nature electronics

ARTICLES

https://doi.org/10.1038/s41928-020-0410-3

In-memory hyperdimensional computing

Geethan Karunaratne[1,2], Manuel Le Gallo[1], Giovanni Cherubini[1], Luca Benini[2], Abbas Rahimi[2]✉ and Abu Sebastian[1]✉

Hyperdimensional computing is an emerging computational framework that takes inspiration from attributes of neuronal circuits including hyperdimensionality, fully distributed holographic representation and (pseudo)randomness. When employed for machine learning tasks, such as learning and classification, the framework involves manipulation and comparison of large patterns within memory. A key attribute of hyperdimensional computing is its robustness to the imperfections associated with the computational substrates on which it is implemented. It is therefore particularly amenable to emerging non-von Neumann approaches such as in-memory computing, where the physical attributes of nanoscale memristive devices are exploited to perform computation. Here, we report a complete in-memory hyperdimensional computing system in which all operations are implemented on two memristive crossbar engines together with peripheral digital complementary metal–oxide–semiconductor (CMOS) circuits. Our approach can achieve a near-optimum trade-off between design complexity and classification accuracy based on three prototypical hyperdimensional computing-related learning tasks: language classification, news classification and hand gesture recognition from electromyography signals. Experiments using 760,000 phase-change memory devices performing analog in-memory computing achieve comparable accuracies to software implementations.

Biological computing systems trade accuracy for efficiency. Thus, one solution to reduce energy consumption in artificial systems is to adopt computational approaches that are inherently robust to uncertainty. Hyperdimensional computing (HDC) is one such framework and is based on the observation that key aspects of human memory, perception and cognition can be explained by the mathematical properties of hyperdimensional spaces comprising high-dimensional binary vectors known as hypervectors. Hypervectors are defined as d-dimensional (where $d \geq 1{,}000$) (pseudo)random vectors with independent and identically distributed (i.i.d.) components[1]. When the dimensionality is in the thousands, a large number of quasi-orthogonal hypervectors exist. This allows HDC to combine such hypervectors into new hypervectors using well-defined vector space operations, defined such that the resulting hypervector is unique, and with the same dimension. A powerful system of computing can be built on the rich algebra of hypervectors[2]. Groups, rings and fields over hypervectors become the underlying computing structures with permutations, mappings and inverses as primitive computing operations.

In recent years, HDC has been employed in a range of applications, including machine learning, cognitive computing, robotics and traditional computing. It has shown significant promise in machine learning applications that involve temporal patterns, such as text classification[3], biomedical signal processing[4,5], multimodal sensor fusion[6] and distributed sensors[7,8]. A key advantage is that the training algorithm in HDC works in one or few shots: that is, object categories are learned from one or few examples, and in a single pass over the training data as opposed to many iterations. In the highlighted machine learning applications, HDC has achieved similar or higher accuracy with fewer training examples compared to support vector machines (SVMs)[4], extreme gradient boosting[9] and convolutional neural networks (CNNs)[10], and lower execution energy on embedded CPU/GPUs compared to SVMs[11], CNNs and long short-term memory[5]. Applications of HDC in cognitive computing include solving Raven's progressive matrices[12], functional imitation of concept learning in honey bees[13] and analogical reasoning[14]. In the field of robotics, HDC has been employed for learning sensorimotor control for active perception in robots[10]. In traditional forms of computing, HDC has been proposed for efficient representation of structured information[15] as well as the synthesis and execution of finite state automata[16] and variants of recurrent neural networks[17].

HDC begins by representing symbols with i.i.d. hypervectors that are combined by nearly i.i.d.-preserving operations, namely binding, bundling and permutation, and then stored in associative memories (AMs) to be recalled, matched, decomposed or reasoned about. This chain implies that failure in a component of a hypervector is not contagious and forms a computational framework that is intrinsically robust to defects, variations and noise[18]. The manipulation of large patterns stored in memory and its inherent robustness make HDC particularly well suited to emerging computing paradigms such as in-memory computing or computational memory based on emerging nanoscale resistive memory or memristive devices[19–23]. In one such work, a 3D vertical resistive random access memory (ReRAM) device was used to perform individual operations for HDC[24,25]. In another work, a carbon-nanotube field-effect transistor-based logic layer was integrated with ReRAMs, improving efficiency further[26]. However, these architectures offered only limited applications such as a single language recognition task[24,26] or a restricted binary classification version of the same task[26], and their evaluation is based on simulations and compact models derived from small prototypes with only 256 ReRAM cells[24] or a small 32 bit datapath for hypervector manipulations that results in three orders of magnitude higher latency overhead[26].

In this Article, we report a complete integrated in-memory HDC system in which all the operations of HDC are implemented on two planar memristive crossbar engines together with peripheral digital CMOS circuits. We devise a way of performing hypervector binding entirely within a first memristive crossbar using an in-memory read logic operation and hypervector bundling near the crossbar with CMOS logic. These key operations of HDC cooperatively encode hypervectors with high precision, while eliminating the need to

[1]IBM Research – Zurich, Rüschlikon, Switzerland. [2]Department of Information Technology and Electrical Engineering, ETH Zürich, Zürich, Switzerland. ✉e-mail: abbas@iis.ee.ethz.ch; ase@zurich.ibm.com

repeatedly program (write) the memristive devices. In contrast, previous work on HDC using memristive devices did not employ in-memory logic operations for binding; instead, a ReRAM-based XOR lookup table[24] or digital logic[26] was used. Moreover, the previous in-memory compute primitives for permutation[24] and bundling[26] resulted in repeated programming of the memristive devices, which is prohibitive given the limited cycling endurance.

In our architecture, an AM search is performed using a second memristive crossbar for in-memory dot-product operations on the encoded output hypervectors from the first crossbar, realizing the full HDC system functionality. Our combination of analog in-memory computing with CMOS logic allows continual functioning of the memristive crossbars with desired accuracy for a wide range of multiclass classification tasks. We verify the integrated inference functionality of the system through large-scale mixed hardware/software experiments, in which up to 49 $d=10{,}000$-dimensional hypervectors are encoded in 760,000 hardware phase-change memory (PCM) devices performing analog in-memory computing. Our experiments achieve comparable accuracies to the software baselines and surpass those reported in previous work on an emulated small ReRAM crossbar[24]. Furthermore, a complete system-level design of the in-memory HDC architecture synthesized using 65 nm CMOS technology demonstrates >6× end-to-end reductions in energy compared with a dedicated digital CMOS implementation. With our approach, we map all operations of HDC either in-memory or near-memory and demonstrate their integrated functionality for three specific machine learning related tasks.

The concept of in-memory HDC

When HDC is used for learning and classification, a set of i.i.d., hence quasi-orthogonal hypervectors, referred to as basis hypervectors, are first selected to represent each symbol associated with a dataset. For example, if the task is to classify an unknown text into the corresponding language, the symbols could be the letters of the alphabet. The basis hypervectors stay fixed throughout the computation. Assuming that there are h symbols, $\{s_i\}_1^h$, the set of the h, d-dimensional basis hypervectors $\{\mathbf{B}_i\}_1^h$ is referred to as the item memory (IM) (Fig. 1). Basis hypervectors serve as the basis from which further representations are made by applying a well-defined set of component-wise operations: addition of binary hypervectors [+] is defined as the component-wise majority, multiplication ($\oplus$) is defined as the component-wise exclusive-OR (XOR) (or equivalently as the component-wise exclusive-NOR (XNOR)) and permutation (ρ) is defined as a pseudo-random shuffling of the coordinates. Applied on dense binary hypervectors where each component has equal probability of being zero or one[27], all these operations produce a d-bit hypervector resulting in a closed system.

Subsequently, during the learning phase, the basis hypervectors in the IM are combined with the component-wise operations inside an encoder to compute, for example, a quasi-orthogonal n-gram hypervector representing an object of interest[28], and to add n-gram hypervectors from the same category of objects to produce a prototype hypervector representing the entire class of category. In the language example, the encoder would receive input text associated with a known language and would generate a prototype hypervector corresponding to that language. In this case, n determines the smallest number of symbols (letters in the example) that are combined while performing an n-gram encoding operation. When the encoder receives n consecutive symbols, $\{s[1], s[2], \ldots, s[n]\}$, it produces an n-gram hypervector through a binding operation given by

$$G(s[1], s[2], \cdots, s[n]) = B[1]\overline{\oplus}\rho(B[2])\overline{\oplus}\cdots\overline{\oplus}\rho^{n-1}(B[n]) \quad (1)$$

where $B[k]$ corresponds to the associated basis hypervector for symbol $s[k]$. The operator $\overline{\oplus}$ denotes the XNOR and ρ denotes a pseudo-random permutation operation, for example, a circular shift by 1 bit. The encoder then bundles several such n-gram hypervectors from the training data using component-wise addition followed by a binarization (majority function) to produce a prototype hypervector for the given class. The overall encoding operation results in c, d-dimensional prototype hypervectors (referred to as associative memory (AM)) assuming there are c classes.

