in addition to the SRAM cells [22–24]. It was recently shown how 6-bit inputs can be multiplied with binary matrices stored in SRAM [22]. This involves a three-step process that is illustrated in Figure 2**f**. Note that the additional capacitors and switches could be shared among a group of SRAM cells at the expense of reduced parallelism and hence operational bandwidth. It is also possible to build the analog capacitor-based circuits in the vicinity of the SRAM array to accelerate MVM via near-memory computing [25, 26].

Flash memory can also be used to perform MVM operations[27, 28]. The gate voltage is modulated in accordance with a binary input vector (see Figure 2**g**). The matrix elements are stored as charge on the floating gate [28]. Because the devices can be accessed in parallel along a BL, NOR Flash has generally been preferred over NAND Flash for in-memory computing. However, there is recent work describing the use of 3D NAND, consisting of vertically stacked layers of serially connected FLASH devices, whereby each layer of the array encodes a unique matrix [29]. This approach could help to overcome the scalability issue of NOR Flash, which is difficult to scale beyond the 28 nm technology node.

B. Resistance-based memory

Memristive devices can be programmed to be in a low resistance state (LRS) or a high resistance state (HRS) through the application of electrical SET and RESET pulses, respectively. There is also the possibility to achieve intermediate resistance levels in certain types of memristive devices. The devices are typically organized in a 2D array and require a selection device in series with each device to prevent parasitic sneak path currents during writing and reading [30].

Resistive random access memory (RRAM) devices comprise metal-insulator-metal (MIM) stacks (see Figure 3**a**) and the resistive switching process typically involves the creation and disruption of conductive filaments (CF) comprising a localized concentration of defects. An LRS state corresponds to CFs bridging the two metal layers. Even though the history of RRAM can be traced back to at least the 1960s [31], key technological demonstrations in the 2000s[32–34] gave significant impetus to this technology. Phase change memory (PCM), which also dates back to the 1960s [35], is based on the property of certain types of materials, such as $Ge_2Sb_2Te_5$, to undergo a Joule heating-induced, rapid and reversible transition from a highly resistive amorphous phase to a highly conductive crystalline phase [36, 37]. As shown in Figure 3**b**, a typical PCM device has a mushroom shape where the bottom electrode confines heat and current. This results in a near-hemispherical shape of the amorphous region in the HRS state. By crystallizing the amorphous region, the LRS state is obtained. A relative newcomer to the resistive memory family, magnetoresistive random access memory (MRAM) consists of a magnetic tunnel junction (MTJ) structure with two ferromagnetic metal layers (pinned and free). These layers, for example made of the CoFeB alloy are separated by a thin tunnel oxide such as MgO (see Figure 3**c**). In the pinned layer, the magnetic polarization is structurally fixed to act as a reference, whereas in the free layer it is free to change during the write operation. Voltage pulses of opposite polarity are applied to switch the polarization of the free layer. Depending on whether the two ferromagnetic polarizations are parallel or antiparallel, the LRS and HRS states are obtained due to the tunnel magnetoresistive effect. Spin transfer torque MRAM (STT-MRAM) is currently the most promising MRAM technology [38, 39]. RRAM and PCM operate based on the rearrangement of atomic configurations and hence have worse access times (write speed) and cycling endurance than MRAM. However, they have substantially larger resistance windows that enable the storage of intermediate resistances even at an array level. RRAM has the advantage of using materials that are common in semiconductor manufacturing. However, in spite of the simplicity of the device concept, a comprehensive understanding of the switching mechanism is still lacking compared to PCM and MRAM.

One of the attributes of memristive devices that can be exploited for computation is their non-volatile binary storage capability. Logical operations are enabled through the interaction between the voltage and resistance state variables [40]. One particularly interesting characteristic of certain memristive logic families is statefulness, where the Boolean variable is represented solely in terms of the resistance states [41–43]. A schematic illustration of one such stateful memristive logic, MAGIC, that realizes the NOR logic operation is shown in Figure 3**d** [44]. Both the operands and the result are stored in terms of the resistance state variable. Stateful logic can be realized almost entirely in the memory array and has been demonstrated for RRAM[41] and STT-MRAM[45]. Stateful logic is also cascadable, whereby the output from one logical gate can directly feed into the input of a second logic gate [46, 47]. However, in stateful logic, the devices repeatedly get written into during the execution of the logical operations, which is a key drawback due to the associated energy cost and the limited cycling endurance of the devices. Hence, there is renewed interest in non-stateful logic such as the one shown in Figure 3**e**. Here, the logical operands are stored as resistance values, but the result of the logical operation is computed as a voltage signal [48, 49]. The operands stay fixed in the memory array and the devices need not be programmed during the evaluation of the logical operation. However, the sequential cascading of these logical operations requires additional circuits, typically located outside of the memory array. Memristive threshold logic is yet another non-stateful logic family where both the inputs and outputs are voltage signals and the logical functions are defined using the resistance values [50].

The non-volatile storage capability, in particular, the ability to store a continuum of conductance values, facilitates the key computational primitive of analog MVM [51–53]. The physical laws that are exploited to perform this operation are Ohm's law and Kirchhoff's current summation laws (see Fig. 3**f**). Memristive devices also exhibit an accumulative behavior [52, 54, 55], whereby the conductance of devices such as PCM and RRAM progressively increases or decreases with the successive application of appropriate programming pulses. This non-volatile accumulative behavior, in spite of its nonlinear and stochastic nature, can be exploited in several applications, such as training deep neural networks, where the conductance values need to be incrementally modified.

II. APPLICATIONS

The computational primitives reviewed in Section I have been applied to a wide range of application domains, ranging from scientific computing that requires high precision to stochastic computing that is enabled by imprecision and randomness. A high-level overview of the main applications that are being researched for in-memory computing is shown in Fig. 4. In-memory computing can be applied both to reduce the computational complexity of a problem as well as to reduce the amount of data being accessed by performing computations inside the memory arrays. The problems that could benefit the most from the complexity reduction are the NP-hard problems involving combinatorial optimization. Data-centric applications in machine learning and scientific computing benefit the most from reduced memory access. In this section, we review how in-memory computing has been applied to those applications and discuss the challenges involved with respect to the device properties presented previously.

A. Scientific computing

Linear algebra computational kernels, such as MVM, are common not only to machine learning but also to scientific computing applications. However, both memristive and charge-based memory devices suffer from significant inter-device variability and inhomogeneity across an array. Moreover, they exhibit intra-device variability and randomness that is intrinsic to how they operate. Hence, the precision of analog MVM operations with these devices is rather low. Although approximate solutions are sufficient for many computational tasks in the domain of AI, building an in-memory computing unit that can effectively address scientific computing and data analytics problems – which typically require high numerical accuracy – remains challenging.

The aforementioned accuracy limitation can, to a certain extent, be remedied by an old technique in computer architecture called “bit-slicing”. Bit-slicing is a general approach for constructing a processor from modules of smaller bit width. Each of the modules processes one bit-field or “slice” of an operand [56]. The grouped processing components will then have the capability to process, in parallel, an arbitrarily chosen full word-length of a particular task. This concept has been proposed for increasing the accuracy of the in-place MVM based on in-memory computing (see Fig. 5**a**) [57–60]. According to this technique, an n-bit element of the matrix is mapped onto device conductance values of n binary crossbar arrays, i.e., n bit slices. Thus, each bit slice contains the binary values of the matrix elements in a particular bit position. Similarly, bit-slicing can also be applied to the input vector elements, where each bit slice is input to the crossbar arrays one at a time. To perform an in-place MVM, a vector bit slice is multiplied with a matrix bit slice, with $O(1)$ time complexity, and the partial products of these operations are combined outside of the crossbar arrays through a shift-and-add reduction network [57]. Note that the bit slices can also be implemented on the same crossbar array in a column-by-column manner. In this case, columns at a distance n from each other represent a single bit slice. Although the above concept has been described based on bit slices, i.e., binary memristive arrays, it can easily be generalized to multi-level memristive devices. The bit slice approach applied to a 16-bit input vector sliced into 16 1-bit slices for increasing numerical precision has been demonstrated experimentally where a numerical differential equation solver using a small Ta_2O_{5-x} RRAM 16×3 crossbar array was successfully implemented [61].

Although the bit-slice technique appears to address the limitations surrounding the precision of analog MVM operations, there are still inaccuracies arising from the analog summation along columns, which potentially could be more detrimental in larger crossbar arrays. Moreover, the extra peripheral circuitry of the shift-and-add external reduction networks could substantially increase the energy consumption and area. Mixed-precision computing is an alternate approach to achieve high precision processing based on in-memory computing. This approach is based on the well-established iterative refinement technique for improving a computed solution to a system of linear equations [62]. Through this technique, the time complexity of iterative linear solvers can be reduced by combining low-precision with high-precision arithmetic [63]. The adaptation of this concept for in-memory computing and experimental demonstration of solving a system of 5,000 linear equations using 998,752 PCM devices with arbitrarily high accuracy was presented in [64]. Here, the idea is to use fast but imprecise MVM, via in-memory computing in an iterative linear solver, to obtain an approximate solution, and then refine this solution based on the residual error calculated precisely through digital computing (Fig. 5**b**). The main limitation of this technique is that the data need to be stored both in crossbar arrays as well as in the memory of a high-precision digital processing unit, which increases the resources needed to solve the problem. Moreover, the achievable speedup comes from reducing the number of iterations needed to solve the problem, resulting in an overall computational complexity of $O(N^2)$ for a $N \times N$ matrix, i.e., still proportional to the problem size.

Several extensions to these two techniques are imaginable to further improve the performance benefits and reliability. One way to potentially speed up linear solvers further is to realize a one-step linear solver in the analog domain [65], which has been demonstrated using a 3×3 RRAM crossbar array [66]. This approach is based on an old idea of analog matrix inversion [67], whereby a known vector, forced as currents on the columns of the crossbar, establishes an output voltage vector at the rows, which is equal to the product of the inverse of the conductance matrix multiplied by the vector of currents. Although the high parallelism provided by this approach is promising, its implementation is hardwired and therefore not scalable, and requires very precise conductance tuning and high linearity of current/voltage characteristics. There are also initial results on error correction schemes [68] as well as extensions to the bit-slicing concept for achieving floating-point accuracy [69] on memristive crossbar arrays. These research avenues could enlarge the application space of in-memory computing to encompass applications in scientific computing where high computational accuracy is required.

B. Signal processing, optimization, and machine learning

There are several applications in the domain of signal processing, optimization, and machine learning where approximate solutions can be considered acceptable, and the bulk of the computation could thus be performed with in-memory computing. The crossbar-based analog MVM can be used in many applications such as image compression, compressed sensing, combinatorial optimization, sparse coding, principal component analysis, associative memories, and reservoir computing.

The application of in-memory computing to analog image compression has been studied experimentally in [70]. The idea is to encode a transform matrix, for example, a discrete cosine transform, as the conductance values of devices organized in a crossbar array. The image pixel intensities, represented as voltages, are applied to the crossbar first row by row and, in a second step, column by column. The compression is then performed by keeping only a certain ratio of the highest coefficients of the transformed image and discarding the rest. Compression experiments using a 128×64 crossbar array of hafnium oxide (HfO_2) devices yielded reasonably well-reconstructed images, although with a few visible artifacts due to device non-idealities [70]. The transform coding described above for sparsifying large signals is fundamental to common compression schemes such as JPEG or MPEG, but can also be used for compressed sensing. The basic idea of compressed sensing is to acquire a few (M) sampling measurements from a high-dimensional signal of size N, and to subsequently recover that signal accurately. Compressed sensing can be realized via in-memory computing by encoding the $M \times N$ measurement matrix used for this process, which typically contains randomly distributed elements, in a crossbar array of memory devices [65, 71]. This array can be used to perform the MVM operations associated with both the compression and recovery tasks. The efficacy of this scheme has been experimentally demonstrated through 128×128 image compression and reconstruction tasks using more than 256,000 PCM devices [71]. However, here as well, device non-idealities such as conductance noise were found to reduce the reconstruction accuracy.

In the field of optimization, a promising application of in-memory computing is for combinatorial optimization problems, such as the traveling salesman problem, Boolean satisfiability, and integer linear programming. Combinatorial optimization is the process of searching for maxima or minima of an objective function whose domain is a discrete but large configuration space. To address these computationally intensive typically NP-hard problems, simulated annealing inspired approaches, such as the massively parallel Boltzmann machines and Hopfield networks, have been proposed. The basic idea is to compute the inner products, the fundamental building blocks in Boltzmann machines [57] or Hopfield networks [72], in place via in-memory computing. For solving the problem, the network is run until convergence, i.e., the energy is minimized, which involves updating only the state variables, while the weights implemented in the crossbar array remain constant. An interesting prospect is to utilize the device noise as an explicit source of noise to force the network to continuously explore the solution space, which is necessary to achieve proper convergence [72, 73]. However, it is required to precisely control this noise via an annealing schedule, which is challenging to implement. Another intriguing approach, going beyond simply accelerating the inner products in recurrent networks, is to use a network of coupled nonlinear analog oscillators whose dynamics execute an efficient search for solutions of combinatorial optimization problems [73]. Volatile memristive devices based on Mott insulator-metal transition materials, such as VO_2[74] and NbO_2[75], as well as spintronic oscillators based on MTJs [76] can be used to realize compact nanoscale oscillators that facilitate this form of computing.

Several memory-centric problems in machine learning could also benefit from in-memory computing. One is sparse dictionary learning, a learning framework in which a sparse representation of input data is obtained in the form of a linear combination of basic elements, which form the so-called dictionary of features. As opposed to the transform coding approach described earlier, both the dictionary and the sparse representation are learned from the input data. If the learned dictionary is mapped onto device conductance values in a crossbar array, it is possible to obtain the sparse representation using the iterative-shrinking threshold [77] or locally competitive algorithms [78]. The matrix-vector and the transpose-matrix-vector multiplications associated with the algorithms are performed in the crossbar. Dictionary learning requires updating the conductance values by exploiting the accumulative behavior of the memristive devices, based on, for example, stochastic gradient descent [77, 79], which is challenging due to device stochasticity and nonlinear conductance change with the number of applied pulses [79]. Another application is principal component analysis, a dimensionality reduction technique to reveal the internal structure of data by using a limited number of principal components. It is usually achieved by finding the eigenvectors of the data covariance matrix. This can be realized using the "power iteration" method in which the MVM operations can be performed using in-memory computing [65]. An alternative approach is to use a linear feedforward neural network in which the weights are implemented in a crossbar array. The network is optimized via unsupervised learning using Sanger's rule to obtain the principal components, given by the weights connected to each output neuron representing the classes in which the data is clustered [80].

Another relevant application for in-memory computing, which is used in several machine learning algorithms, is associative memory. An associative memory compares input search data with the data stored in it and finds the address of the data with the closest match to the input data [81]. This capability is used in several learning frameworks, such as brain-inspired hyperdimensional computing [82, 83] and memory-augmented neural networks [84, 85]. One way to realize associative memory is to use a Hopfield network, which can be trained to minimize the energy of the states that it should remember. This has been successfully demonstrated on small arrays of PCM [86] and RRAM [87] devices. Another more straightforward way to realize associative memory is simply to encode the stored data directly in a crossbar array and compute, in parallel, the Hamming distances of each stored data vector with the input search data vector via in-memory dot-products [88].

Finally, the collective dynamics of an ensemble of dynamical systems could be exploited to perform certain machine learning tasks. One prominent example of this is reservoir computing (RC). The essential idea of reservoir computing is to map inputs into a high-dimensional space such that it is possible to classify the input patterns with a simple linear classifier. One of the approaches to implement RC is to feed the input into a fixed physically realized dynamical system. Memristive devices could play a key role in these types of physical RC. For example, Du et al. proposed the use of a collection of memristive devices with short-term temporal dynamics to serve as the physical reservoir and to classify temporal signals [89]. Sebastian et al. used a reservoir of a million PCM devices and exploited their accumulative behavior to classify binary random processes into correlated and uncorrelated classes [90].

C. Deep learning

Recently, deep artificial neural networks, loosely inspired by biological neural networks, have shown a remarkable human-like performance in tasks such as image processing and voice recognition [91]. A deep neural network (DNN) consists of at least two layers of nonlinear neuron units interconnected by adjustable synaptic weights. Modern DNNs can have over 1000 layers [92]. By tuning the adjustable weights, for instance, optimizing them by using millions of labeled examples, these networks can solve certain problems remarkably well. Dedicated mixed-signal chips that could implement multi-layer networks were already developed in the early 1990s but were eventually abandoned in favor of field-programmable gate arrays (FPGAs) and general-purpose graphics processing units (GPGPUs), partly due to lack of flexibility [93]. While high-performance GPGPUs are incontestably the hardware that has been primarily responsible for the recent success of deep learning, mixed-signal architectures based on in-memory computing are being actively researched, targeting mostly edge computing applications where high energy efficiency is critical.

A DNN can be mapped onto multiple crossbar arrays of memory devices that communicate with each other as illustrated in Fig. 6**a**. A layer of the DNN can be implemented on (at least) one crossbar, in which the weights of that layer are stored in the charge or conductance state of the memory devices at the crosspoints. The propagation of data through that layer is performed in a single step by inputting the data to the crossbar rows and deciphering the results at the columns. The results are then passed through the neuron nonlinear function and input to the next layer. The neuron nonlinear function is typically implemented at the crossbar periphery, using analog or digital circuits. Because every layer of the network is stored physically on different arrays, each array needs to communicate at least with the array(s) storing the next layer for feed-forward networks, such as multi-layer perceptrons (MLPs) or convolutional neural networks (CNNs). For recurrent neural networks (RNNs), the output of an array needs to communicate with its input. Array-to-array communication can be realized using a flexible on-chip network, akin to those used in digital DNN accelerators [94]. However, their efficient adaptation to in-memory computing based architectures is still being explored[95].

The efficient MVM realized via in-memory computing is very attractive for inference-only applications, where data is propagated through the network on offline-trained weights. With respect to specialized inference accelerators operating at reduced digital precision (4 to 8-bit), such as Google's tensor processing unit [4] and low-power GPGPUs such as NVIDIA T4 [96], in-memory computing aims to improve the energy efficiency even further by eliminating the separation between memory and processing for the MVM operations. Implementations using SRAM-based in-memory computing has focused on binary weight networks, in which weights are represented by a single bit [97]. Various implementations, such as current-based [21] and charge-based [22, 23] computational circuits, have been proposed and were able to demonstrate 1-bit arithmetic energy efficiencies of >100 TOPS/W for MVM. Chips using in-memory computing on non-volatile memory devices have also been fabricated using NOR-Flash [28] and RRAM [98–100]. Using non-volatile memory ensures that the weights will be retained when the power supply is turned off, unlike with SRAM. Also, the multi-level storage capability of these devices can be exploited to implement non-binary networks, which yield higher accuracy and are easier to train than binary weight networks. Usually, at least two devices per weight are used in a differential configuration to implement positive and negative weights [101]. Multiple binary/multi-level devices using the bit-slicing technique can be used to further increase the precision [58, 59, 98, 102]. The state-of-the-art experimental demonstrations of DNN inference based on in-memory computing have reported a competitive energy efficiency of ≥ 10 TOPS/W for reduced-precision MVM (see Table I). Nonetheless, for all these implementations, custom training [103–105] and/or on-chip retraining [25, 100] of the network is needed to mitigate the effect of defects, and device and circuit level non-ideality on the network accuracy. The training procedure should be generic and as agnostic as possible to the hardware such that the network would have to be trained only once to be deployed on a multitude of different chips. Another important research topic is the design of efficient intra- and inter-layer pipelines [58] to ensure that all the arrays on the chip are always active during inference, together with flexible array-to-array communication and control. It is especially important for CNNs, in which a large image is passed through small kernels at only a few pixels at a time [106], leading to prohibitive latencies and buffer requirements if no pipelining is used.

In-memory computing can also be used in the context of supervised training of DNNs with backpropagation. This training involves three stages: forward propagation of labeled data through the network, backward propagation of the error gradients from output to the input of the network, and weight update based on the computed gradients with respect to the weights of each layer. This procedure is repeated over a large dataset of labeled examples for multiple epochs until satisfactory performance is reached by the network. This makes the training of state-of-the-art networks very time and energy-consuming even with high-performance GPGPUs. When performing training of a neural network encoded in crossbar arrays, forward propagation is performed in the same way as for the inference described above. The only difference is that all the activations x_i of each layer have to be stored locally in the periphery. Next, backward propagation is performed by inputting the error gradient δ_j from the subsequent layer onto the columns of the current layer and deciphering the result from the rows. The resulting weighted sum $\sum_i \delta_j W_{ij}$ needs to be multiplied by the derivative of the neuron nonlinear function, which is computed externally, to obtain the error gradient of the current layer. Finally, the weight update is performed based on the outer product of activations and error gradients $x_i\delta_j$ of each layer. One approach is to perform a parallel weight update by sending deterministic or stochastic overlapping pulses from the rows and columns simultaneously to implement an approximate outer product and program the devices at the same time (see Fig. 6**b**) [107–111]. While this parallelism may be efficient in terms of speed, each outer product needs to be applied to the arrays one at a time (either after every training example or one by one after a batch of examples), leading to a large number of pulses applied to the devices. This results in stringent requirements on the device granularity, asymmetry, and linearity to obtain accurate training [109, 112], and high device endurance is critical. Using multiple devices per synapse with a periodic carry can relax some of the device requirements, at the price of a costly reprogramming of the entire array every time the carry is performed [110, 111]. Another approach is a mixed analog/digital weight update whereby ΔW_{ij} is computed digitally and applied to the arrays row-by-row or column-by-column (Fig. 6c). ΔW_{ij} can be applied either at every individual training example (online training) or batch of training examples (by accumulating all the updates within one batch in a digital memory) [113–115]. ΔW_{ij} can also be accumulated in a digital memory across batches and specific devices are programmed when their corresponding accumulated values reach a threshold [116]. This approach is more flexible than the parallel weight update based on overlapping pulses because it can implement any learning rule, not only stochastic gradient descent, and the digital computation and accumulation of weight updates significantly relax the requirements on the device granularity and endurance [116]. However, the cost is the need for additional digital computing and memory hardware. The training approaches presented here are still at the stage of functionality demonstration and need to overcome the device-related challenges before they could be employed on edge devices in applications where online learning is desirable.

A third application domain for in-memory computing in deep learning is spiking neural networks (SNNs). The main difference between SNNs and the non-spiking neural networks discussed so far is that SNN neurons compute with asynchronous spikes that are temporally precise, as opposed to continuous-valued activations that operate on a common clock cycle. Hence, SNNs are ideally suited for processing spatio-temporal event-driven information from neuromorphic sensors. There has been significant progress in recent years in designing deep SNNs trained with supervised learning that can perform close to conventional DNNs [117]. The main approaches rely either on converting weights from a previously trained non-spiking DNN [118, 119], or implementing backpropagation training using spike signals on the SNN itself [120, 121]. Recently it has been shown that a spiking neuron can be transformed into a recurrent neural network unit, and thus it is possible to apply the existing deep learning frameworks for seamless training of any SNN architecture with backpropagation through time [122]. However, most of the efforts in applying in-memory computing to SNNs have focused on unsupervised learning with local learning rules. The best-known example for this is spike-timing-dependent plasticity (STDP), which adjusts a synaptic weight based on the relative timing between its output and input neuron spikes. In-memory implementations of SNNs have traditionally been done using slow subthreshold analog CMOS circuits that directly emulate the functions of neurons and synapses, together with fast event-driven digital communication [12, 123]. Support for STDP learning was also successfully implemented [124]. Non-volatile nanoscale devices, such as PCM [125–128] and RRAM [129, 130], have been proposed to be integrated as part of the synapse and neuron circuits in a hardware SNN. Support for STDP learning with these devices has been generally implemented using rather complex schemes based on overlapping pulses. However, STDP-based learning rules have still not been able to reach the accuracy of conventional DNNs trained with backpropagation, despite significant recent progress [131]. Although SNNs are believed to be computationally more powerful than conventional DNNs because of the added temporal dimension, an application where this advantage is clearly demonstrated and exploited is still lacking. This is one of the reasons why generally SNNs have not been as widely adopted as conventional DNNs. However, with the incorporation of additional bio-inspired neuronal and synaptic dynamics[132], SNNs could transcend conventional deep learning in certain application domains and memristive devices could be exploited to natively implement such dynamics [133].

D. Stochastic computing and security

The stochasticity associated with the switching behavior in memristive devices can also be exploited for in-memory computing [134]. In an MRAM, the MTJ switching is inherently stochastic due to the thermal fluctuations affecting the free layer and the write voltage and duration can be used to tune the switching probability. In RRAM, if the write voltage is comparable to V_{SET}, then the SET transition takes place after a certain time delay. This delay time exhibits significant cycle to cycle statistical variations [135]. This behavior is also observed in PCM devices and is attributed to the threshold switching dynamics as well as the variability associated with the HRS states [136, 137]. In both RRAM and PCM, the dependence of the delay time on the write voltage provides us a means to tune its distribution. PCM exhibits additional stochasticity associated with crystallization time. It is attributed to the small variations in the atomic configurations of the amorphous volume created upon the preceding RESET. This results in variability associated with the number of pulses that are needed to fully crystallize the amorphous volume [137].