When inference or classification is performed, a query hypervector (for example, from a text of unknown language) is generated identically to the way the prototype hypervectors are generated. Subsequently, the query hypervector is compared with the prototype hypervectors inside the AM to make the appropriate classification. Equation (2) defines how a query hypervector $\mathbf{Q}$ is compared against each of the prototype hypervector $\mathbf{P}_i$ out of c classes to find the predicted class with maximum similarity. This AM search operation can, for example, be performed by calculating the inverse Hamming distance:

$$\text{Class}_{\text{Pred}} = \underset{i\in\{1,\ldots,c\}}{\arg\max} \sum_{j=1}^{d} \mathbf{Q}(j)\overline{\oplus}\mathbf{P}_i(j) \quad (2)$$

One key observation is that the two main operations presented above, namely the encoding and AM search, are about manipulating and comparing large patterns within the memory. Both IM and AM (after learning) represent permanent hypervectors stored in the memory. As a lookup operation, different input symbols activate the corresponding stored patterns in the IM that are then combined inside or around memory with simple local operations to produce another pattern for comparison in AM. These component-wise arithmetic operations on patterns allow a high degree of parallelism as each hypervector component needs to communicate with only a local component or its immediate neighbours. This highly memory-centric aspect of HDC is the key motivation for the in-memory computing implementation proposed in this work.

The essential idea of in-memory HDC is to store the components of both the IM and the AM as the conductance values of nanoscale memristive devices[29,30] organized in crossbar arrays and enable HDC operations in or near to those devices (Fig. 1). The IM of h rows and d columns is stored in the first crossbar, where each basis hypervector is stored on a single row. To perform $\overline{\oplus}$ operations between the basis hypervectors for the n-gram encoding, an in-memory read logic primitive is employed. Unlike the majority of reported in-memory logic operations[31–33], the proposed in-memory read logic is non-stateful and this obviates the need for high write endurance of the memristive devices. Additional peripheral circuitry is used to implement the remaining permutations and component-wise additions needed in the encoder. The AM of c rows and d columns is implemented in the second crossbar, where each prototype hypervector is stored on a single row. During supervised learning, each prototype hypervector output from the first crossbar is programmed into a certain row of the AM based on the provided label. During inference, the query hypervector output from the first crossbar is input as voltages on the wordline driver, to perform the AM search using an in-memory dot product primitive. Because every memristive device in the AM and IM is reprogrammable, the representation of hypervectors is not hardcoded, unlike refs. [24–26], which used device variability for projection.

This design ideally fits the memory-centric architecture of HDC, because it allows us to perform the main computations on the IM and AM within the memory units with a high degree of parallelism. Furthermore, the IM and AM are only programmed once while training on a specific dataset, and the two types of in-memory computation that are employed involve just read operations. Therefore, non-volatile memristive devices are very well suited for implementing the IM and AM, and only binary conductance states are required.

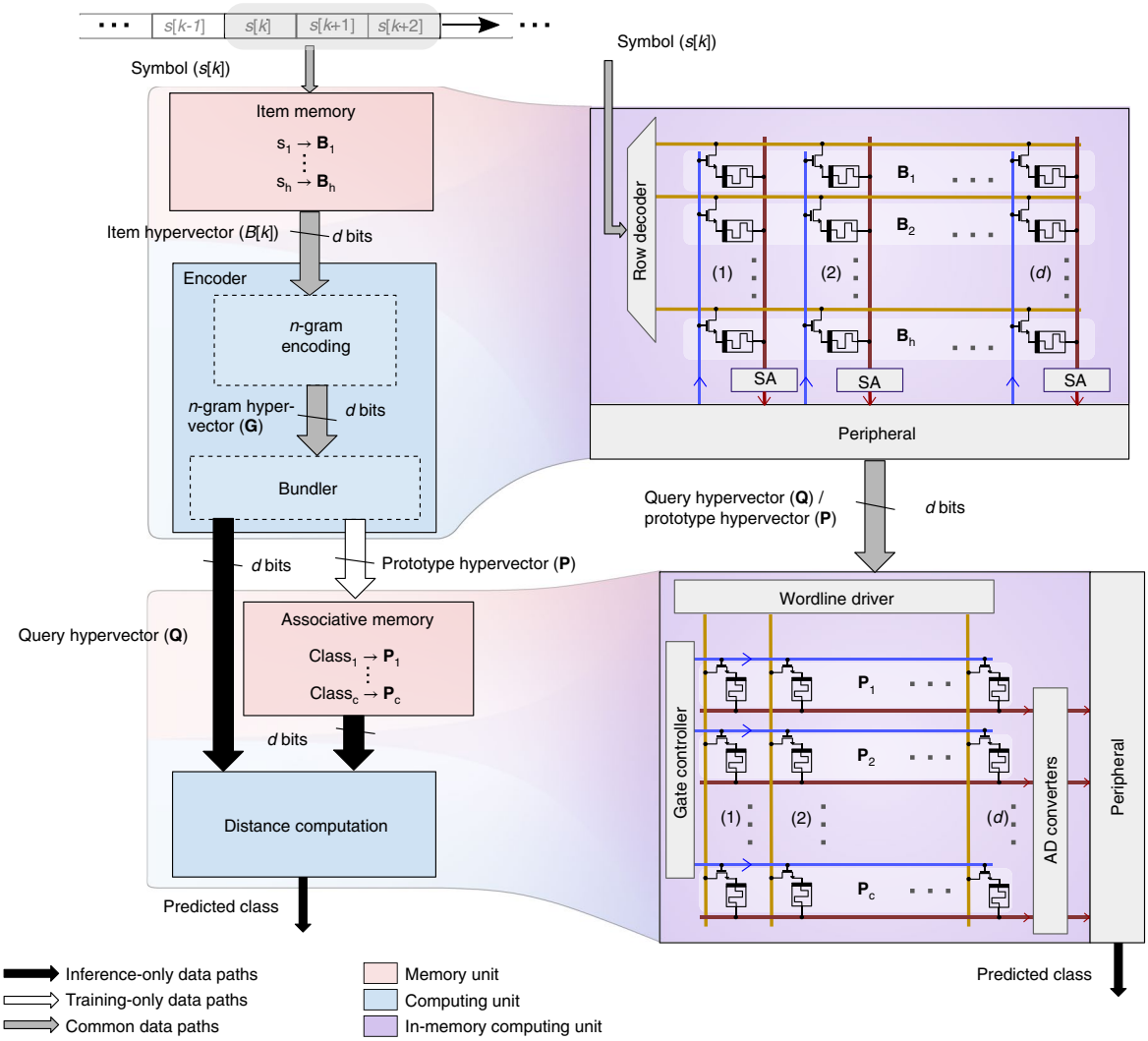

Fig. 1 | The concept of in-memory HDC. A schematic of the concept of in-memory HDC showing the essential steps associated with HDC (left) and how they are realized using in-memory computing (right). An item memory (IM) stores h, d-dimensional basis hypervectors that correspond to the symbols associated with a classification problem. During learning, based on a labelled training dataset, an encoder performs dimensionality, preserving mathematical manipulations on the basis hypervectors to produce c, d-dimensional prototype hypervectors that are stored in an AM. During classification, the same encoder generates a query hypervector based on a test example. Subsequently, an AM search is performed between the query hypervector and the hypervectors stored in the AM to determine the class to which the test example belongs. In in-memory HDC, both the IM and AM are mapped onto crossbar arrays of memristive devices. The mathematical operations associated with encoding and AM search are performed in place by exploiting in-memory read logic and dot-product operations, respectively. A dimensionality of $d = 10{,}000$ is used. SA, sense amplifier; AD converters, analog-to-digital converters.

In this work, we have used PCM technology[34,35], which operates by switching a phase-change material between amorphous (high resistivity) and crystalline (low resistivity) phases to implement binary data storage (see Methods). PCM has also been successfully employed in novel computing paradigms such as neuromorphic computing[36–40] and computational memory[20,22,41,42], which makes it a good candidate for realizing the in-memory HDC system.

In the remaining part of this Article, we will elaborate the detailed designs of the AM, the encoder and finally propose a complete in-memory HDC system that achieves a near-optimum trade-off between design complexity and output accuracy. The functionality of the in-memory HDC system will be validated through experiments using a prototype PCM chip fabricated in 90 nm CMOS technology (see Methods), and a complete system-level design implemented using 65 nm CMOS technology will be presented.