Random number generation is important for a variety of areas, such as stochastic computing, data encryption, machine learning, and deep learning [138, 139]. Therefore, there is a significant interest in employing memristive devices as an entropy source for a compact and efficient true random number generator (TRNG). As opposed to a pseudo-random number generator (PRNG), a TRNG does not require a seed and uses the entropy arising from physical phenomena such as Johnson-Nyquist noise, time-dependent dielectric breakdown or ring oscillator jitter [140]. The stochastically switching memristive device in conjunction with a simple circuitry, comprising a comparator and some digital logic, can be used to realize a TRNG (see Figure 7**a**) [141]. Several variants of this idea have been explored using RRAM [142, 143], PCM [137] and STT-MRAM [144, 145].

The stochastic number streams generated by memristive TRNG blocks have also been employed to realize efficient multiply units [142]. For example, a multiply operation between two numbers between 0 and 1 can be efficiently realized by performing an AND operation between binary random bitstreams representing those numbers [138]. Another interesting application is that of performing probabilistic inference using Bayes's rule (see Figure 7**b**). For example, the required probability distributions can be generated as random bitstreams using a stochastically switching MRAM device [146]. The stochasticity associated with memristive devices has also found applications in spiking neural networks where stochastically firing neurons [147, 148] (see Figure 7**c**) and stochastic binary synapses [149] have been proposed.

Another promising application is in the domain of security. A physically unclonable function is a physical system that statistically maps an input digital word to an output one through a secret key depending on an intrinsically stochastic property of the chip. Typically, silicon process variations or the inherent physical variability of device parameters are exploited. PUF can be viewed as a computational unit that returns an output response, $r = f(c)$, for each input challenge, c. f describes the unique internal physical characteristics of the PUF. A specific PUF instance is defined by a set of possible challenge-response pairs (CRPs). SRAM devices are commonly used to implement PUF circuitry by exploiting the metastable states of cross-coupled inverters [150]. However, memristive devices organized in a crossbar array can be exploited to design a much stronger PUF with a significantly larger CRP set (see Figure 7**d**). The key idea is to exploit the broad distribution of memristive resistance values as well as the exponential number of available current sneak paths [151–153].

III. OPPORTUNITIES, CHALLENGES AND PERSPECTIVE

There are different attributes in the applications discussed in Section II that can be leveraged through in-memory computing in order to increase the overall system performance. To take advantage of in-memory computing for MVM, it is preferable for the application to perform many MVMs on large squarish and dense matrices that stay constant throughout its execution. In this way, only smaller vector data have to be moved in and out of the crossbar arrays. This effectively reduces the overall data movement by eliminating frequent accesses to the matrix data. Applications that fall into this category include deep learning inference, dense iterative linear solvers, compressed sensing, sparse coding, and associative memories. Although there has been some work on leveraging sparse MVM through in-memory computing[69] as well, more research is needed to efficiently orchestrate the allocation of the partial vector components across different arrays and maximize the areal efficiency in coding sparse matrices on crossbars. The inherent parallelism offered by analog computations can also potentially reduce the computational complexity of a problem. For instance, NP-hard problems involving combinatorial optimization can benefit from analog acceleration of MVMs or using networks of chaotic and nonlinear memristive elements to accelerate the solution search. For applications in stochastic computing, in which memristive devices are not employed to reduce data accesses, the overall benefits can be expected only from the memristive TRNG acceleration over a conventional implementation. For the logic primitives, performance benefits come from avoiding moving data to a processor to perform the logic operations. However, efficiently cascading the logic primitives to perform more complex logic operations, such as a full adder [47, 154, 155] or fixed-point multiplier [156], is critical in achieving end-to-end benefits in applications. Candidate applications in which in-memory logic could be leveraged include database query and encryption of data [157], object detection and evaluation of Fast Fourier Transforms [50], and image processing kernels [156].

Computing with charge-based computing devices is attractive due to their technological maturity, even though SRAM has a relatively large areal footprint even at advanced technology nodes and DRAM and Flash memory face severe scaling challenges. Charge-based analog computation is inherently subject to thermal noise, which sets an upper limit to the precision achievable for a given capacitor size and ambient temperature. Additionally, the manufacturing process introduces non-idealities in the form of capacitor size variations, thus limiting the maximum achievable accuracy. Memristive devices, on the other hand, could potentially be scaled to dimensions of a few nanometers [158–161]. The key challenges for memristive devices are write variability and conductance variations. Write variability captures the inaccuracies associated with writing an array of devices to desired conductance values. In RRAM, the physical origin of this variability lies mostly in the stochastic nature of filamentary switching and one prominent approach to counter this is that of establishing preferential paths for CF formation [162, 163]. Representing single computational elements by using multiple memory devices could also mitigate variability issues [164]. Conductance variations refer to the temporal and temperature-induced variations of the programmed conductance values. One prominent example is “drift” in PCM devices, which is attributed to the intrinsic structural relaxation of the amorphous phase. A promising approach towards addressing “drift” is that of projected phase change memory, which comprises a non-insulating material segment parallel to the phase change material segment [165, 166].

There are also several challenges to be tackled at the peripheral circuit-level for in-memory computing. A critical issue is the need for digital-to-analog (analog-to-digital) conversion every time data goes in to (out of) the crossbar arrays. There are solutions that employ fully analog peripheral circuits to avoid such conversions [28, 111], at the cost of less flexibility and accuracy. Usually, the preferred method for inputting digital data to memristive crossbars is pulse-width modulation, because the result of the computation based on Ohm’s law will not be affected by the nonlinearity of the current/voltage characteristics of the devices. For digitizing the crossbar output, most works have employed analog-to-digital converters (ADCs) [21, 22] or sense amplifiers [98]. The precision of the digitization needs to be sufficient to properly resolve the analog multiply-accumulate operations, and a precision of at least four bits (including sign) has so far been necessary for DNN inference applications [21, 22, 98]. Because of their large area and power consumption, it is typically required to multiplex ADCs across multiple columns, which increases the latency. Moreover, it is critical to properly scale the input and output ranges, such that the crossbar output falls within the limited dynamic range of the ADC; otherwise there would be a prohibitive loss of computational precision. Another important challenge is the finite resistance of the crossbar wires. It can lead to parasitic voltage drops on the devices during readout when a high current is flowing through them (IR drop), creating errors in the analog computation results. This not only limits the maximum crossbar size that can be reliably operated, but also the integration density because of the difficulty to use the metal layers close to the CMOS front-end due to their higher resistivity. From an architectural point of view, a computational memory unit could have multiple in-memory computing cores connected through an on-chip network [95]. Besides the memory arrays and associated peripheral circuitry, each in-memory compute core could also have some rudimentary digital processing units as well as conventional memory such as registers and buffers. There is significant on-going research on defining such hierarchical organizations of in-memory computing cores to tackle a range of applications [58, 167, 168]. Another crucial aspect is the design of a software stack that extends from the user-level application to the low-level driver that directly controls the computational memory unit. The software stack is responsible for transparently compiling, for example a machine learning model, into optimized operations and routing, and orchestrating data movement to and from the unit. Recent works have started to explore some of these aspects for specific DNN inference workloads [168, 169].

The specific requirements that the devices need to fulfill when employed for computational memory are likely to be different from those needed for conventional memory and will also be highly application dependent. One requirement for memristive devices, which is common to most computing applications, is that the low-resistance state should be resistive enough to limit the impact of the IR drop during writing and readout of the array. For memristive stateful logic, the requirements include an abrupt, fast and low-power threshold switching characteristic [170], high cycling endurance ($> 10^{12}$ cycles) as well as low device-to-device variability of switching voltages and LRS/HRS values. For computational tasks involving read-only operations, such as MVM, endurance is much less critical as long as the conductance states remain unchanged during their execution. However, a gradual analog-type switching characteristic is desirable for programming a continuum of conductance values in a single device, and temporal conductance variations, device failures and variability can severely affect the performance [171]. Gradual, linear, and symmetric conductance changes are also desired in applications where the device conductance needs to be incrementally modified such as neural network training [112]. For stochastic computing applications, random device variability is not an issue, but graceful device degradation is [137]. Moreover, very fast and low-power switching devices with high endurance are necessary for being competitive with efficient CMOS-based implementations [140].

Besides the conventional memory devices presented in this article, several new memory concepts are being proposed for in-memory computing [172–174]. Even though promising, it is difficult to fully assess their benefits in the absence of large scale experimental demonstrations and/or integration with CMOS technology. Ferroelectric devices, such as ferroelectric random access memory [175], ferroelectric field effect transistors [176] and ferroelectric tunnel junctions [177], have also been explored for in-memory computing and the newly discovered ferroelectricity in hafnium oxide has given significant impetus to this research. There is also a recent interest in photonic memory devices [178, 179], where data can be written, erased, and read optically. Such devices are being explored for all-photonic chip-scale information processing. For example, by integrating phase-change materials onto an integrated photonics chip, the analog multiplication of an incoming optical signal by a scalar value, encoded in the state of the phase change material, was performed [180]. One of the primary advantages of the optical approach is the potential for inherent wavelength division multiplexing.

The explosive growth of AI, in particular deep neural networks, has created a market for high performance and efficient inference and training chips, both in the cloud and on the edge. Moreover, mobile devices, which are particularly hampered by energy constraints, are playing an increasingly important role in defining the future of computing. Yet another reason is that the cost per transistor is plateauing even though transistor sizes continue to get smaller (albeit not at the rate envisaged by Gordon Moore anymore). This could prompt many chip manufacturers to sustain older technology nodes but instead equip the chips with high performance computing engines such as computational memory. Note that most of the memristive device technologies are amenable to back end of line (BEOL) integration, thus enabling their integration with a wide range of front end CMOS technologies. To conclude, in-memory computing, using both charge-based as well as resistance-based memory devices, is poised to have a significant impact on improving the energy/area efficiency as well as the latency compared to conventional computing systems and given the conducive market environment, this could usher in a new era of non-von Neumann computing.

[1] Mutlu, O., Ghose, S., Gómez-Luna, J. & Ausavarungnirun, R. Processing data where it makes sense: Enabling in-memory computation. *Microprocessors and Microsystems* **67**, 28–41 (2019).

[2] Horowitz, M. Computing's energy problem (and what we can do about it). In *Proceedings of the International Solid-state Circuits Conference (ISSCC)*, 10–14 (IEEE, 2014).

[3] Keckler, S. W., Dally, W. J., Khailany, B., Garland, M. & Glasco, D. GPUs and the future of parallel computing. *IEEE Micro* **31**, 7–17 (2011).

[4] Jouppi, N. P. *et al.* In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 1–12 (IEEE, 2017).

[5] Sze, V., Chen, Y.-H., Yang, T.-J. & Emer, J. S. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE* **105**, 2295–2329 (2017).

[6] Patterson, D. *et al.* A case for intelligent RAM. *IEEE micro* **17**, 34–44 (1997).

[7] Farooq, M. *et al.* 3D copper TSV integration, testing and reliability. In *Proceedings of the International Electron Devices Meeting*, 7–1 (IEEE, 2011).

[8] Pawlowski, J. T. Hybrid memory cube (HMC). In *Proceedings of the Hot Chips Symposium (HCS)*, 1–24 (IEEE, 2011).

[9] Kim, J. & Kim, Y. HBM: Memory solution for bandwidth-hungry processors. In *Proceedings of the Hot Chips Symposium (HCS)*, 1–24 (IEEE, 2014).

[10] Shulaker, M. M. *et al.* Three-dimensional integration of nanotechnologies for computing and data storage on a single chip. *Nature* **547**, 74 (2017).

[11] Di Ventra, M. & Pershin, Y. V. The parallel approach. *Nature Physics* **9**, 200 (2013).

[12] Indiveri, G. & Liu, S.-C. Memory and information processing in neuromorphic systems. *Proceedings of the IEEE* **103**, 1379–1397 (2015).

[13] Zhirnov, V. V. & Marinella, M. J. *Emerging Nanoelectronic Devices*, chap. Memory Technologies: Status and Perspectives (Wiley Online Library, 2015).

[14] Wong, H.-S. P. & Salahuddin, S. Memory leads the way to better computing. *Nature Nanotechnology* **10**, 191 (2015).

[15] Chua, L. Resistance switching memories are memristors. *Applied Physics A* **102**, 765–783 (2011).

[16] Li, S. *et al.* DRISA: A DRAM-based reconfigurable in-situ accelerator. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 288–301 (2017).

[17] Seshadri, V. *et al.* Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology. In *Proceedings of International Symposium on Microarchitecture*, 273–287 (New York, NY, USA, 2017).

[18] Jeloka, S., Akesh, N. B., Sylvester, D. & Blaauw, D. A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6T bit cell enabling logic-in-memory. *IEEE Journal of Solid-State Circuits* **51**, 1009–1021 (2016).

[19] Aga, S. *et al.* Compute caches. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 481–492 (IEEE, 2017).

[20] Wang, J. *et al.* A compute SRAM with bit-serial integer/floating-point operations for programmable in-memory vector acceleration. In *Proceedings of the International Solid- State Circuits Conference (ISSCC)*, 224–226 (2019).

[21] Jiang, Z., Yin, S., Seok, M. & Seo, J. XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks. In *Proceedings of the Symposium on VLSI Technology*, 173–174 (2018).

[22] Biswas, A. & Chandrakasan, A. P. CONV-SRAM: an energy-efficient SRAM with in-memory dot-product computation for low-power convolutional neural networks. *IEEE Journal of Solid-State Circuits* **54**, 217–230 (2019).

[23] Valavi, H., Ramadge, P. J., Nestler, E. & Verma, N. A 64-tile 2.4-Mb in-memory-computing CNN accelerator employing charge-domain compute. *IEEE Journal of Solid-State Circuits* **54**, 1789–1799 (2019).

[24] Verma, N. *et al.* In-memory computing: Advances and prospects. *IEEE Solid-State Circuits Magazine* **11**, 43–55 (2019).

[25] Gonugondla, S. K., Kang, M. & Shanbhag, N. R. A variation-tolerant in-memory machine learning classifier via on-chip training. *IEEE Journal of Solid-State Circuits* **53**, 3163–3173 (2018).

[26] Bankman, D., Yang, L., Moons, B., Verhelst, M. & Murmann, B. An always-on 3.8 μ J/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28-nm CMOS. *IEEE Journal of Solid-State Circuits* **54**, 158–172 (2019).

[27] Diorio, C., Hasler, P., Minch, A. & Mead, C. A. A single-transistor silicon synapse. *IEEE Transactions on Electron Devices* **43**,

1972–1980 (1996).

[28] Merrikh-Bayat, F. *et al.* High-performance mixed-signal neurocomputing with nanoscale floating-gate memory cell arrays. *IEEE Transactions on Neural Networks and Learning Systems* **29**, 4782–4790 (2018).

[29] Wang, P. *et al.* Three-dimensional NAND flash for vector-matrix multiplication. *IEEE Transactions on Very Large Scale Integration Systems* **27**, 988–991 (2019).

[30] Burr, G. W. *et al.* Access devices for 3D crosspoint memory. *Journal of Vacuum Science & Technology B, Nanotechnology and Microelectronics: Materials, Processing, Measurement, and Phenomena* **32**, 040802 (2014).

[31] Hickmott, T. Low-frequency negative resistance in thin anodic oxide films. *Journal of Applied Physics* **33**, 2669–2682 (1962).

[32] Beck, A., Bednorz, J., Gerber, C., Rossel, C. & Widmer, D. Reproducible switching effect in thin oxide films for memory applications. *Applied Physics Letters* **77**, 139–141 (2000).

[33] Waser, R. & Aono, M. Nanoionics-based resistive switching memories. *Nature Materials* **6**, 833 (2007).

[34] Strukov, D. B., Snider, G. S., Stewart, D. R. & Williams, R. S. The missing memristor found. *Nature* **453**, 80 (2008).

[35] Ovshinsky, S. R. Reversible electrical switching phenomena in disordered structures. *Physical Review Letters* **21**, 1450 (1968).

[36] Wong, H.-S. P. *et al.* Phase change memory. *Proceedings of the IEEE* **98**, 2201–2227 (2010).

[37] Burr, G. W. *et al.* Recent progress in phase-change memory technology. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* **6**, 146–162 (2016).

[38] Khvalkovskiy, A. *et al.* Basic principles of STT-MRAM cell operation in memory arrays. *Journal of Physics D: Applied Physics* **46**, 074001 (2013).

[39] Kent, A. D. & Worledge, D. C. A new spin on magnetic memories. *Nature Nanotechnology* **10**, 187 (2015).

[40] Vourkas, I. & Sirakoulis, G. C. Emerging memristor-based logic circuit design approaches: A review. *IEEE Circuits and Systems Magazine* **16**, 15–30 (2016).

[41] Borghetti, J. *et al.* Memristive switches enable stateful logic operations via material implication. *Nature* **464**, 873 (2010).

[42] Linn, E., Rosezin, R., Tappertzhofen, S., Böttger, U. & Waser, R. Beyond von neumann-logic operations in passive crossbar arrays alongside memory operations. *Nanotechnology* **23**, 305205 (2012).

[43] Jeong, D. S., Kim, K. M., Kim, S., Choi, B. J. & Hwang, C. S. Memristors for energy-efficient new computing paradigms. *Advanced Electronic Materials* **2**, 1600090 (2016).

[44] Kvatinsky, S. *et al.* MAGIC-memristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs* **61**, 895–899 (2014).

[45] Mahmoudi, H., Windbacher, T., Sverdlov, V. & Selberherr, S. Implication logic gates using spin-transfer-torque-operated magnetic tunnel junctions for intrinsic logic-in-memory. *Solid-State Electronics* **84**, 191–197 (2013).

[46] Kim, K. M. *et al.* Single-cell stateful logic using a dual-bit memristor. *physica status solidi (RRL)–Rapid Research Letters* **13**, 1800629 (2019).

[47] Xu, N., Fang, L., Kim, K. M. & Hwang, C. S. Time-efficient stateful dual-bit-memristor logic. *physica status solidi (RRL)–Rapid Research Letters* **13**, 1900033 (2019).

[48] Li, S. *et al.* Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *Proceedings of the Design Automation Conference (DAC)*, 173 (ACM, 2016).

[49] Xie, L. *et al.* Scouting logic: A novel memristor-based logic design for resistive computing. In *Proceedings of the IEEE Symposium on VLSI (ISVLSI)*, 176–181 (2017).

[50] Maan, A. K., Jayadevi, D. A. & James, A. P. A survey of memristive threshold logic circuits. *IEEE Transactions on Neural Networks and Learning Systems* **28**, 1734–1746 (2016).

[51] Burr, G. W. *et al.* Neuromorphic computing using non-volatile memory. *Advances in Physics: X* **2**, 89–124 (2017).

[52] Ielmini, D. & Wong, H.-S. P. In-memory computing with resistive switching devices. *Nature Electronics* **1**, 333 (2018).

[53] Wang, Z. *et al.* Resistive switching materials for information processing. *Nature Reviews Materials* 1–23 (2020).

[54] Wright, C. D., Hosseini, P. & Diosdado, J. A. V. Beyond von-neumann computing with nanoscale phase-change memory devices. *Advanced Functional Materials* **23**, 2248–2254 (2013).

[55] Sebastian, A. *et al.* Brain-inspired computing using phase-change memory devices. *Journal of Applied Physics* **124**, 111101 (2018).

[56] Godse, A. P. & Godse, D. A. *Computer Organization and Architecture* (Technical Publications, 2008).

[57] Bojnordi, M. N. & Ipek, E. Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 1–13 (IEEE, 2016).

[58] Shafiee, A. *et al.* ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *SIGARCH Computer Architecture News* **44**, 14–26 (2016).

[59] Chi, P. *et al.* PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In *ACM SIGARCH Computer Architecture News*, vol. 44, 27–39 (IEEE Press, 2016).

[60] Song, L., Qian, X., Li, H. & Chen, Y. PIPELAYER: A pipelined ReRAM-based accelerator for deep learning. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 541–552 (IEEE, 2017).

[61] Zidan, M. A. *et al.* A general memristor-based partial differential equation solver. *Nature Electronics* **1**, 411 (2018).

[62] Higham, N. J. *Accuracy and stability of numerical algorithms*, vol. 80 (Society for Industrial and Applied Mathematics, 2002).

[63] Bekas, C., Curioni, A. & Fedulova, I. Low cost high performance uncertainty quantification. In *Workshop on High Performance Computational Finance*, 8:1–8:8 (ACM, 2009).

[64] Le Gallo, M. *et al.* Mixed-precision in-memory computing. *Nature Electronics* **1**, 246–253 (2018).

[65] Liu, S., Wang, Y., Fardad, M. & Varshney, P. K. A memristor-based optimization framework for artificial intelligence applications. *IEEE Circuits and Systems Magazine* **18**, 29–44 (2018).

[66] Sun, Z. *et al.* Solving matrix equations in one step with cross-point resistive arrays. *Proceedings of the National Academy of Sciences* **116**, 4123–4128 (2019).

[67] Sturges, R. H. Analog matrix inversion (robot kinematics). *IEEE Journal on Robotics and Automation* **4**, 157–162 (1988).

[68] Feinberg, B., Wang, S. & Ipek, E. Making memristive neural network accelerators reliable. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 52–65 (IEEE, 2018).

[69] Feinberg, B., Vengalam, U. K. R., Whitehair, N., Wang, S. & Ipek, E. Enabling scientific computing on memristive accelerators. In *International Symposium on Computer Architecture (ISCA)*, 367–382 (IEEE, 2018).

[70] Li, C. *et al.* Analogue signal and image processing with large memristor crossbars. *Nature Electronics* **1**, 52 (2018).

[71] Le Gallo, M., Sebastian, A., Cherubini, G., Giefers, H. & Eleftheriou, E. Compressed sensing with approximate message passing using in-memory computing. *IEEE Transactions on Electron Devices* **65**, 4304–4312 (2018).

[72] Cai, F. *et al.* Harnessing intrinsic noise in memristor hopfield neural networks for combinatorial optimization. *arXiv preprint arXiv:1903.11194* (2019).

[73] Mostafa, H., Müller, L. K. & Indiveri, G. An event-based architecture for solving constraint satisfaction problems. *Nature Communications* **6**, 8941 (2015).

[74] Parihar, A., Shukla, N., Jerry, M., Datta, S. & Raychowdhury, A. Vertex coloring of graphs via phase dynamics of coupled oscillatory networks. *Scientific reports* **7**, 911 (2017).

[75] Kumar, S., Strachan, J. P. & Williams, R. S. Chaotic dynamics in nanoscale NbO_2 Mott memristors for analogue computing. *Nature* **548**, 318 (2017).

[76] Torrejon, J. *et al.* Neuromorphic computing with nanoscale spintronic oscillators. *Nature* **547**, 428 (2017).

[77] Seo, J. *et al.* On-chip sparse learning acceleration with CMOS and resistive synaptic devices. *IEEE Transactions on Nanotechnology* **14**, 969–979 (2015).

[78] Sheridan, P. M. *et al.* Sparse coding with memristor networks. *Nature Nanotechnology* **12**, 784 (2017).

[79] Sheridan, P. M., Du, C. & Lu, W. D. Feature extraction using memristor networks. *IEEE Transactions on Neural Networks and Learning Systems* **27**, 2327–2336 (2016).

[80] Choi, S., Sheridan, P. & Lu, W. D. Data clustering using memristor networks. *Scientific reports* **5**, 10492 (2015).

[81] Karam, R., Puri, R., Ghosh, S. & Bhunia, S. Emerging trends in design and applications of memory-based computing and content-addressable memories. *Proceedings of the IEEE* **103**, 1311–1330 (2015).

[82] Rahimi, A. *et al.* High-dimensional computing as a nanoscalable paradigm. *IEEE Transactions on Circuits and Systems I: Regular Papers* **64**, 2508–2521 (2017).

[83] Wu, T. F. *et al.* Hyperdimensional computing exploiting carbon nanotube FETs, resistive RAM, and their monolithic 3D integration. *IEEE Journal of Solid-State Circuits* **53**, 3183–3196 (2018).

[84] Graves, A. *et al.* Hybrid computing using a neural network with dynamic external memory. *Nature* **538**, 471 (2016).

[85] Ni, K. *et al.* Ferroelectric ternary content-addressable memory for one-shot learning. *Nature Electronics* **2**, 521–529 (2019).

[86] Eryilmaz, S. B. *et al.* Brain-like associative learning using a nanoscale non-volatile phase change synaptic device array. *Frontiers in Neuroscience* **8**, 205 (2014).

[87] Hu, S. *et al.* Associative memory realized by a reconfigurable memristive Hopfield neural network. *Nature Communications* **6**, 7522 (2015).

[88] Kavehei, O. *et al.* An associative capacitive network based on nanoscale complementary resistive switches for memory-intensive computing. *Nanoscale* **5**, 5119–5128 (2013).

[89] Du, C. *et al.* Reservoir computing using dynamic memristors for temporal information processing. *Nature Communications* **8**, 2204 (2017).

[90] Sebastian, A. *et al.* Temporal correlation detection using computational phase-change memory. *Nature Communications* **8**, 1115 (2017).

[91] LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436 (2015).

[92] He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778 (IEEE, 2016).

[93] LeCun, Y. Deep learning hardware: Past, present, and future. In *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, 12–19 (2019).

[94] Chen, Y., Yang, T., Emer, J. & Sze, V. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* **9**, 292–308 (2019).

[95] Dazzi, M. *et al.* 5 parallel prism: A topology for pipelined implementations of convolutional neural networks using computational memory. In *NeurIPS MLSys Workshop* (2019).

[96] Jia, Z., Maggioni, M., Smith, J. & Scarpazza, D. P. Dissecting the NVidia Turing T4 GPU via microbenchmarking. *arXiv preprint arXiv:1903.07486* (2019).