The AM search module

Classification involves an AM search between the prototype hypervectors and the query hypervector using a suitable similarity metric, such as the inverse Hamming distance (*invHamm*) computed from equation (2). Using associativity of addition operations, the expression in equation (2) can be decomposed into the addition of two dot-product terms as shown in equation (3):

$$\begin{aligned} \mathrm{Class}_{\mathrm{Pred}} &= \underset{i\in\{1,\ldots,c\}}{\arg\max}\, \mathbf{Q}\cdot\mathbf{P}_i + \overline{\mathbf{Q}}\cdot\overline{\mathbf{P}}_i \\ &\simeq \underset{i\in\{1,\ldots,c\}}{\arg\max}\, \mathbf{Q}\cdot\mathbf{P}_i \end{aligned} \tag{3}$$

where $\overline{\mathbf{Q}}$ denotes the logical complement of $\mathbf{Q}$. Because the operations associated with HDC ensure that both the query and prototype hypervectors have an almost equal number of zeros and ones,

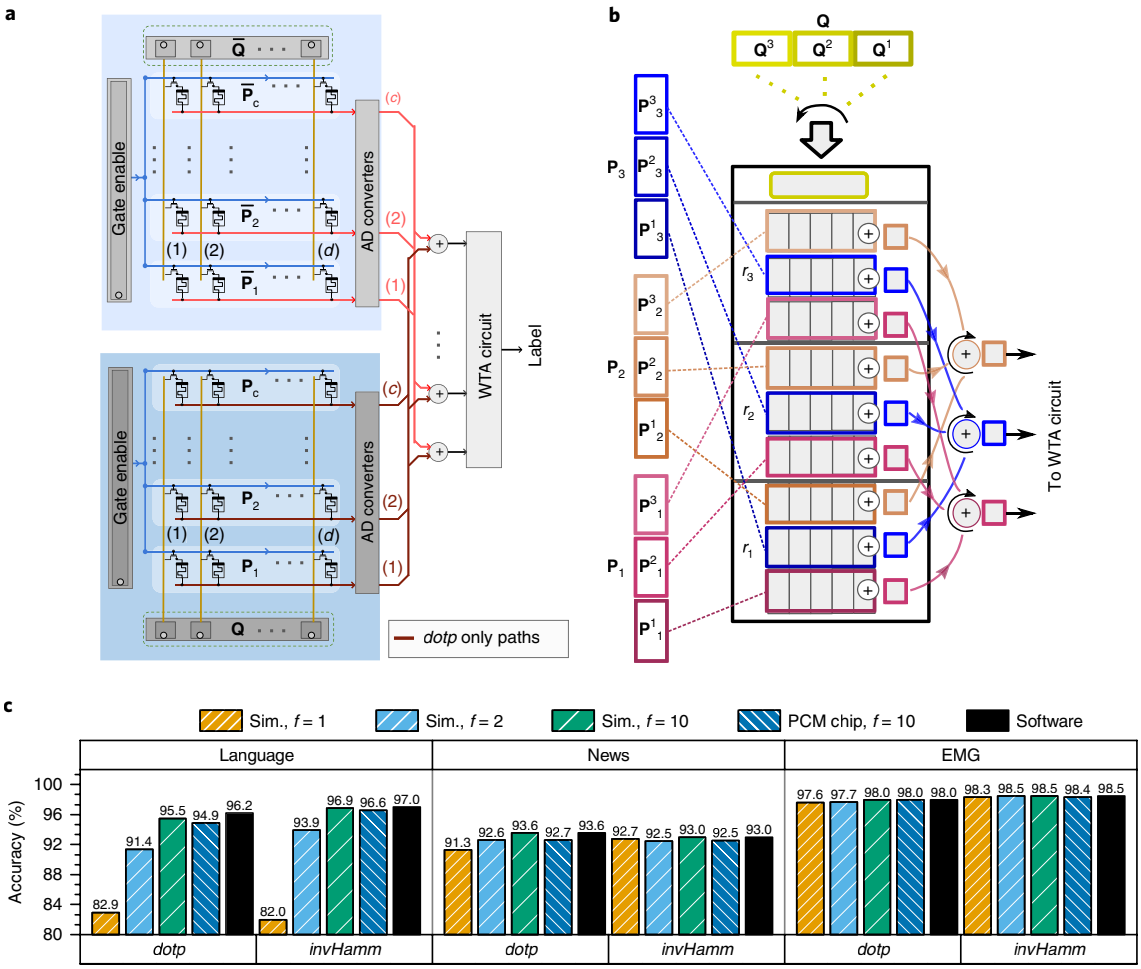

Fig. 2 | AM search. a, Schematic of the AM search architecture to compute the *invHamm* similarity metric. Two PCM crossbar arrays of *c* rows and *d* columns are employed. **b**, Schematic of the coarse-grained randomization strategy employed to counter the variations associated with the crystalline PCM state. **c**, Results of the classification task show that the experimental on-chip accuracy results compare favourably with the 10-partition simulation results and software baseline for both similarity metrics on the three datasets.

the dot product (*dotp*) $\arg\max_{i\in\{1,\ldots,c\}} \mathbf{Q}\cdot\mathbf{P}_i$ can also serve as a viable similarity metric.

To compute the *invHamm* similarity metric, two memristive crossbar arrays of *c* rows and *d* columns are required, as shown in Fig. 2a. The prototype hypervectors, $\mathbf{P}_i$, are programmed into one of the crossbar arrays as conductance states. Binary '1' components are programmed as crystalline states and binary '0' components are programmed as amorphous states. The complementary hypervectors $\overline{\mathbf{P}_i}$ are programmed in a similar manner into the second crossbar array. The query hypervector $\mathbf{Q}$ and its complement $\overline{\mathbf{Q}}$ are applied as voltage values along the wordlines of the respective crossbars. In accordance with Kirchoff's current law, the total current on the *i*th bitline will be equal to the dot product between the query hypervector and the *i*th prototype hypervector. The results of these in-memory dot-product operations from the two arrays are added in a pairwise manner using a digital adder circuitry in the periphery and are subsequently input to a winner-take-all (WTA) circuit that outputs a '1' only on the bitline corresponding to the class of maximum similarity value. When the *dotp* similarity metric is considered, only the crossbar encoding $\mathbf{P}_i$ is used and the array of adders in the periphery is eliminated, resulting in reduced hardware complexity.

Experiments were performed using a prototype PCM chip to evaluate the effectiveness of the proposed implementation on three common HDC benchmarks: language classification, news classification and hand gesture recognition from electromyography (EMG) signals (see Methods). These tasks demand a generic programmable architecture to support different numbers of inputs, classes and data types (see Methods). In the experiments, the prototype hypervectors (and their complements) are learned beforehand in software and are then programmed into the PCM devices on the chip. Inference is then performed with a software encoder and using equation (3) for the AM search, in which all multiplication operations are performed in the analog domain (by exploiting Ohm's law) on chip and the remaining operations are implemented in software (see Methods and Supplementary Note 1). The software encoder was employed to precisely assess the performance and accuracy of the AM search alone when implemented in hardware. The in-memory encoding scheme and its experimental validation are presented in sections 'The *n*-gram encoding module' and 'The complete in-memory HDC system'.

Although HDC is remarkably robust to random variability and device failures, deterministic spatial variations in the conductance values could pose a challenge. Unfortunately, in our prototype PCM chip, the conductance values associated with the crystalline state do exhibit a deterministic spatial variation (Supplementary Note 2). However, given the holographic nature of the hypervectors, this can be easily addressed by a random partitioning approach. We employed a coarse-grained randomization strategy, where the

idea is to segment the prototype hypervector and to place the resulting segments spatially distributed across the crossbar array (Fig. 2b). This helps all the components of prototype hypervectors to uniformly mitigate long-range variations. The proposed strategy involves dividing the crossbar array into f equal sized partitions (r_1, r_2, … r_f) and storing a $1/f$ segment of each of the prototype hypervectors ($\mathbf{P}_1$, $\mathbf{P}_2$, ..., $\mathbf{P}_c$) per partition. Here, f is called the 'partition factor' and it controls the granularity associated with the randomization. To match the segments of prototype hypervectors, the query vector is also split into equally sized subvectors $\mathbf{Q}^1$, $\mathbf{Q}^2$,..., $\mathbf{Q}^f$, which are input sequentially to the wordline drivers of the crossbar.

A statistical model that captures the spatiotemporal conductivity variations was used to evaluate the effectiveness of the coarse-grained randomized partitioning method (Supplementary Note 2). Simulations were carried out for different partition factors 1, 2 and 10 for the two similarity metrics *dotp* and *invHamm*, as shown in Fig. 2c. These results indicate that the classification accuracy increases with the number of partitions. For example, for language classification, the accuracy improves from 82.5% to 96% with *dotp* by randomizing with a partition factor of 10 instead of 1. The experimental on-chip accuracy (performed with a partition factor of 10) is close to the 10-partition simulation result and the software baseline for both similarity metrics on all three datasets. When the two similarity metrics are compared, *invHamm* provides slightly better accuracy for the same partition size, at the expense of almost doubled area and energy consumption. Therefore, for low-power applications, a good trade-off is the use of the *dotp* similarity metric with a partition factor of 10.

The *n*-gram encoding module

In this section, we will focus on the design of the n-gram encoding module. As described in the section 'The concept of in-memory HDC', one of the key operations associated with the encoder is calculation of the n-gram hypervector $\mathbf{G}$ given by equation (1). To find in-memory hardware-friendly operations, equation (1) is rewritten as the component-wise summation of 2^{n-1} minterms given by equation (4):

$$\mathbf{G} = \bigvee_{j=0}^{2^{n-1}-1} L_{1,j}(B[1]) \wedge \rho(L_{2,j}(B[2])) \wedge \ldots \wedge \rho^{n-1}(L_{n,j}(B[n])) \quad (4)$$

The operator $L_{k,j}$ is given by

$$L_{k,j}(B[k]) = B[k] \text{ if } (-1)^{Z(k,j)} = 1$$
$$= \overline{B[k]} \text{ otherwise}$$

where $Z(k,j) = \left\lfloor \frac{1}{2^k}(2j + 2^{k-1}) \right\rfloor$, $k \in \{1, 2, \ldots, n\}$ is the item hypervector index within an n-gram and $j \in \{0, 1, \ldots, 2^{n-1}-1\}$ is used to index minterms. The representation given by equation (4) can be mapped into memristive crossbar arrays where the bitwise AND ($\wedge$) function can be realized using an in-memory read logic operation. However, the number of minterms ($2^{n-1}-1$) rises exponentially with the size n of the n-gram, making the hardware computations costly. It is thus desirable to reduce the number of minterms and to use a fixed number of minterms independent of n.