[97] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R. & Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* **18**, 6869–6898 (2017).

[98] Xue, C. *et al.* 24.1 a 1mb multibit ReRAM computing-in-memory macro with 14.6ns parallel MAC computing time for CNN based AI edge processors. In *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, 388–390 (2019).

[99] Hu, M. *et al.* Memristor-based analog computation and neural network classification with a dot product engine. *Advanced Materials* **30**, 1705914 (2018).

[100] Yao, P. *et al.* Fully hardware-implemented memristor convolutional neural network. *Nature* **577**, 641–646 (2020).

[101] Suri, M. *et al.* Phase change memory as synapse for ultra-dense neuromorphic systems: Application to complex visual pattern extraction. In *Proceedings of the International Electron Devices Meeting (IEDM)*, 4.4.1–4.4.4 (2011).

[102] Chen, W.-H. *et al.* CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors. *Nature Electronics* **2**, 420–428 (2019).

[103] Murray, A. F. & Edwards, P. J. Enhanced mlp performance and fault tolerance resulting from synaptic weight noise during training. *IEEE Transactions on Neural Networks* **5**, 792–802 (1994).

[104] Liu, B. *et al.* Vortex: Variation-aware training for memristor X-bar. In *Proceedings of the Design Automation Conference (DAC)*, 1–6

(2015).

[105] Sebastian, A. *et al.* Computational memory-based inference and training of deep neural networks. In *Proceedings of the Symposium on VLSI Technology*, T168–T169 (IEEE, 2019).

[106] Gokmen, T., Onen, M. & Haensch, W. Training deep convolutional neural networks with resistive cross-point devices. *Frontiers in Neuroscience* **11**, 538 (2017).

[107] Alibart, F., Zamanidoost, E. & Strukov, D. B. Pattern classification by memristive crossbar circuits using ex situ and in situ training. *Nature Communications* **4**, 2072 (2013).

[108] Burr, G. W. *et al.* Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element. *IEEE Transactions on Electron Devices* **62**, 3498–3507 (2015).

[109] Gokmen, T. & Vlasov, Y. Acceleration of deep neural network training with resistive cross-point devices: design considerations. *Frontiers in neuroscience* **10**, 333 (2016).

[110] Agarwal, S. *et al.* Achieving ideal accuracies in analog neuromorphic computing using periodic carry. In *Proceedings of the Symposium on VLSI Technology*, T174–T175 (2017).

[111] Ambrogio, S. *et al.* Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature* **558**, 60 (2018).

[112] Yu, S. Neuro-inspired computing with emerging nonvolatile memory. *Proceedings of the IEEE* **106**, 260–285 (2018).

[113] Prezioso, M. *et al.* Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* **521**, 61 (2015).

[114] Yao, P. *et al.* Face classification using electronic synapses. *Nature Communications* **8**, 15199 (2017).

[115] Li, C. *et al.* Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nature Communications* **9**, 2385 (2018).

[116] Nandakumar, S. *et al.* Mixed-precision architecture based on computational memory for training deep neural networks. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, 1–5 (IEEE, 2018).

[117] Pfeiffer, M. & Pfeil, T. Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in Neuroscience* **12**, 774 (2018).

[118] Diehl, P. U. *et al.* Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *International Joint Conference on Neural Networks (IJCNN)*, 1–8 (2015).

[119] Sengupta, A., Ye, Y., Wang, R., Liu, C. & Roy, K. Going deeper in spiking neural networks: VGG and residual architectures. *Frontiers in Neuroscience* **13**, 95 (2019).

[120] Esser, S. K., Appuswamy, R., Merolla, P., Arthur, J. V. & Modha, D. S. Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems*, 1117–1125 (2015).

[121] Lee, J. H., Delbruck, T. & Pfeiffer, M. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience* **10**, 508 (2016).

[122] Woźniak, S., Pantazi, A. & Eleftheriou, E. Deep networks incorporating spiking neural dynamics. *arXiv preprint arXiv:1812.07040* (2018).

[123] Benjamin, B. V. *et al.* Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE* **102**, 699–716 (2014).

[124] Qiao, N. *et al.* A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Frontiers in Neuroscience* **9**, 141 (2015).

[125] Kuzum, D., Jeyasingh, R. G., Lee, B. & Wong, H.-S. P. Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing. *Nano letters* **12**, 2179–2186 (2011).

[126] Kim, S. *et al.* NVM neuromorphic core with 64k-cell (256-by-256) phase change memory synaptic array with on-chip neuron circuits for continuous in-situ learning. In *Proceedings of the International Electron Devices Meeting (IEDM)*, 17–1 (IEEE, 2015).

[127] Tuma, T., Le Gallo, M., Sebastian, A. & Eleftheriou, E. Detecting correlations using phase-change neurons and synapses. *IEEE Electron Device Letters* **37**, 1238–1241 (2016).

[128] Pantazi, A., Woźniak, S., Tuma, T. & Eleftheriou, E. All-memristive neuromorphic computing with level-tuned neurons. *Nanotechnology* **27**, 355205 (2016).

[129] Covi, E. *et al.* Analog memristive synapse in spiking networks implementing unsupervised learning. *Frontiers in Neuroscience* **10**, 482 (2016).

[130] Serb, A. *et al.* Unsupervised learning in probabilistic neural networks with multi-state metal-oxide memristive synapses. *Nature Communications* **7**, 12611 (2016).

[131] Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J. & Masquelier, T. STDP-based spiking deep convolutional neural networks for object recognition. *Neural Networks* **99**, 56–67 (2018).

[132] Moraitis, T., Sebastian, A. & Eleftheriou, E. The role of short-term plasticity in neuromorphic learning: Learning from the timing of rate-varying events with fatiguing spike-timing-dependent plasticity. *IEEE Nanotechnology Magazine* **12**, 45–53 (2018).

[133] Wang, Z. *et al.* Memristors with diffusive dynamics as synaptic emulators for neuromorphic computing. *Nature Materials* **16**, 101 (2017).

[134] Carboni, R. & Ielmini, D. Stochastic memory devices for security and computing. *Advanced Electronic Materials* 1900198 (2019).

[135] Jo, S. H., Kim, K.-H. & Lu, W. Programmable resistance switching in nanoscale two-terminal devices. *Nano letters* **9**, 496–500 (2008).

[136] Le Gallo, M., Athmanathan, A., Krebs, D. & Sebastian, A. Evidence for thermally assisted threshold switching behavior in nanoscale phase-change memory cells. *Journal of Applied Physics* **119**, 025704 (2016).

[137] Le Gallo, M., Tuma, T., Zipoli, F., Sebastian, A. & Eleftheriou, E. Inherent stochasticity in phase-change memory devices. In *2016 46th European Solid-State Device Research Conference (ESSDERC)*, 373–376 (IEEE, 2016).

[138] Alaghi, A. & Hayes, J. P. Survey of stochastic computing. *ACM Transactions on Embedded computing systems (TECS)* **12**, 92 (2013).

[139] Gupta, S., Agrawal, A., Gopalakrishnan, K. & Narayanan, P. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, 1737–1746 (2015).

[140] Yang, K. *et al.* 16.3 a 23mb/s 23pj/b fully synthesized true-random-number generator in 28nm and 65nm CMOS. In *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, 280–281 (IEEE, 2014).

[141] Jiang, H. *et al.* A novel true random number generator based on a stochastic diffusive memristor. *Nature Communications* **8**, 882 (2017).

[142] Gaba, S., Sheridan, P., Zhou, J., Choi, S. & Lu, W. Stochastic memristive devices for computing and neuromorphic applications. *Nanoscale* **5**, 5872–5878 (2013).

[143] Balatti, S. *et al.* Physical unbiased generation of random numbers with coupled resistive switching devices. *IEEE Transactions on Electron Devices* **63**, 2029–2035 (2016).

[144] Choi, W. H. *et al.* A magnetic tunnel junction based true random number generator with conditional perturb and real-time output probability tracking. In *Proceedings of the International Electron Devices Meeting*, 12–5 (IEEE, 2014).

[145] Carboni, R. *et al.* Random number generation by differential read of stochastic switching in spin-transfer torque memory. *IEEE Electron Device Letters* **39**, 951–954 (2018).

[146] Shim, Y., Chen, S., Sengupta, A. & Roy, K. Stochastic spin-orbit torque devices as elements for bayesian inference. *Scientific reports* **7**, 14101 (2017).

[147] Tuma, T., Pantazi, A., Le Gallo, M., Sebastian, A. & Eleftheriou, E. Stochastic phase-change neurons. *Nature Nanotechnology* **11**, 693–699 (2016).

[148] Mizrahi, A. *et al.* Neural-like computing with populations of superparamagnetic basis functions. *Nature Communications* **9**, 1533 (2018).

[149] Bichler, O. *et al.* Visual pattern extraction using energy-efficient 2-PCM synapse neuromorphic architecture. *IEEE Transactions on Electron Devices* **59**, 2206–2214 (2012).

[150] Holcomb, D. E., Burleson, W. P. & Fu, K. Power-up SRAM state as an identifying fingerprint and source of true random numbers. *IEEE Transactions on Computers* **58**, 1198–1210 (2009).

[151] Gao, L., Chen, P.-Y., Liu, R. & Yu, S. Physical unclonable function exploiting sneak paths in resistive cross-point array. *IEEE Transactions on Electron Devices* **63**, 3109–3115 (2016).

[152] Nili, H. *et al.* Hardware-intrinsic security primitives enabled by analogue state and nonlinear conductance variations in integrated memristors. *Nature Electronics* **1**, 197 (2018).

[153] Jiang, H. *et al.* A provable key destruction scheme based on memristive crossbar arrays. *Nature Electronics* **1**, 548 (2018).

[154] Talati, N., Gupta, S., Mane, P. & Kvatinsky, S. Logic design within memristive memories using memristor-aided logic (MAGIC). *IEEE Transactions on Nanotechnology* **15**, 635–650 (2016).

[155] Cheng, L. *et al.* Functional demonstration of a memristive arithmetic logic unit (MemALU) for in-memory computing. *Advanced Functional Materials* (2019).

[156] Haj-Ali, A., Ben-Hur, R., Wald, N., Ronen, R. & Kvatinsky, S. IMAGING: In-memory algorithms for image processing. *IEEE Transactions on Circuits and Systems I: Regular Papers* **65**, 4258–4271 (2018).

[157] Hamdioui, S. *et al.* Applications of computation-in-memory architectures based on memristive devices. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 486–491 (IEEE, 2019).

[158] Xiong, F., Liao, A. D., Estrada, D. & Pop, E. Low-power switching of phase-change materials with carbon nanotube electrodes. *Science* **332**, 568–570 (2011).

[159] Li, K.-S. *et al.* Utilizing sub-5 nm sidewall electrode technology for atomic-scale resistive memory fabrication. In *Proceedings of the Symposium on VLSI Technology*, 1–2 (IEEE, 2014).

[160] Salinga, M. *et al.* Monatomic phase change memory. *Nature Materials* 1 (2018).

[161] Pi, S. *et al.* Memristor crossbar arrays with 6-nm half-pitch and 2-nm critical dimension. *Nature nanotechnology* **14**, 35 (2019).

[162] Brivio, S., Frascaroli, J. & Spiga, S. Role of Al doping in the filament disruption in HfO_2 resistance switches. *Nanotechnology* **28**, 395202 (2017).

[163] Choi, S. *et al.* SiGe epitaxial memory for neuromorphic computing with reproducible high performance based on engineered dislocations. *Nature Materials* **17**, 335 (2018).

[164] Boybat, I. *et al.* Neuromorphic computing with multi-memristive synapses. *Nature Communications* **9**, 2514 (2018).

[165] Koelmans, W. W. *et al.* Projected phase-change memory devices. *Nature Communications* **6**, 8181 (2015).

[166] Giannopoulos, I. *et al.* 8-bit precision in-memory multiplication with projected phase-change memory. In *Proceedings of the International Electron Devices Meeting (IEDM)*, 27–7 (IEEE, 2018).

[167] Chen, Y. *et al.* DaDianNao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 609–622 (IEEE Computer Society, 2014).

[168] Ankit, A. *et al.* PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 715–731 (ACM, 2019).

[169] Eleftheriou, E. *et al.* Deep learning acceleration based on in-memory computing. *IBM Journal of Research and Development* (2019).

[170] Yoon, K. J., Bae, W., Jeong, D.-K. & Hwang, C. S. Comprehensive writing margin analysis and its application to stacked one diode-one memory device for high-density crossbar resistance switching random access memory. *Advanced Electronic Materials* **2**, 1600326 (2016).

[171] Le Gallo, M., Sebastian, A., Cherubini, G., Giefers, H. & Eleftheriou, E. Compressed sensing recovery using computational memory. In *Proceedings of the International Electron Devices Meeting (IEDM)*, 28–3 (IEEE, 2017).

[172] van de Burgt, Y. *et al.* A non-volatile organic electrochemical device as a low-voltage artificial synapse for neuromorphic computing. *Nature Materials* **16**, 414 (2017).

[173] Tang, J. *et al.* ECRAM as scalable synaptic cell for high-speed, low-power neuromorphic computing. In *Proceedings of the International Electron Devices Meeting (IEDM)*, 13–1 (IEEE, 2018).

[174] Fuller, E. J. *et al.* Parallel programming of an ionic floating-gate memory array for scalable neuromorphic computing. *Science* **364**,

570–574 (2019).

[175] Kimura, H. *et al.* Complementary ferroelectric-capacitor logic for low-power logic-in-memory VLSI. *IEEE Journal of Solid-State Circuits* **39**, 919–926 (2004).

[176] Aziz, A. *et al.* Computing with ferroelectric FETs: Devices, models, systems, and applications. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1289–1298 (IEEE, 2018).

[177] Chanthbouala, A. *et al.* A ferroelectric memristor. *Nature Materials* **11**, 860 (2012).

[178] Ríos, C. *et al.* Integrated all-photonic non-volatile multi-level memory. *Nature Photonics* **9**, 725 (2015).

[179] Wuttig, M., Bhaskaran, H. & Taubner, T. Phase-change materials for non-volatile photonic applications. *Nature Photonics* **11**, 465 (2017).

[180] Ríos, C. *et al.* In-memory computing on a photonic platform. *Science Advances* **5** (2019).

ACKNOWLEDGEMENTS

We would like to thank Tomas Tuma for technical discussions and assistance with scientific illustrations, Ghazi Sarwat and Irem Boybat for critical review of the manuscript, and Linda Rudin and Nicola Gustafsson for editorial help. A.S. acknowledges funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement number 682675).

ADDITIONAL INFORMATION

Reprints and permission information is available online at www.nature.com/reprints. Correspondence and requests for materials should be addressed to A.S.

COMPETING FINANCIAL INTERESTS

The authors declare no competing financial interests.

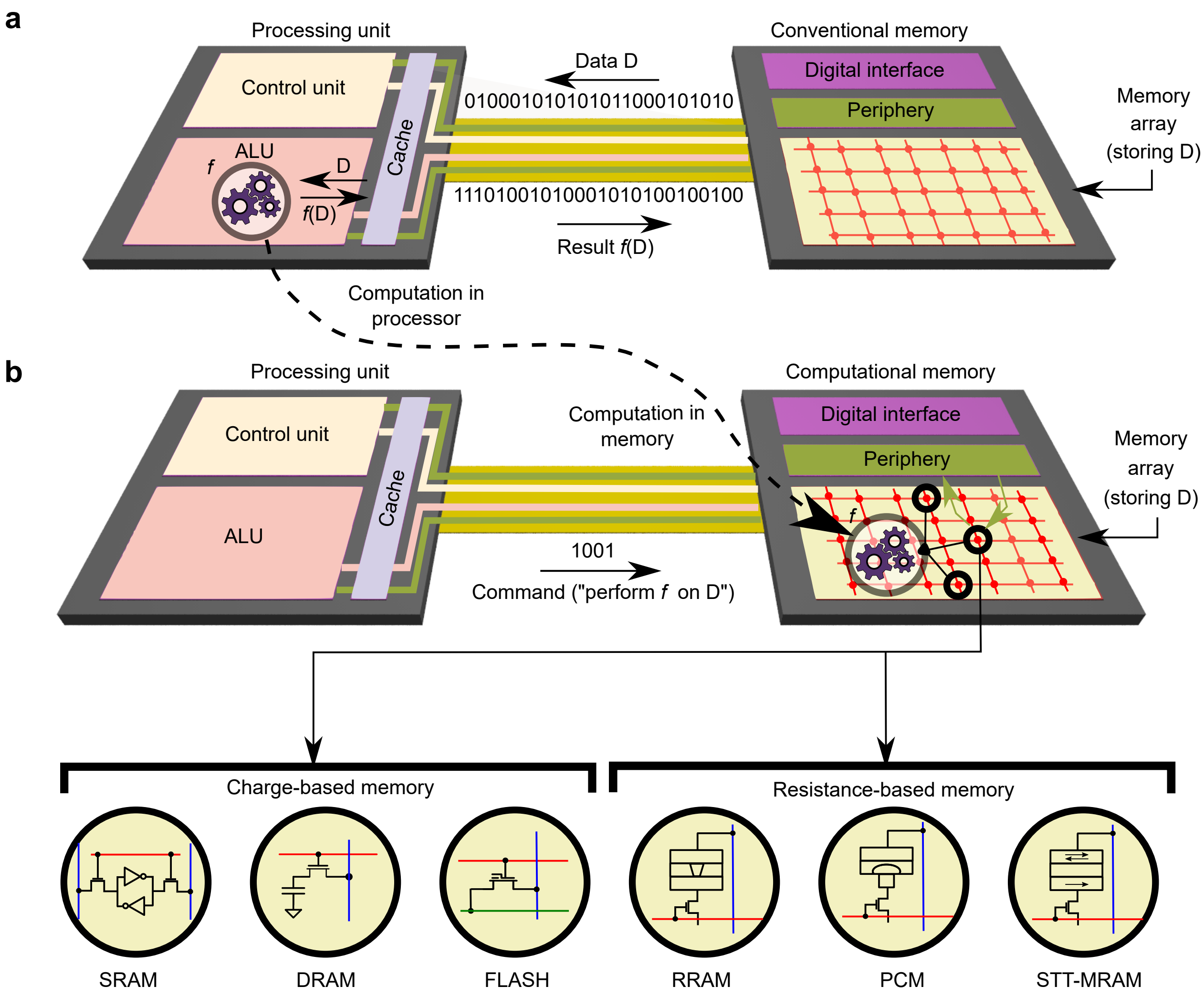

FIG. 1. **In-memory computing.** **a** In a conventional computing system, when an operation f is performed on data D, D has to be moved into a processing unit, leading to significant costs in latency and energy. **b** In the case of in-memory computing, $f(D)$ is performed within a computational memory unit by exploiting the physical attributes of the memory devices, thus obviating the need to move D to the processing unit. The computational tasks are performed within the confines of the memory array and its peripheral circuitry, albeit without deciphering the content of the individual memory elements. Both charge-based memory technologies, such as static-random access memory (SRAM), dynamic random access memory (DRAM) and Flash memory, and resistance-based memory technologies, such as resistive random access memory (RRAM), phase change memory (PCM) and spin-transfer torque magnetic random access memory (STT-MRAM), can serve as elements of such a computational memory unit.

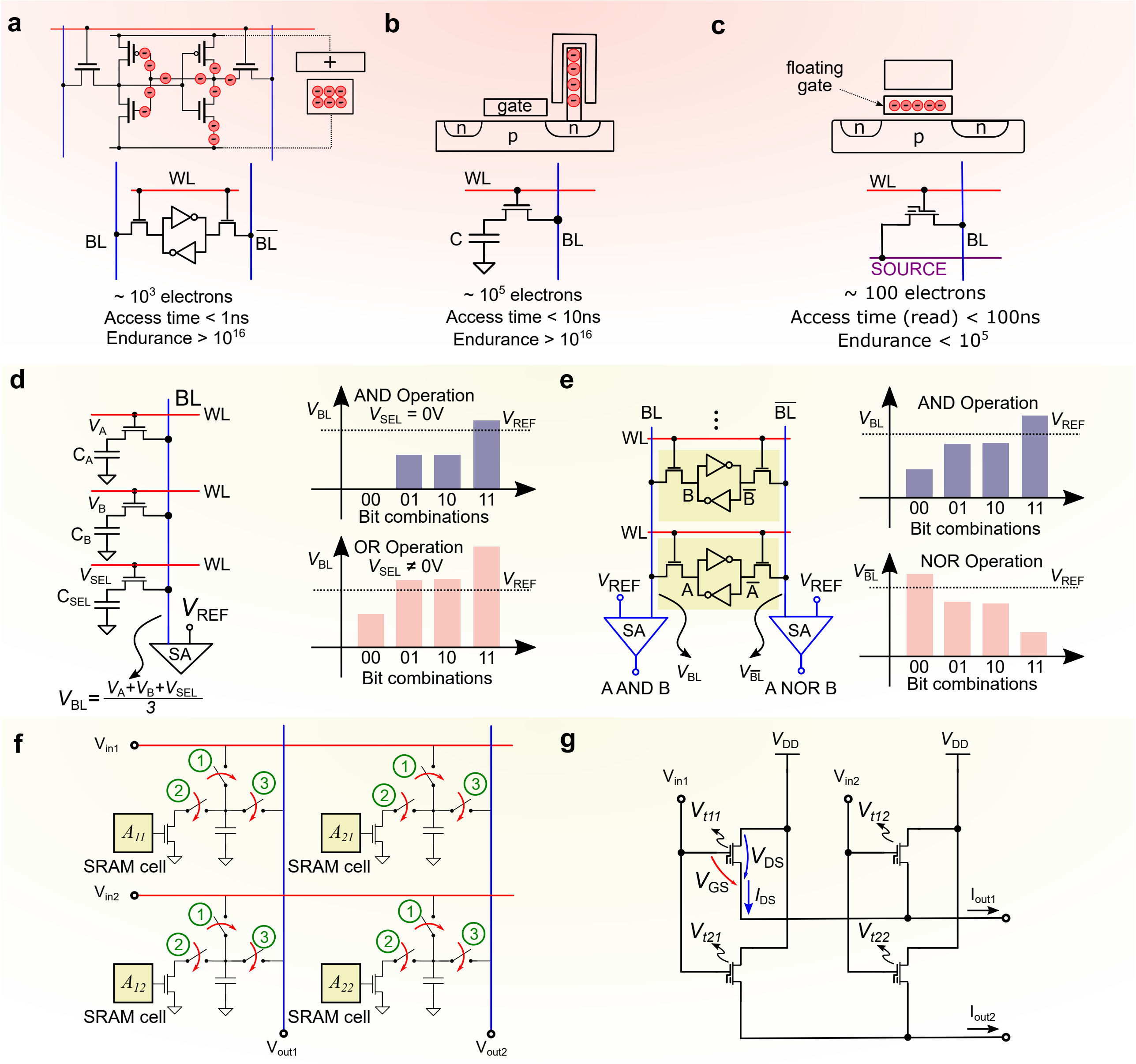

FIG. 2. **Charge-based memory devices and computational primitives. a** A 6T SRAM cell consists of two CMOS inverters connected back to back. The charge is confined within the barriers formed by FET channels and by gate insulators. The stored charge retention is small and an external source constantly replenishes the lost charge. SRAM has almost unlimited cycling endurance and sub-nanosecond read and write access times. **b** A DRAM cell comprises a capacitor that serves as the storage node, which is connected in series to a FET. **c** The storage node of a Flash memory cell is coupled to the gate of a FET. **d** Schematic illustration of bit-wise logical operations performed using three DRAM cells. The operands are stored in cells A and B. AND or OR operations are performed by simultaneously activating the three WLs corresponding to the cells. The logical state of cell SEL is used to dictate whether an AND or an OR operation is performed, with logical one and zero corresponding to OR and AND operations, respectively. The BL voltage corresponds to the average voltage across the three capacitors and is sensed using a sense amplifier with a decision threshold voltage of V_{REF}. **e** Bit-wise logical operations using an SRAM array. The BL and $\overline{BL}$ are pre-charged to the supply voltage, V_{DD}, prior to the execution of the operation. After deactivation of the pre-charge signal, both the WLs are activated so that both BL and $\overline{BL}$ are discharged at different rates that depend on the data stored in the bit-cells. When the two activated SRAM cells in a column are both ones (zeros), V_{BL} ($V_{\overline{BL}}$) will be comparable to V_{DD}, whereas for the other bit combinations both V_{BL} and $V_{\overline{BL}}$ will be lower than V_{DD}. Hence, by sensing V_{BL} and $V_{\overline{BL}}$ with a SA, AND and NOR operations are performed, respectively. **f** Schematic illustration of performing MVM operation using an array of SRAM cells and capacitors. The SRAM cells are used to store the elements of the binary matrix. In the first step, the inputs are provided per row that charges the capacitors on that row to a value proportional to the input. In step two, the capacitors that are associated with the SRAM elements storing 0s are discharged. Finally in step three, the capacitors are shorted along the columns performing a charge sharing operation so that the final voltage on the capacitors corresponds to the analog MVM result. **g** Illustration of an MVM operation performed using Flash memory devices. The current I_{DS} is a function of V_{t} as well as the cell's drain-source voltage V_{DS} and the gate-source voltage V_{GS}. By fixing V_{DS}, Kirchhoff's current law can be employed to perform MVM between a matrix, stored in terms of V_{t}, and a binary input vector that is used to modulate V_{GS}.