Based on equation (4), we empirically obtained a 2-minterm encoding function for calculating the n-gram hypervector given by

$$\hat{\mathbf{G}} = (B[1] \wedge \rho(B[2]) \wedge \ldots \rho^{n-1}(B[n])) \vee (\overline{B[1]} \wedge \rho(\overline{B[2]}) \wedge \ldots \rho^{n-1}(\overline{B[n]})) \quad (5)$$

Encoding based on $\hat{\mathbf{G}}$ shows mostly functional equivalence with the ideal XNOR-based encoding scheme in certain key attributes such as similarity between the basis and prototype hypervectors (Supplementary Note 3). A schematic illustration of the corresponding n-gram encoding system is presented in Fig. 3a. The basis hypervectors are programmed on one of the crossbars and their complement vectors are programmed on the second. The component-wise logical AND operation between two hypervectors in equation (5) is realized in-memory by applying one of the hypervectors as the gate control lines of the crossbar, while selecting the wordline of the second hypervector. The result of the AND function from the crossbar is passed through an array of sense amplifiers to convert the analog values to binary values. The binary result is then stored in the minterm buffer, whose output is fed back as the gate controls by a single component shift to the right (left in the complementary crossbar). This operation approximates the permutation operation in equation (5) as a 1 bit right shift instead of a circular 1 bit shift. By performing these operations n times, it is possible to generate the n-gram. After n-gram encoding, the generated n-grams are accumulated and binarized with a threshold that depends on n (for details see Methods).

To test the effectiveness of the encoding scheme with in-memory computing, simulations were carried out using the PCM statistical model. The training was performed in software with the same encoding technique used thereafter for inference, and both the encoder and AM were implemented with modelled PCM crossbars for inference. The simulations were performed only on the language and news classification datasets, because for the EMG dataset the hypervectors used for the n-gram encoding are generated by a spatial encoding process and cannot be mapped entirely into a fixed IM of reasonable size. From the results presented in Fig. 3b, it is clear that the all-minterm approach to encoding provides the best classification accuracy in most configurations of AM, as expected. However, the 2-minterm-based encoding method yields a stable and, in some cases, particularly in the language dataset, a similar accuracy level to that of the all-minterm approach, while significantly reducing the hardware complexity. One of the perceived drawbacks of the 2-minterm approach is the increasing sparsity of the n-gram hypervectors with n. However, it can be shown that the dot-product similarity between the prototype hypervectors and hence the classification accuracy remain relatively unchanged due to the thresholding operation that depends on n (Supplementary Note 4).

The complete in-memory HDC system

In this section, the complete HDC system and the associated experimental results are presented. The proposed architecture comprises the 2-minterm encoder and *dotp* similarity metric with a partition factor of 10, as this provides the best trade-off between classification accuracy and hardware complexity (Supplementary Note 3). As shown in Fig. 4a, the proposed architecture has three PCM crossbar arrays—two with h rows and d columns and one with $c \times f$ rows and d/f columns, with $f = 10$.

The system includes several peripheral circuits—an index buffer, a minterm buffer and a bundler that reside inside the encoder—while the AM search module contains a sum buffer and a comparator circuit. The index buffer is located at the input of the IM to keep the indices of the symbols in the sequence and to feed them into the crossbar rows. The bundler accumulates the n-gram hypervectors to produce a sum hypervector. Once the threshold is applied on the sum hypervector, the result is a prototype hypervector during training or a query hypervector during inference. The controller inside the encoder module generates control signals according to the n-gram size and the length of the query sequence to allow different configurations of the encoder. During inference, one segment of the query hypervector at the output buffer of the encoder is fed at a time to the AM through an array of multiplexers so that only the corresponding partition is activated in the AM. Depending on the

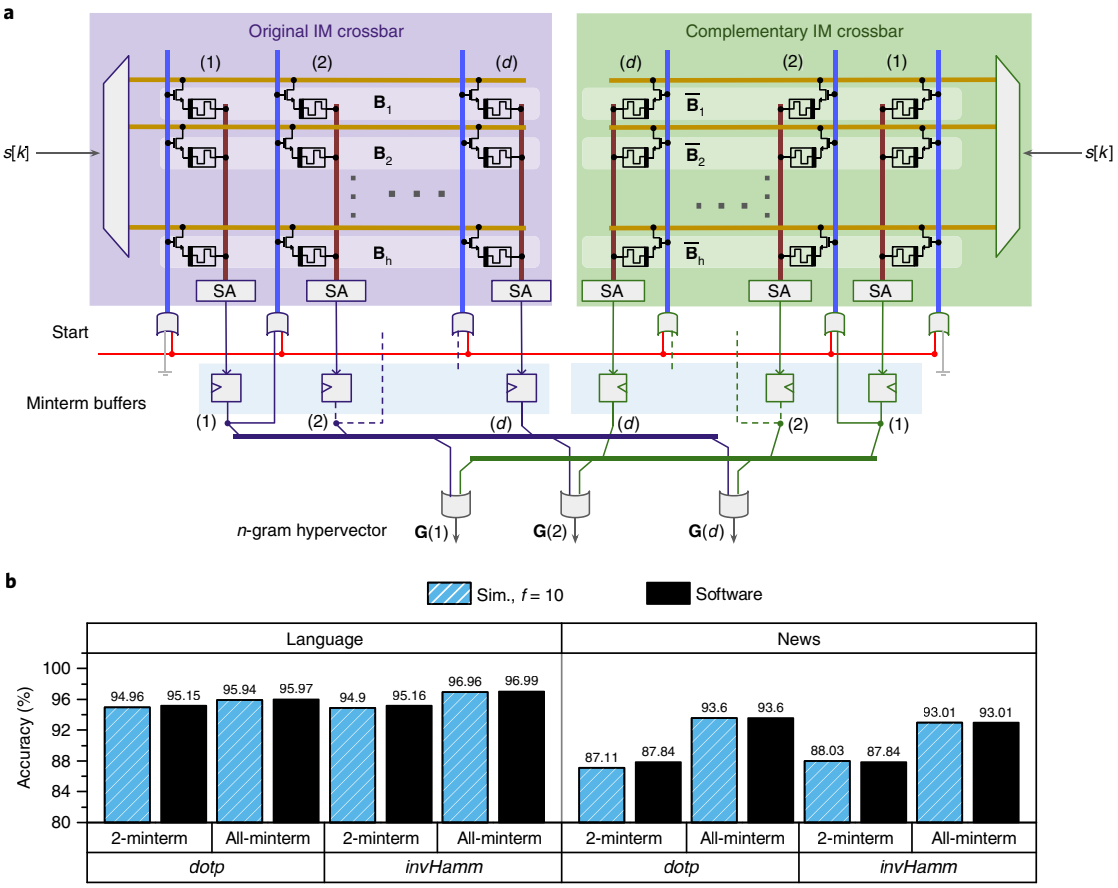

Fig. 3 | In-memory *n*-gram encoding based on 2-minterm. a, The basis hypervectors and their complements are mapped onto two crossbar arrays. Through a sequence of in-memory logical operations, the approximated *n*-gram **G**, as in equation (5), is generated. **b**, Classification results on the language (using $n = 4$) and news (using $n = 5$) datasets, showing the performance of the 2-minterm approximation compared with the all-minterm approach.

partition that is selected, the relevant gates are activated through a controller sitting inside the AM search module. Finally, the results in the sum buffer are sent through WTA circuitry to find the maximum index that provides the prediction.

To experimentally validate the functionality of the complete in-memory HDC architecture, we chose to implement the inference operation, which comprises both encoding (to generate the query hypervectors) and AM search (Supplementary Video 1). For faster experiments, we trained our HDC model in software using the 2-minterm approximate encoding method described in the section 'The *n*-gram encoding module', which could be performed as well with our proposed in-memory HDC architecture. This software generates the hypervectors for AM from a given dataset. Subsequently, the components of all hypervectors of both IM and AM were programmed on individual hardware PCM devices, and the inference operation was implemented leveraging the two in-memory computing primitives (for both 2-minterm encoding and the AM search) using the prototype PCM chip (see Methods and Supplementary Note 1). Figure 4b summarizes the accuracy results with software, the PCM statistical model and the on-chip experiment for the language and news classification benchmarks. Compared with the previous experiment, where only AM was contained on-chip, the full chip experiment results show a similar accuracy level, indicating the minimal effect on accuracy when porting the IM into PCM devices with in-memory *n*-gram encoding. Furthermore, the accuracy level reported in this experiment is close to the accuracy reported with the software for the same parametric configuration of the HD inference model.

Finally, to benchmark the performance of the system in terms of energy consumption, the digital submodules in the system-level architecture (marked with dotted boundaries in Fig. 4a) that fall outside the PCM crossbars arrays were synthesized using 65 nm CMOS technology. The synthesis results for these modules were combined with the performance characteristics of PCM crossbar arrays to evaluate the energy, area and throughput of the full system (see Methods). Furthermore, PCM crossbar sections were implemented in CMOS distributed standard cell registers with associated multiplier–adder tree logic and binding logic for AM and IM, respectively, to construct a complete CMOS HD processor to compare with the proposed PCM crossbar-based architecture.