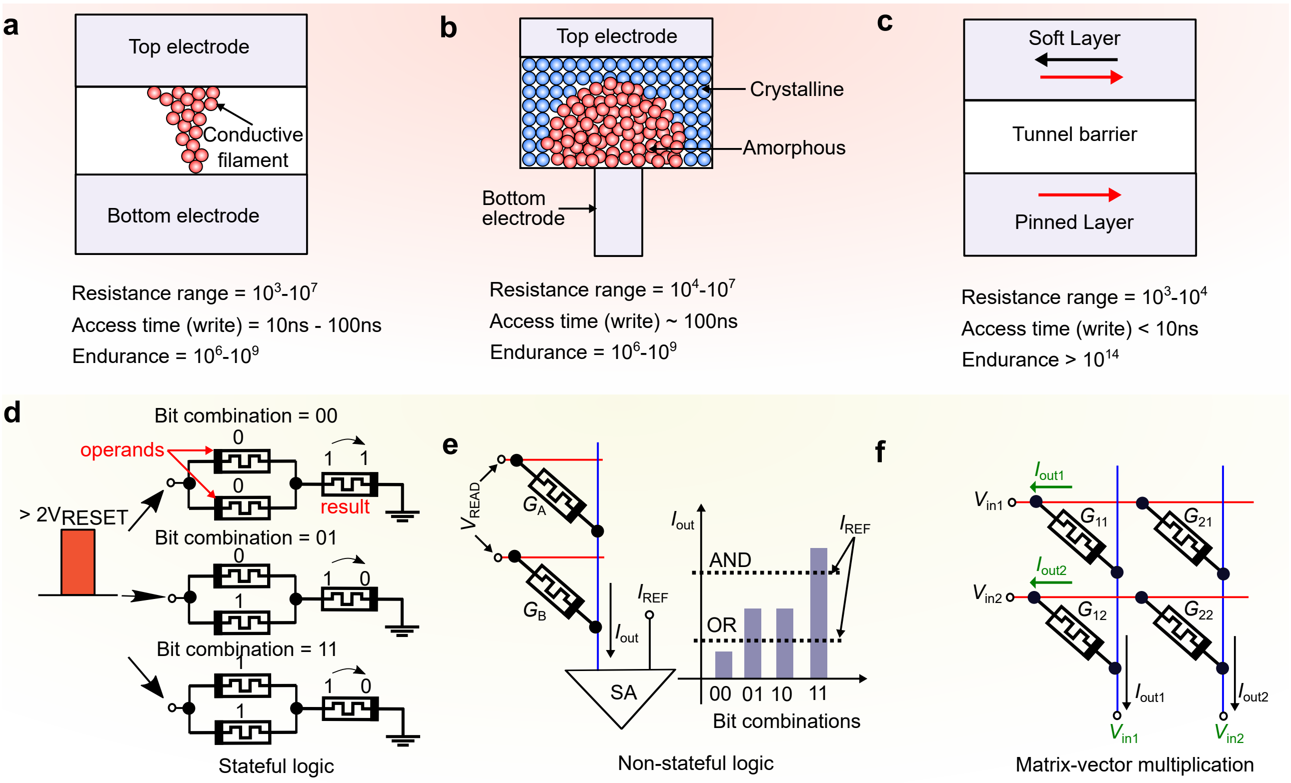

FIG. 3. **Resistance-based memory devices and computational primitives. a** An RRAM device in the low resistance state (LRS) where the conductive filament (CF) comprises a large concentration of defects for example oxygen vacancies in metal oxides or metallic ions injected from the electrodes. By the application of appropriate voltage pulses, the defects can be migrated back to the top electrode thus disconnecting the CF and achieving a high resistance state (HRS). **b** A mushroom-type PCM device in the HRS state where the amorphous phase blocks the bottom electrode. To create this state, a RESET pulse is applied that can melt a significant portion of the phase change material. When the pulse is stopped abruptly, the molten material quenches into the amorphous phase due to glass transition. When a current pulse of lesser amplitude is applied to the PCM device in the HRS state, a part of the amorphous region crystallizes. By fully crystallizing the phase change material, the LRS state is obtained. **c** An STT-MRAM device with two ferromagnetic layers (pinned and free) separated by a tunnel oxide layer. The magnetic polarization of the free layer can be changed upon writing. Depending on whether the ferromagnetic polarizations are parallel or antiparallel, the device assumes a low or high resistance, respectively. The transition to the parallel state takes place directly through conduction electrons, which are previously spin-polarized by the pinned layer. Subsequently, the magnetic polarization of the free layer is rotated using magnetic momentum conservation. To switch to the antiparallel state, an opposite voltage, and hence current direction, is employed. **d** Schematic illustration of a stateful NOR logic operation using 3 bipolar memristive devices [44]. Two devices represent the operands and one represents the result. First, the result device is initialized to logic 1 (LRS). Subsequently, a voltage pulse with an amplitude larger than twice that of V_{RESET} is applied simultaneously to both the operand devices. If either operand device is at logic 1 (LRS), then at least half of the voltage drops across the result device and the latter switches to logic 0 (HRS). Note that, due to the bipolar switching behavior, the operand devices remain unchanged as long as $V_{\text{SET}} \gg 2V_{\text{RESET}}$. When both the operand devices are at logic 0 (HRS), the voltage dropped across the result device is not sufficient to switch it to logic 0. Hence it remains at logic 1. Thus this simple circuit implements a NOR operation where all the logic state variables are represented purely in terms of resistance values. **e** Non-stateful AND and OR operations using 2 memristive devices and a variable threshold SA. By simultaneously activating multiple rows, and with the appropriate choice of current thresholds, it is possible to implement logical operations such as AND and OR. **f** To perform the operation $Ax = b$, the elements of A are mapped linearly to the conductance values of memristive devices organized in a crossbar configuration. The x values are mapped linearly to the amplitudes or durations of read voltages and are applied to the crossbar along the rows. The result of the computation, b, will be proportional to the resulting current measured along the columns of the array. Note that, if the inputs are mapped onto durations, the result b will be proportional to the total charge (e. g. current integrated over a certain fixed period of time). It is also possible to perform an MVM operation with the transpose of A using the same cross-bar configuration by applying the input voltage to the column lines and measuring the resulting current along the rows. The negative elements of x are typically applied as negative voltages whereas the negative elements of A are coded on separate devices together with a subtraction circuit.

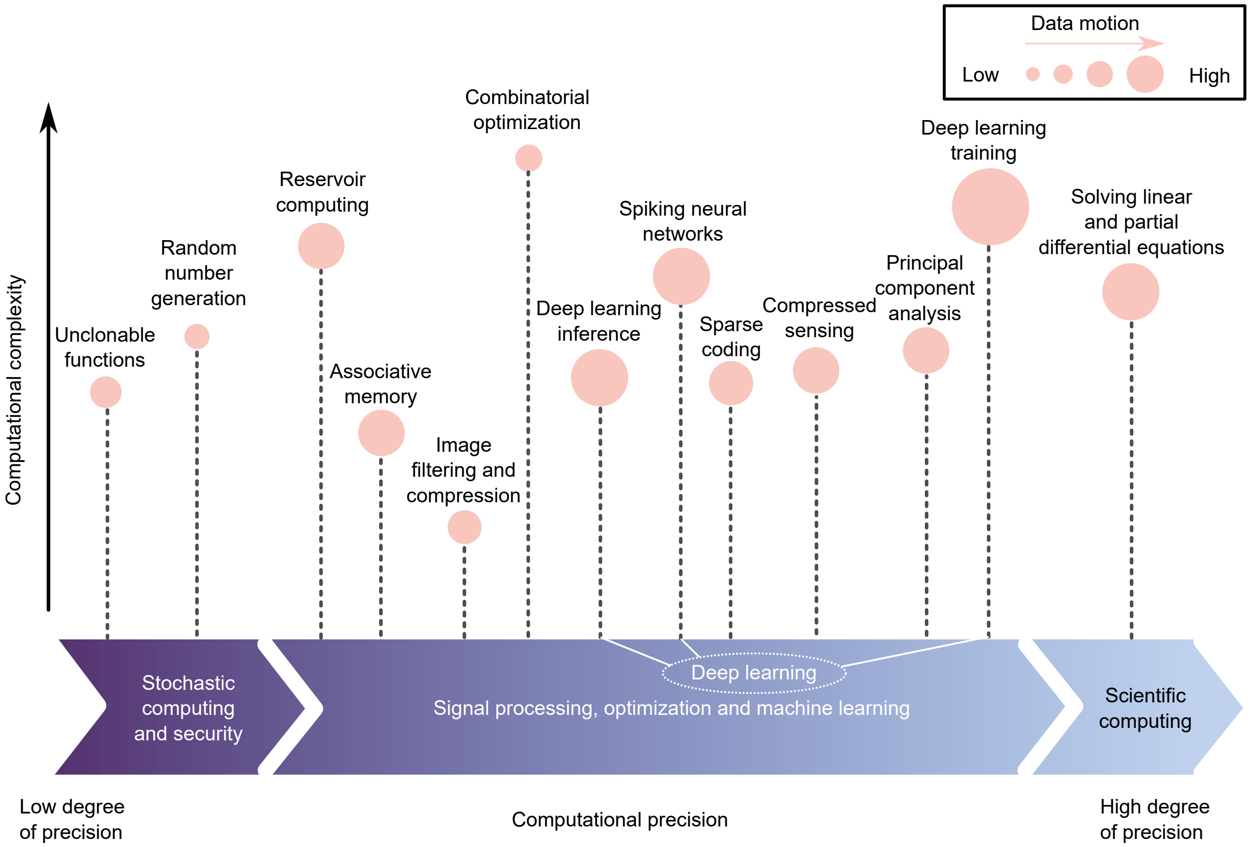

FIG. 4. **The application landscape for in-memory computing.** The applications are grouped into three main categories based on the overall degree of computational precision that is required. A qualitative measure of the computational complexity and data accesses involved in the different applications is also shown.

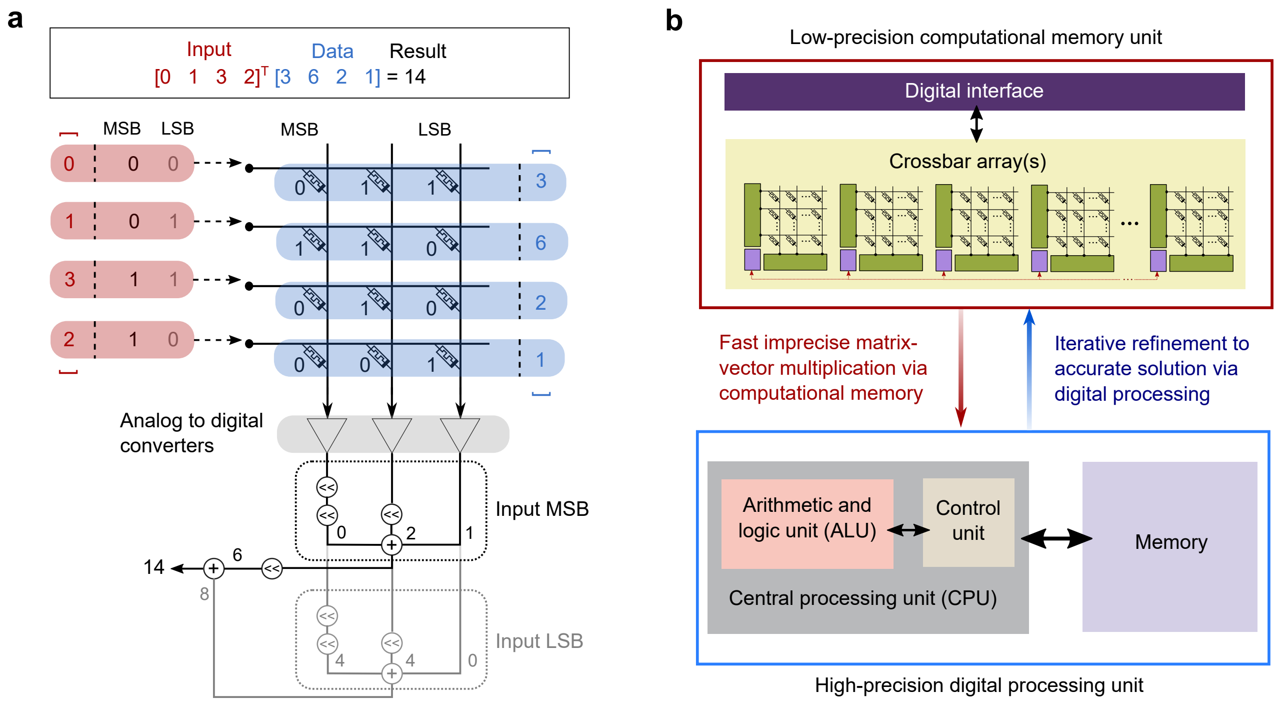
a
Input
Data
Result
[0 1 3 2]T [3 6 2 1] = 14
MSB
LSB
MSB
LSB
Analog to digital converters
Input MSB
Input LSB
14
6
8
b
Low-precision computational memory unit
Digital interface
Crossbar array(s)
Fast imprecise matrix-vector multiplication via computational memory
Iterative refinement to accurate solution via digital processing
Arithmetic and logic unit (ALU)
Control unit
Memory
Central processing unit (CPU)
High-precision digital processing unit

FIG. 5. **Increasing the precision of in-memory computing for scientific computing.** **a** Implementation of the bit-slicing concept in a crossbar array for an inner product operation. The 3-bit data vector is sliced into three 1-bit vectors stored on three separated columns of the crossbar array. The 2-bit input vector is sliced into two 1-bit vectors sequentially applied to the crossbar array as voltages. The outputs of the crossbar from the first input bit slice go through an analog to digital conversion and appropriate shifting prior to accumulation and storage in a local buffer as a partial inner product result. The second input bit slice undergoes the same process, producing the second partial inner product result. These two partial inner product results are added up, yielding the final result of the in-place inner-product vector operation. **b** The concept of mixed-precision in-memory computing used to iteratively improve the computed solution to a system of linear equations based on inaccurate MVM operations performed via analog in-memory computing.