A comparison of the performance between the all-CMOS approach and the PCM crossbar-based approach is presented in Table 1. A 6.01× improvement in total energy efficiency and 3.74× reduction in area is obtained with the introduction of the PCM crossbar modules. The encoder's energy expense for processing a query reduces by a factor of 3.50 with the PCM crossbar implementation, whereas that of the AM search module reduces by a factor of 117.5. However, these efficiency factors are partially masked by the CMOS peripheral circuitry that is common to both implementations, specifically that in the encoder module, which accounts for the majority of its energy consumption. When peripheral circuits are ignored and only the parts of the design that are exclusive to each approach are directly compared to each other, 14.4× and 334× energy savings and 24.5× and 31.9× area savings are obtained for the encoder and AM search module, respectively. It remains part of future work to investigate methods in which peripheral modules

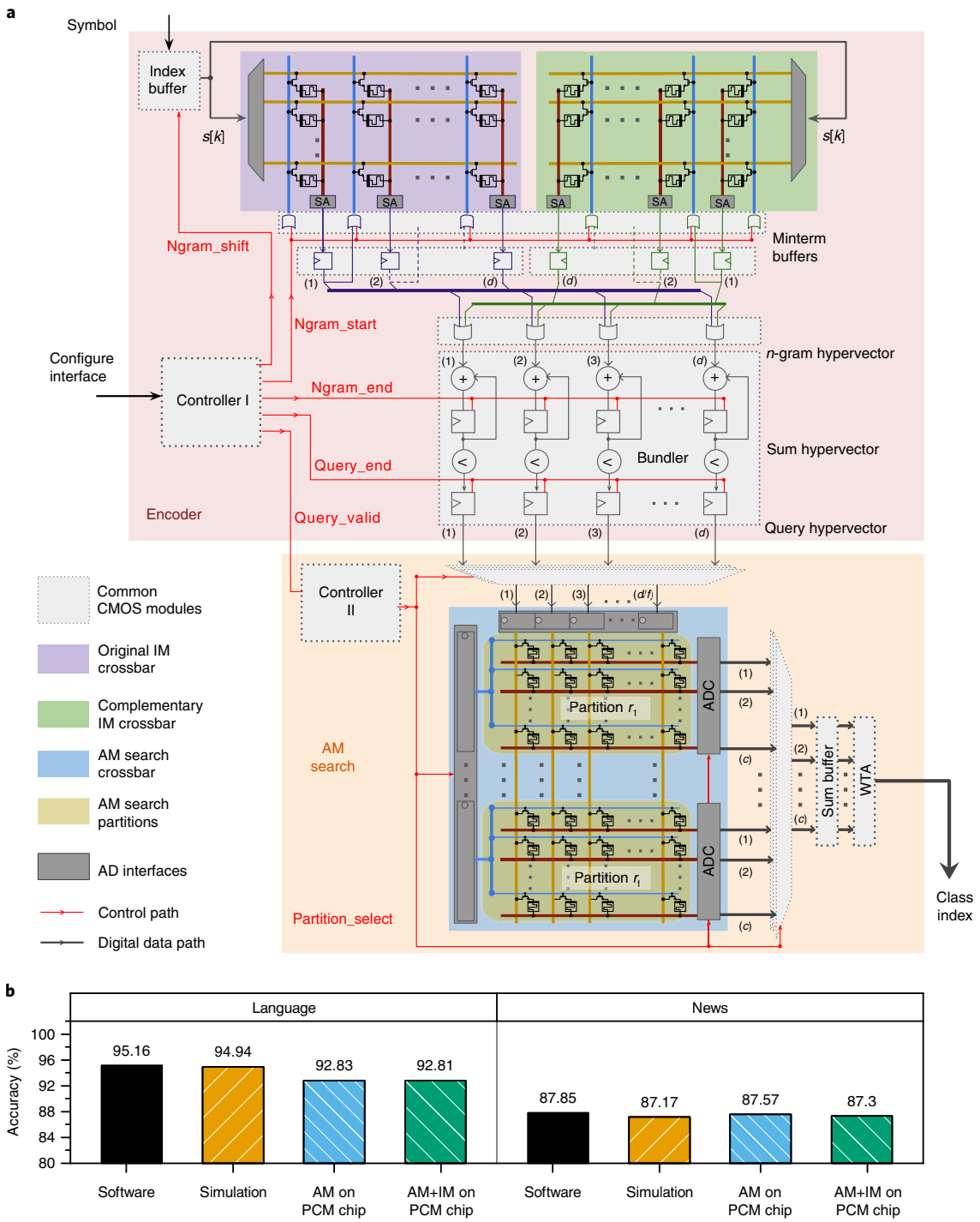

Fig. 4 | The complete in-memory HDC system. a, Schematic of the architecture, showing the 2-minterm encoder and AM search engine employing the *dotp* metric. **b**, The classification accuracy results on the news and language datasets, where both the encoding and AM search are performed in software, simulated using the PCM model and are experimentally realized on the chip.

are designed more energy efficiently so that the overall system efficiency can be improved further.

Conclusions

HDC is a brain-inspired computational framework that is particularly well suited for the emerging computational paradigm of in-memory computing. We have reported a complete in-memory HDC system whose two main components are an encoder and an AM search engine. The main computations are performed in-memory with logical and dot-product operations on memristive devices. Due to the inherent robustness of HDC to errors, it was possible to approximate the mathematical operations associated with HDC to make it suitable for hardware implementation, and to use analog in-memory computing without significantly degrading the output accuracy. Our architecture is programmable to support different hypervector representations, dimensionality and number of input symbols and output classes to accommodate a variety of applications.

Table 1 | Performance comparison between a dedicated all-CMOS implementation and in-memory HDC with PCM crossbars

	All-CMOS			PCM crossbar based		
	Encoder	AM search	Total	Encoder	AM search	Total
Energy						
Average energy per query (nJ)	1,474	1,110	2,584	420.8	9.44	430.3
Improvement				3.50×	117.5×	6.01×
Exclusive modules avg. energy per query (nJ)	1,132	1,104	2,236	78.60	3.30	81.90
Improvement				14.40×	334.6×	27.30×
Area						
Total area (mm^2)	4.77	2.99	7.76	1.39	0.68	2.07
Improvement				3.43×	4.38×	3.74×
Exclusive modules area (mm^2)	3.53	2.38	5.91	0.14	0.075	0.22
Improvement				24.57×	31.94×	27.09×

Hardware/software experiments using a prototype PCM chip delivered accuracies comparable to software baselines on language and news classification benchmarks with 10,000-dimensional hypervectors. These experiments used hardware PCM devices to implement both in-memory encoding and AM search, thus demonstrating the hardware functionality of all the operations involved in a generic HDC processor for learning and inference. A comparative study performed against a system-level design implemented using 65 nm CMOS technology showed that the in-memory HDC approach could result in more than 6× end-to-end savings in energy. By designing more energy-efficient peripheral circuits and with the potential of scaling PCM devices to nanoscale dimensions[43], these gains could increase several-fold. The in-memory HDC concept is also applicable to other types of memristive device based on ionic drift[44] and magnetoresistance[45]. Future work will focus on taking in-memory HDC beyond learning and classification to perform advanced cognitive tasks alongside data compression and retrieval on dense storage devices, as well as building more power-efficient peripheral hardware to harness the best of in-memory computing.

Methods

PCM-based hardware platform. The experimental hardware platform is built around a prototype PCM chip that contains PCM cells based on doped-$Ge_2Sb_2Te_5$ (d-GST) that are integrated into the prototype chip in 90 nm CMOS baseline technology. In addition to the PCM cells, the prototype chip integrates the circuitry for cell addressing, on-chip ADCs for cell readout and voltage- or current-mode cell programming. The experimental platform comprises the following main units:

- a high-performance analog-front-end (AFE) board that contains digital-to-analog converters (DACs) along with discrete electronics, such as power supplies, voltage and current reference sources
- a field-programmable gate array (FPGA) board that implements the data acquisition and digital logic to interface with the PCM device under test and with all the electronics of the AFE board
- a second FPGA board with an embedded processor and Ethernet connection that implements the overall system control and data management as well as the interface with the host computer

The prototype chip[46] contains three million PCM cells, as well as the CMOS circuitry to address, program and read out any of these three million cells. In the PCM devices used for experimentation, two 240-nm-wide access transistors were used in parallel per PCM element (cell size of 50 F^2). The PCM array is organized as a matrix of 512 wordlines and 2,048 bitlines. The PCM cells were integrated into the chip in 90 nm CMOS technology using the keyhole process[47]. The bottom electrode had a radius of ~20 nm and length of ~65 nm. The phase-change material was ~100 nm thick and extended to the top electrode, whose radius was ~100 nm. The selection of one PCM cell was performed by serially addressing a wordline and a bitline. The addresses were decoded and drove the wordline driver and the bitline multiplexer. The single selected cell could be programmed by forcing a current through the bitline with a voltage-controlled current source. It could also be read by an 8 bit on-chip ADC. To read a PCM cell, the selected bitline was biased to a constant voltage of 300 mV by a voltage regulator via a voltage V_{read} generated via an off-chip DAC. The sensed current, I_{read}, was integrated by a capacitor, and the resulting voltage was then digitized by the on-chip 8 bit cyclic ADC. The total time of one read was 1 µs. To program a PCM cell, a voltage V_{prog} generated off chip was converted on chip into a programming current, I_{prog}. This current was then mirrored into the selected bitline for the desired duration of the programming pulse. The pulse used to program the PCM to the amorphous state (reset) was a box-type rectangular pulse with duration of 400 ns and amplitude of 450 µA. The pulse used to program the PCM to the crystalline state (set) was a ramp-down pulse with total duration of ~12 µs. The access-device gate voltage (wordline voltage) was kept high at 2.75 V during the programming pulses. These programming conditions were optimized to achieve the highest on/off ratio and to minimize device-to-device variability for binary storage.

Datasets to evaluate in-memory HDC. We targeted three highly relevant learning and classification tasks to evaluate the proposed in-memory HDC architecture. These tasks demand a generic programmable architecture to support different numbers of inputs, classes and data types, as shown in Extended Data Table 1. In the following, we describe these tasks that are used to benchmark the performance of in-memory HDC in terms of classification accuracy.

1. Language classification: in this task, HDC is applied to classify raw text composed of Latin characters into their respective language[48]. The training texts are taken from the Wortschatz Corpora[49], where large numbers of sentences (about a million bytes of text) are available for 22 European languages. Another independent dataset, Europarl Parallel Corpus[50], with 1,000 sentences per language, is used as the test dataset for the classification. The former database is used for training 22 prototype hypervectors for each of the languages while the latter is used to run inference on the trained HDC model. For subsequent simulations and experiments with the language dataset, we use dimensionality $d = 10{,}000$ and n-gram size $n = 4$. We use an IM of 27 symbols, representing the 26 letters of the Latin alphabet plus a whitespace character. Training is performed using the entire training dataset, containing a labelled text of 120,000–240,000 words per language. For inference, a query is composed of a single sentence of the test dataset, so, in total, 1,000 queries per language are used.
2. News classification: the news dataset comprises a database of *Reuters* news articles, subjected to a lightweight pre-processing step, covering eight different news genres[51]. The pre-processing step removes frequent 'stop' words and words with fewer than three letters. The training set has 5,400+ documents, while the testing set contains 2,100+ documents. For subsequent simulations and experiments with the news dataset, we use dimensionality $d = 10{,}000$ and n-gram size $n = 5$, as suggested in ref. [18]. Similar to the language task, we use an IM of 27 symbols, representing the 26 letters of the Latin alphabet plus a whitespace character. Training is performed using the entire training dataset, where all labelled documents pertaining to the same class are merged into a single text. This merged text contains 8,000–200,000 words per class. For inference, a query is composed of a single document of the test dataset.
3. Hand gesture recognition from EMG signals: in this task, we focus on use of HDC in a smart prosthetic application, namely hand gesture recognition from a stream of EMG signals. A database[52] that provides EMG samples recorded from four channels covering the forearm muscles is used for this benchmark. Each channel data is quantized into 22 intensity levels of electric potential. The sampling frequency of the EMG signal is 500 Hz.
A label is provided for each time sample. The label varies from one to five corresponding to five classes of performed gestures. This dataset is used to train an HDC model to detect hand gestures of a single subject. For training on the

EMG dataset, a spatial encoding scheme is first employed to fuse data from the four channels so the IM has four discrete symbols, and it is paired with a continuous IM to jointly map the 22 intensity levels per channel (details about the encoding procedure for the EMG dataset are provided in Supplementary Note 5). The pairing of IM and continuous IM allows a combination of orthogonal mapping with distance proportionality mapping. The spatial encoding creates one hypervector per time sample.
A temporal encoding step is then performed, whereby n consecutive spatially encoded hypervectors are combined into an n-gram. For the subsequent simulations and experiments with the EMG dataset we use dimensionality $d = 10{,}000$ and n-gram size $n = 5$. Training and inference are performed using the same EMG channel signals from the same subject, but on non-overlapping sections of recording. The recording used for training contains 1,280 time samples after downsampling by a factor of 175. For inference, 780 queries are generated from the rest of the recording, where each query contains five time samples captured with the same downsampling factor.

For the different tasks, Extended Data Table 1 provides details on the desired hypervector representations and different hyperparameters including the dimension of hypervectors, the alphabet size, the n-gram size and the number of classes. For the EMG dataset, the hypervectors for the encoding operation are drawn by binding items from a pair of IM and continuous IM (Supplementary Note 5). In hardware implementation of the in-memory HDC, the IM and AM may be distributed into multiple narrower crossbars in case electrical/physical limitations arise.

Coarse-grained randomization. The programming methodology followed to achieve the coarse-grained randomized partitioning in the memristive crossbar for the AM search is explained in the following steps. First, we split all prototype hypervectors ($\mathbf{P}_1$, $\mathbf{P}_2$,..., $\mathbf{P}_c$) into f subvectors of equal length, where f is the partition factor. For example, subvectors from the prototype hypervector of the first class are denoted as ($\mathbf{P}_1^1$, $\mathbf{P}_1^2$, ..., $\mathbf{P}_1^f$). The crossbar array is then divided into f equally sized partitions (r_1, r_2, ..., r_f). Each partition must contain d/f rows and c columns. A random permutation e of numbers 1 to c is then selected. Next, the first subvector from each class ($\mathbf{P}_1^1$, $\mathbf{P}_2^1$, ..., $\mathbf{P}_c^1$) is programmed into the first partition r_1 such that each subvector fits to a column in the crossbar partition. The order of programming of subvectors into the columns in the partition is determined by the previously selected random permutation e. The above steps must be repeated to program all the remaining partitions (r_2, r_3, ..., r_f).

The methodology followed in feeding query vectors during inference is detailed in the following steps. First, we split query hypervector $\mathbf{Q}$ into f subvectors ($\mathbf{Q}^1$, $\mathbf{Q}^2$,...,$\mathbf{Q}^f$) of equal length. We then translate $\mathbf{Q}^i$ component values into voltage levels and apply them onto the wordline drivers in the crossbar array. Bitlines corresponding to the partition r_i are enabled. Depending on the belonging class, the partial dot products are then collected onto the respective destination in a sum buffer through AD converters at the end of the r_i partition of the array. This procedure is repeated for each partition r_i. Classwise partial dot products are accumulated together in each iteration and updated in the sum buffer. After the fth iteration, full dot-product values are ready in the sum buffer. The results are then compared against each other using a WTA circuit to find the maximum value to assign its index as the predicted class.

Experiments on AM search. To obtain the prototype hypervectors used for the AM search, training with HDC is first performed in software on the three datasets described in the section 'Datasets to evaluate in-memory HDC'. For the language and news datasets, XOR-based encoding (see section 'The concept of in-memory HDC') is used with an n-gram size of $n = 4$ and $n = 5$, respectively. For the EMG dataset, an initial spatial encoding step creates one hypervector per time sample. A temporal encoding step is then performed, whereby n consecutive spatially encoded hypervectors are combined into an n-gram with XOR-based encoding and $n = 5$. The detailed encoding procedure for the EMG dataset is explained in Supplementary Note 5.

Once training is performed, the prototype hypervectors are programmed on the prototype PCM chip. In the experiment conducted with *invHamm* as the similarity metric, $d \times c \times 2$ devices on the PCM prototype chip are allocated. Each device in the first half of the address range (from 1 to $d \times c$) is programmed with a component of a prototype hypervector $\mathbf{P}_i$, where $i = 1, \ldots, c$. Devices in the second half of the array are programmed with components of the complementary prototype hypervectors. The exact programming order is determined by the partition factor (f) employed in the coarse-grained randomized partitioning scheme. For $f = 10$ used in the experiment, devices from the first address up to the $1{,}000 \times c$th address are programmed with the content of the first partition, that is, the first segment of each prototype hypervector. The second set of $1{,}000 \times c$ addresses is programmed with content of the second partition, and so on. As the hypervector components are binary, devices mapped to the logical 1 components and devices mapped to logical 0 components are programmed to the maximum (~20 μS) and minimum conductance (~0 μS) levels, respectively. The devices are programmed in a single shot (no iterative program-and-verify algorithm is used) with a single reset/set pulse for minimum/maximum conductance devices.

Once the programming phase is completed, the queries from the testing set of a given task are encoded. Only for the experiments in section '3The AM search module', the query hypervectors are generated using the same software HD encoder used for training. In the experiments of section 'The complete in-memory HDC system', the query hypervectors are generated with in-memory encoding using the prototype PCM chip as described in the section 'Experiments on the complete in-memory HDC system'.

The AM search on a given query hypervector is performed using the prototype PCM chip as follows. The components of the query hypervector carrying a value 1 trigger a read (300 mV applied voltage) on the devices storing the corresponding components of prototype hypervectors, thus realizing the analog multiplications through Ohm's law of the in-memory dot-product operation. The same procedure is performed with the complementary query hypervector on the devices storing complementary prototype hypervectors. The resulting current values are digitized via the on-chip ADC, transferred to the host computer and classwise summed up in software according to the predetermined partition order to obtain classwise similarity values (Supplementary Note 1). The class with the highest similarity is assigned as the predicted class for the given query. For experiments with *dotp* as the similarity metric, the devices attributed to complementary prototype hypervectors are not read when forming the classwise aggregate.