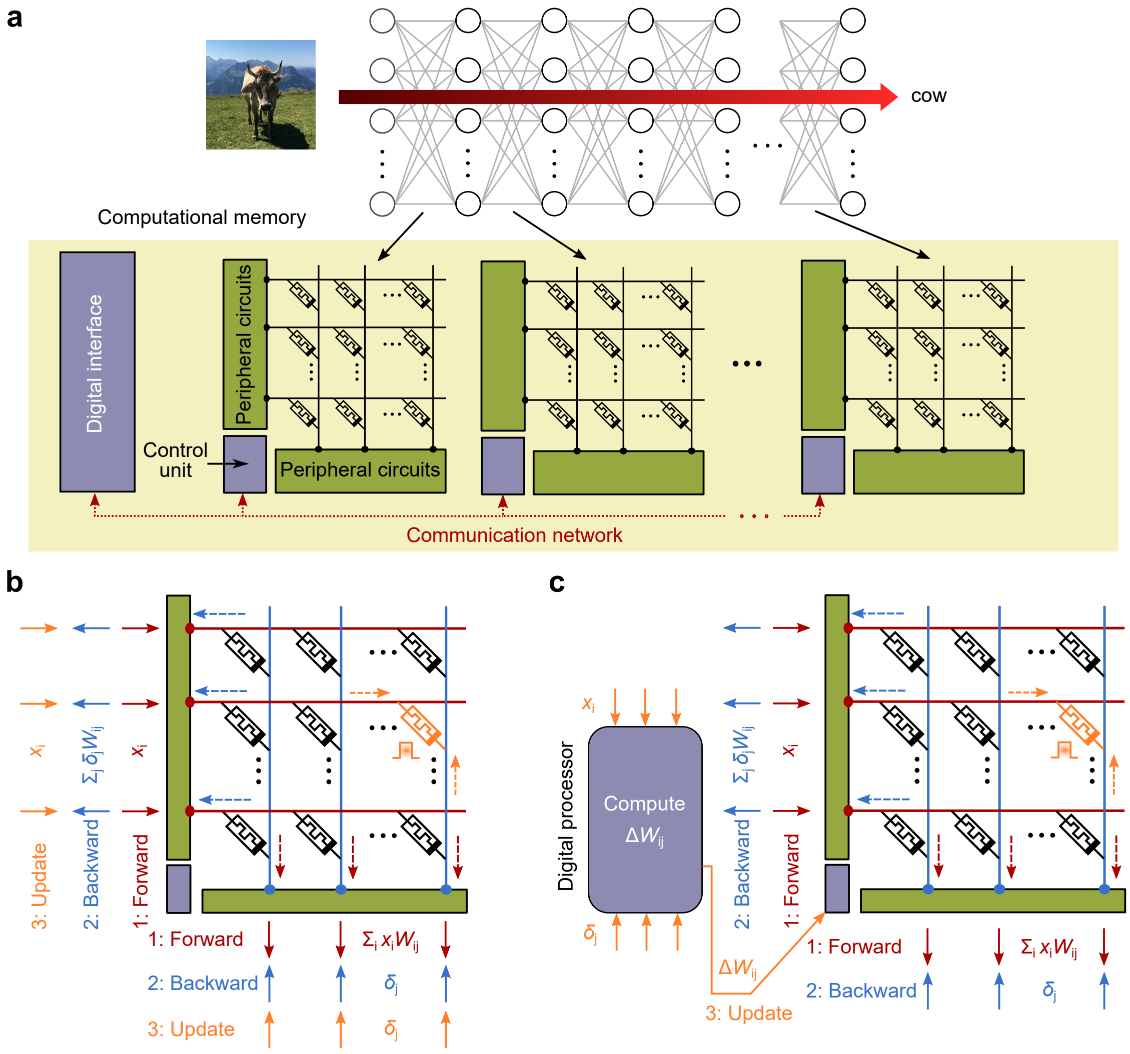

FIG. 6. **Deep learning training and inference using in-memory computing. a** Implementation of a feed-forward DNN on multiple crossbar arrays of memory devices. The synaptic weights are stored in the conductance or charge state of the memory devices. Each layer of the network is implemented in a different crossbar. Forward propagation of data through the network is performed by applying, for each layer, input data on the crossbar rows, and deciphering the results at the column level. The results are then passed through a nonlinear function implemented at the periphery and input to the next layer. A global communication network is used to send data from one array to another. **b** A first possible implementation of the three steps performed in training a layer of a neural network in a crossbar array. Forward and backward propagations are implemented by inputting activations x_i and errors δ_j on the rows and columns, respectively. An in-place weight update can be performed by sending pulses based on the values of x_i and δ_j from the rows and columns simultaneously. This implements an approximate outer product and programs the devices at the same time. **c** A second possible implementation, whereby the weight update ΔW_{ij} is computed in the digital domain and applied via programming pulses to the corresponding devices.

Device	SRAM	SRAM	SRAM	nor-Flash	RRAM	RRAM
CMOS technology	65-nm	65-nm	65-nm	180-nm	130-nm	55-nm
Array size	16-kb	16-kb	2.4-Mb	100-kb	16-kb	1-Mb
Weight/activation precision	1-b/6-b	1-b/ternary	1-b/1-b	analog/analog	analog/8-b	3-b/2-b
Network	LeNet-5 CNN	MLP/CNN	5/9-layer CNN	2-layer MLP	5-layer CNN	CNN
Dataset	MNIST	MNIST/CIFAR-10	MNIST/CIFAR-10	MNIST	MNIST	CIFAR-10
Accuracy	98.3%	98.3%/85.7%	98.6%/83.3%	94.7%	96.2%	88.52%
Peak MAC efficiency[a]	40.3 TOPS/W	139 TOPS/W	658 TOPS/W	10 TOPS/W	11 TOPS/W	21.9 TOPS/W
Reference	[22]	[21]	[23]	[28]	[100]	[98]

[a] 1 multiply-and-accumulate (MAC) = 2 OPs.

TABLE I. State-of-the-art chip-level experimental demonstrations of neural network inference based on in-memory computing.

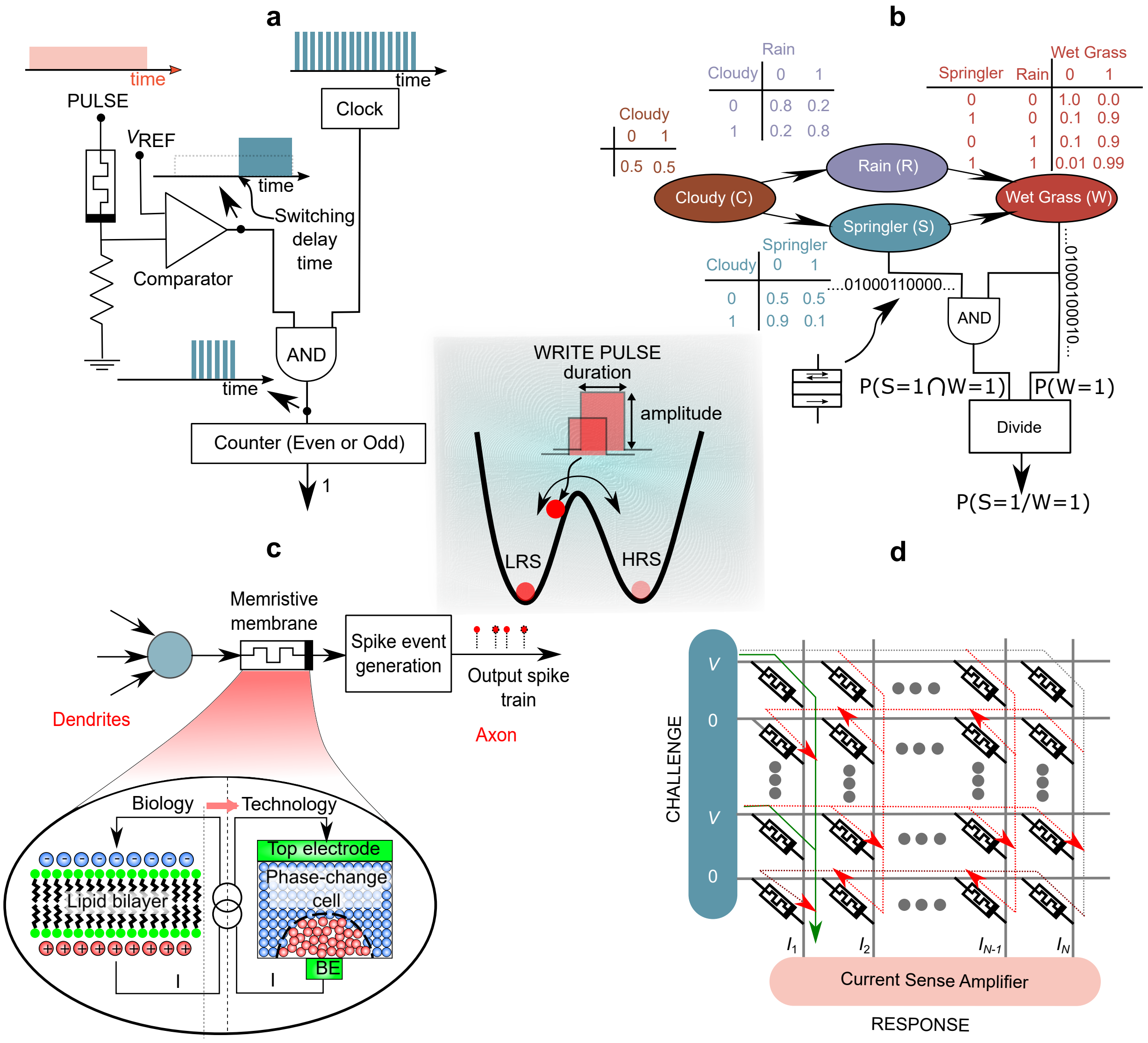

FIG. 7. **Stochasticity associated with memristive devices and applications in computing.** Resistance switching in memristive devices is intrinsically stochastic, with an ability to control the stochasticity via the voltage and duration of write pulses. **a** Schematic illustration of a circuitry that exploits memristive stochasticity for the generation of true random numbers [141]. The device is connected in series with a resistor in a voltage divider configuration. A write pulse of a certain fixed duration is applied to the device. A SET transition in the device after a stochastic delay time will cause the comparator to output a 1. The difference between the pulse duration and the delay time is measured by a counter in units of a fixed clock period. Based on whether the this time is an even or odd multiple of the clock period, a 0 or 1 bit is assigned. By applying a sequence of write pulses, a stochastic bit stream is generated. **b** A Bayesian network is shown where each node represents random variables and each link describes the direct dependence among them, quantified in terms of the transitional conditional probabilities. Such networks can be used to estimate the probability of hidden causes from a given observation. The required probability distributions to perform such probabilistic inference can be generated efficiently using stochastically switching memristive devices. For example the probabilities can be encoded within Poisson distributed binary bit streams generated using MRAM devices [146]. The associated computations such as the intersection operation can be implemented by multiplying the two bit streams with an AND gate. **c** The stochasticity associated with the SET process in PCM can be used to realize stochastically firing neurons. The key computational element is the neuronal membrane, which stores the membrane potential in the phase configuration of a PCM device. These devices enable the emulation of large and dense populations of neurons for bioinspired signal representation and computation. **d** Memristive crossbar arrays can be used to generate physically unclonable functions (PUF). The broad distribution of resistance values as well as the current sneak paths are exploited to obtain a large set of challenge-response pairs (CRP). For example in an $N \times N$ crossbar PUF depicted here, the challenge consists of an N-bit vector applied to the N rows. The current from the N columns is then read and converted to an N-bit response. The theoretical number of CRPs is 2^N.