More details on the 2-minterm encoder. To generate an n-gram hypervector in n cycles, the crossbar is operated using the following procedure. During the first cycle, n-gram encoding is initiated by asserting the 'start' signal while choosing the index of the nth symbol $s[n]$. This enables all the gate lines in both crossbar arrays and the wordline corresponding to $s[n]$ to be activated. The current released onto the bitlines passed through the sense amplifiers should ideally match the logic levels of $B[n]$ in first array and $\overline{B[n]}$ in the second array. The two 'minterm buffers' downstream of the sense amplifier arrays register the two hypervectors by the end of the first cycle. During subsequent jth ($1 < j \leq n$) cycles, the gate lines are driven by the right-shifted version of the incumbent values on the minterm buffers—effectively implementing permutation—while row decoders are fed with symbol $s[n-j+1]$ (the left shift is used for the second crossbar). This ensures that the output currents on the bitlines correspond to the component-wise logical AND between the permuted minterm buffer values and the next basis hypervector $B[n-j]$ (complement for the second array). The expression for the value stored on the left-side minterm buffers at the end of jth cycle is given by $\prod_{k=1}^{j} \rho^{j-k} B[n-k+1]$. The product of the complementary hypervectors $\prod_{k=1}^{j} \rho^{j-k} \overline{B[n-k+1]}$ is stored in the right-side minterm buffers. At the end of the nth cycle, the two minterms are available in the minterm buffers. The elements in the minterm buffers are passed onto the OR gate array following the minterm buffers (shown in Fig. 3a), such that inputs to the array have matching indices from the two minterm vectors. At this point, the output of the OR gate array reflects the desired n-gram hypervector from 2-minterm n-gram encoding. After n-gram encoding, the generated n-grams are accumulated and binarized. In the hardware implementation, this step is realized inside the bundler module shown in Fig. 4a. The threshold applied to binarize the sum hypervector components is given by

$$l \times \left(\frac{1}{2^{n - \log(m)}} \right)$$

where l is the length of the sequence, n is the n-gram size and m is the number of minterms used for the binding operation in the encoder (for example, $m = 2$ for 2-minterm encoder).

Experiments on the complete in-memory HDC system. For the experiments concerning the complete in-memory HDC system, training with HDC is first performed in software on the language and news datasets. 2-minterm encoding (equation (5)) is used with n-gram sizes of $n = 4$ and $n = 5$, respectively.

After training is performed, $h \times d \times 2$ devices are allocated on the PCM chip for storing IM and complementary IM in addition to $d \times c$ devices allocated for AM. The IM and complementary IM hypervectors are programmed on PCM devices in a single shot with reset/set pulses for logical 0/1 components. The prototype hypervectors of the AM are programmed as described in the section 'Experiments on AM search', with the exception that the complementary prototype hypervectors are not programmed because *dotp* is used as the similarity metric.

During inference, for every query to be encoded, the IM and complementary IM are read from the prototype PCM chip. In-memory read logic (AND) is performed by thresholding the read current values from the on-chip ADC in software to emulate the sense amplifiers of the eventual proposed hardware at each step of the 2-minterm n-gram encoding process (Supplementary Note 1). The other operations involved in the encoder that are not supported by the prototype PCM chip, such as the 1 bit right-shift permutation, storing of the intermediate results in the minterm buffers, ORing the results of the original and complementary minterm buffers, and the bundling of n-gram hypervectors are implemented in software. Once the encoding of the query hypervector is completed, the AM search is carried out on that query hypervector as specified in the section 'Experiments on AM search' with *dotp* as the similarity metric.

Performance, energy estimation and comparison. To evaluate and benchmark the energy efficiency of the proposed architecture, a cycle-accurate register transfer level (RTL) model of a complete CMOS design that has equivalent throughput to

that of the proposed in-memory HDC system architecture has been developed (Supplementary Note 6). A testbench infrastructure is then built to verify the correct behaviour of the model. Once the behaviour is verified, the RTL model is synthesized in a UMC 65 nm technology node using a Synopsys Design Compiler. Owing to the limitations in the electronic design automation (EDA) tools used for synthesizing the CMOS-based HDC, dimensionality d had to be limited to 2,000. The post-synthesis netlist is then verified using the same stimulus vectors applied during behavioural simulation. During post-synthesis netlist simulation, the design is clocked at a frequency of 440 MHz to create a switching activity file in value change dump (VCD) format for inference of 100 language classification queries. Then, the energy estimation for the CMOS modules is performed by converting average power values reported by Synopsys Primetime, which takes the netlist and the activity file from the previous steps as the inputs. A typical operating condition with voltage 1.2 V and temperature 25 °C is set as the corner for energy estimation of the CMOS system. Further energy and area results were obtained for d values of 100, 500, 1,000 in addition to 2,000. The results were then extrapolated to derive the energy and area estimates for dimensionality $d = 10{,}000$ to obtain a fair comparison with the in-memory HDC system.

The energy/area of the proposed in-memory HDC system architecture is obtained by adding the energy/area of the modules that are common with the full CMOS design described above, together with the energy of the PCM crossbars and the analog/digital peripheral circuits exclusive to the in-memory HDC architecture. Parameters based on the prototype PCM chip in the 90 nm technology used in the experiments are taken as the basis for the PCM-exclusive energy/area estimation. The parameters of the sense amplifiers that are not present in the PCM hardware platform but present in the proposed in-memory HD encoder are taken from the 65 nm current latched sense amplifier presented by Chandoke and others[53]. The area of the current latched sense amplifier was estimated by scaling the area of the six-transistor SRAM cell in IBM 65 nm technology (0.54 μm^2) according to the number of transistors present in the sense amplifier (19). The parameters used for the PCM crossbars energy estimation are shown in Extended Data Table 2.

Data availability

The data that support the plots within this paper and other findings of this study are available from the corresponding author upon reasonable request.

Received: 13 November 2019; Accepted: 7 April 2020;
Published online: 1 June 2020

References

1. Kanerva, P. Sparse Distributed Memory (MIT Press, 1988).
2. Kanerva, P. Hyperdimensional computing: an introduction to computing in distributed representation with high-dimensional random vectors. *Cogn. Comput.* **1**, 139–159 (2009).
3. Kanerva, P., Kristoferson, J. & Holst, A. Random indexing of text samples for latent semantic analysis. In *Proceedings of the Annual Meeting of the Cognitive Science Society* Vol. 22 (Cognitive Science Society, 2000).
4. Rahimi, A., Kanerva, P., Benini, L. & Rabaey, J. M. Efficient biosignal processing using hyperdimensional computing: network templates for combined learning and classification of ExG signals. *Proc. IEEE* **107**, 123–143 (2019).
5. Burrello, A., Cavigelli, L., Schindler, K., Benini, L. & Rahimi, A. Laelaps: an energy-efficient seizure detection algorithm from long-term human iEEG recordings without false alarms. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition* (*DATE*) 752–757 (IEEE, 2019).
6. Räsänen, O. J. & Saarinen, J. P. Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns. *IEEE Trans. Neural Netw. Learn. Syst.* **27**, 1878–1889 (2015).
7. Kleyko, D. & Osipov, E. Brain-like classifier of temporal patterns. In *Proceedings of the International Conference on Computer and Information Sciences* (*ICCOINS*) 1–6 (IEEE, 2014).
8. Kleyko, D., Osipov, E., Papakonstantinou, N. & Vyatkin, V. Hyperdimensional computing in industrial systems: the use-case of distributed fault isolation in a power plant. *IEEE Access* **6**, 30766–30777 (2018).
9. Chang, E., Rahimi, A., Benini, L. & Wu, A. A. Hyperdimensional computing-based multimodality emotion recognition with physiological signals. In *Proceedings of the IEEE International Conference on Artificial Intelligence Circuits and Systems* (*AICAS*) 137–141 (IEEE, 2019).
10. Mitrokhin, A., Sutor, P., Fermüller, C. & Aloimonos, Y. Learning sensorimotor control with neuromorphic sensors: toward hyperdimensional active perception. *Sci. Robot.* **4**, eaaw6736 (2019).
11. Montagna, F., Rahimi, A., Benatti, S., Rossi, D. & Benini, L. PULP-HD: accelerating brain-inspired high-dimensional computing on a parallel ultra-low power platform. In *Proceedings of the 55th Annual Design Automation Conference* DAC 2018, 111:1–111:6 (ACM, 2018).
12. Emruli, B., Gayler, R. W. & Sandin, F. Analogical mapping and inference with binary spatter codes and sparse distributed memory. In *Proceedings of the International Joint Conference on Neural Networks* (*IJCNN*) 1–8 (IEEE, 2013).
13. Kleyko, D., Osipov, E., Gayler, R. W., Khan, A. I. & Dyer, A. G. Imitation of honey bees' concept learning processes using vector symbolic architectures. *Biol. Inspired Cogn. Architectures* **14**, 57–72 (2015).
14. Slipchenko, S. V. & Rachkovskij, D. A. Analogical mapping using similarity of binary distributed representations. *Inf. Theories Appl.* **16**, 269–290 (2009).
15. Bandaragoda, T. et al. Trajectory clustering of road traffic in urban environments using incremental machine learning in combination with hyperdimensional computing. In *Proceedings of the IEEE Intelligent Transportation Systems Conference* (*ITSC*) 1664–1670 (IEEE, 2019).
16. Osipov, E., Kleyko, D. & Legalov, A. Associative synthesis of finite state automata model of a controlled object with hyperdimensional computing. In *Proceedings of the Annual Conference of the IEEE Industrial Electronics Society* 3276–3281 (IEEE, 2017).
17. Kleyko, D., Frady, E. P. & Osipov, E. Integer echo state networks: hyperdimensional reservoir computing. Preprint at https://arxiv.org/pdf/1706.00280.pdf (2017).
18. Rahimi, A. et al. High-dimensional computing as a nanoscalable paradigm. *IEEE Trans. Circuits Syst. I Regular Papers* **64**, 2508–2521 (2017).
19. Yang, J. J., Strukov, D. B. & Stewart, D. R. Memristive devices for computing. *Nat. Nanotechnol.* **8**, 13–24 (2013).
20. Sebastian, A. et al. Temporal correlation detection using computational phase-change memory. *Nat. Commun.* **8**, 1115 (2017).
21. Zidan, M. A., Strachan, J. P. & Lu, W. D. The future of electronics based on memristive systems. *Nat. Electron.* **1**, 22–29 (2018).
22. Ielmini, D. & Wong, H.-S. P. In-memory computing with resistive switching devices. *Nat. Electron.* **1**, 333–343 (2018).
23. Sebastian, A., Le Gallo, M., Khaddam-Aljameh, R. & Eleftheriou, E. Memory devices and applications for in-memory computing. *Nat. Nanotechnol.* https://doi.org/10.1038/s41565-020-0655-z (2020).
24. Li, H. et al. Hyperdimensional computing with 3D VRRAM in-memory kernels: device-architecture co-design for energy-efficient, error-resilient language recognition. In *Proceedings of the IEEE International Electron Devices Meeting* (*IEDM*) 16.1.1–16.1.4 (IEEE, 2016).
25. Li, H., Wu, T. F., Mitra, S. & Wong, H. S. P. Device-architecture co-design for hyperdimensional computing with 3D vertical resistive switching random access memory (3D VRRAM). In *Proceedings of the International Symposium on VLSI Technology, Systems and Application* (*VLSI-TSA*) 1–2 (IEEE, 2017).
26. Wu, T. F. et al. Brain-inspired computing exploiting carbon nanotube FETs and resistive RAM: hyperdimensional computing case study. In *Proceedings of the International Solid State Circuits Conference* (*ISSCC*) 492–494 (IEEE, 2018).
27. Kanerva, P. Binary spatter-coding of ordered k-tuples. In *Proceedings of the International Conference on Artificial Neural Networks* (*ICANN*), Vol. 1112, 869–873 (Lecture Notes in Computer Science, Springer, 1996).
28. Joshi, A., Halseth, J. T. & Kanerva, P. Language geometry using random indexing. In *Proceedings of the International Symposium on Quantum Interaction* 265–274 (Springer, 2016).
29. Chua, L. Resistance switching memories are memristors. *Appl. Phys. A* **102**, 765–783 (2011).
30. Wong, H.-S. P. & Salahuddin, S. Memory leads the way to better computing. *Nat. Nanotechnol.* **10**, 191–194 (2015).
31. Borghetti, J. et al. 'Memristive' switches enable 'stateful' logic operations via material implication. *Nature* **464**, 873–876 (2010).
32. Kvatinsky, S. et al. Magic—memristor-aided logic. *IEEE Trans. Circuits Syst II Express Briefs* **61**, 895–899 (2014).
33. Shen, W. et al. Stateful logic operations in one-transistor-one-resistor resistive random access memory array. *Electron Device Lett.* **40**, 1538–1541 (2019).
34. Wong, H.-S. P. et al. Phase change memory. *Proc. IEEE* **98**, 2201–2227 (2010).
35. Burr, G. W. et al. Recent progress in phase-change memory technology. *IEEE J. Emerging Selected Topics Circuits Syst.* **6**, 146–162 (2016).
36. Kuzum, D., Jeyasingh, R. G., Lee, B. & Wong, H.-S. P. Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing. *Nano Lett.* **12**, 2179–2186 (2011).
37. Tuma, T., Pantazi, A., Le Gallo, M., Sebastian, A. & Eleftheriou, E. Stochastic phase-change neurons. *Nat. Nanotechnol.* **11**, 693–699 (2016).
38. Boybat, I. et al. Neuromorphic computing with multi-memristive synapses. *Nat. Commun.* **9**, 2514 (2018).
39. Sebastian, A. et al. Tutorial: brain-inspired computing using phase-change memory devices. *J. Appl. Phys.* **124**, 111101 (2018).
40. Joshi, V. et al. Accurate deep neural network inference using computational phase-change memory. *Nat. Commun.* https://doi.org/10.1038/s41467-020-16108-9 (2020).
41. Hosseini, P., Sebastian, A., Papandreou, N., Wright, C. D. & Bhaskaran, H. Accumulation-based computing using phase-change memories with FET access devices. *Electron Device Lett.* **36**, 975–977 (2015).
42. Le Gallo, M. et al. Mixed-precision in-memory computing. *Nat. Electron.* **1**, 246–253 (2018).
43. 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).

44. Waser, R. & Aono, M. in *Nanoscience and Technology: a Collection of Reviews from Nature Journals* 158–165 (World Scientific, 2010).
45. Kent, A. D. & Worledge, D. C. A new spin on magnetic memories. *Nat. Nanotechnol.* **10**, 187–191 (2015).
46. Close, G. et al. Device, circuit and system-level analysis of noise in multi-bit phase-change memory. In *Proceedings of the International Electron Devices Meeting* (*IEDM*) 29.5.1–29.5.4 (IEEE, 2010).
47. Breitwisch, M. et al. Novel lithography-independent pore phase change memory. In *Proceedings of the Symposium on VLSI Technology* 100–101 (IEEE, 2007).
48. Rahimi, A., Kanerva, P. & Rabaey, J. M. A robust and energy-efficient classifier using brain-inspired hyperdimensional computing. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design* ISLPED 2016, 64–69 (ACM, 2016).
49. Quasthoff, U., Richter, M. & Biemann, C. Corpus portal for search in monolingual corpora. In *Proceedings of the International Conference on Language Resources and Evaluation* (*LREC*) 1799–1802 (ELRA, 2006).
50. Koehn, P. Europarl: a parallel corpus for statistical machine translation. In *Proceedings of the MT Summit* Vol. 5, 79–86 (AAMT, 2005).
51. Mimaroglu, D. S. *Some Text Datasets* (Univ. Massachusetts, accessed 9 March 2018); https://www.cs.umb.edu/smimarog/textmining/datasets/
52. Rahimi, A., Benatti, S., Kanerva, P., Benini, L. & Rabaey, J. M. Hyperdimensional biosignal processing: a case study for EMG-based hand gesture recognition. In *Proceedings of the 2016 IEEE International Conference on Rebooting Computing* (*ICRC*) 1–8 (IEEE, 2016).
53. Chandoke, N., Chitkara, N. & Grover, A. Comparative analysis of sense amplifiers for SRAM in 65 nm CMOS technology. In *Proceedings of the International Conference on Electrical, Computer and Communication Technologies* (*ICECCT*), 1–7 (IEEE, 2015).

Acknowledgements

This work was supported in part by the European Research Council through the European Union's Horizon 2020 Research and Innovation Programme under grant no. 682675 and in part by the European Union's Horizon 2020 Research and Innovation Programme through the project MNEMOSENE under grant no. 780215.

Author contributions

All authors collectively conceived the idea of in-memory hyperdimensional computing. G.K. performed the experiments and analysed the results under the supervision of M.L.G., A.R. and A.S. G.K., M.L.G., A.R. and A.S. wrote the manuscript with input from all authors.

Competing interests

The authors declare no competing interests.

Additional information

Extended data is available for this paper at https://doi.org/10.1038/s41928-020-0410-3.

Supplementary information is available for this paper at https://doi.org/10.1038/s41928-020-0410-3.

Correspondence and requests for materials should be addressed to A.R. or A.S.

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

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

Extended Data Table 1 | Architecture configurations and hyperparameters used for the tree different tasks

Dataset	Input type	*n*-gram size	# of channels	Item Memory (IM)		Associative Memory (AM)	
				# Symbols *h*	Dimensionality *d*	Dimensionality *d*	# Classes *c*
Language	Categorical	4	1	27	10,000	10,000	22
News	Categorical	5	1	27	10,000	10,000	8
EMG	Numerical	5	4	4	10,000	10,000	5

Extended Data Table 2 | Parameters for PCM crossbars energy and area estimation

Common parameters	
Parameter	Value
Read voltage	0.1 V
Current on conducting devices	1 µA
Unit device area	0.2 μm^2

Module-specific parameters		
Parameter	Encoder	AM
Readout time	2.8 ns	100 ns
Active devices per query	145,000	66,000
Energy per SA read	9.8 fJ	-
Energy per ADC read	-	12 pJ
Total SA area	0.034 mm^2	-
Total ADC area	-	0.03 mm^2