Article

A crossbar array of magnetoresistive memory devices for in-memory computing

https://doi.org/10.1038/s41586-021-04196-6

Received: 30 December 2020

Accepted: 28 October 2021

Published online: 12 January 2022

Check for updates

Seungchul Jung[1], Hyungwoo Lee[1], Sungmeen Myung[1], Hyunsoo Kim[1], Seung Keun Yoon[1], Soon-Wan Kwon[1], Yongmin Ju[1], Minje Kim[1], Wooseok Yi[1], Shinhee Han[2], Baeseong Kwon[2], Boyoung Seo[2], Kilho Lee[3], Gwan-Hyeob Koh[3], Kangho Lee[2], Yoonjong Song[3], Changkyu Choi[1], Donhee Ham[1,4]✉ & Sang Joon Kim[1]✉

Implementations of artificial neural networks that borrow analogue techniques could potentially offer low-power alternatives to fully digital approaches[1–3]. One notable example is in-memory computing based on crossbar arrays of non-volatile memories[4–7] that execute, in an analogue manner, multiply–accumulate operations prevalent in artificial neural networks. Various non-volatile memories—including resistive memory[8–13], phase-change memory[14,15] and flash memory[16–19]—have been used for such approaches. However, it remains challenging to develop a crossbar array of spin-transfer-torque magnetoresistive random-access memory (MRAM)[20–22], despite the technology's practical advantages such as endurance and large-scale commercialization[5]. The difficulty stems from the low resistance of MRAM, which would result in large power consumption in a conventional crossbar array that uses current summation for analogue multiply–accumulate operations. Here we report a 64 × 64 crossbar array based on MRAM cells that overcomes the low-resistance issue with an architecture that uses resistance summation for analogue multiply–accumulate operations. The array is integrated with readout electronics in 28-nanometre complementary metal–oxide–semiconductor technology. Using this array, a two-layer perceptron is implemented to classify 10,000 Modified National Institute of Standards and Technology digits with an accuracy of 93.23 per cent (software baseline: 95.24 per cent). In an emulation of a deeper, eight-layer Visual Geometry Group-8 neural network with measured errors, the classification accuracy improves to 98.86 per cent (software baseline: 99.28 per cent). We also use the array to implement a single layer in a ten-layer neural network to realize face detection with an accuracy of 93.4 per cent.

Whereas the current success of artificial intelligence (AI)[23] is achieved by computing artificial neural networks (ANNs) in digital processors[1–3], new processor architectures that employ analogue techniques are being keenly sought in the hope of reducing power dissipation. A prominent example is in-memory computing architectures based on the memory crossbar array[4–7]. With each memory storing a synaptic weight as its conductance value, the crossbar array executes the vector–matrix multiplication, the most prevalent ANN algebra[24]. Each column yields a dot product between the input voltage vector fed to the rows and the column weight vector, by first multiplying the memory conductance and the input voltage at each row–column cross-point via Ohm's law and subsequently summing the resulting cross-point currents along the column via Kirchhoff's law (Methods; Extended Data Fig. 1). This physical matrix multiplication, or analogue multiply–accumulate (MAC) operation, consumes far less power than its digital counterpart. The overall in-memory computing architecture for an ANN would consist of multiple crossbar arrays, with each crossbar array accompanied by readout electronics (data converters) and digital circuits before being connected to the next crossbar array.

The power savings have driven active efforts to realize crossbar arrays using non-volatile memories (NVMs). Of the four NVMs that can be produced at volume—that is, resistive random-access memory[8–13], phase-change random-access memory[14,15], flash memory[16–19] and spin-transfer-torque magnetoresistive random-access memory (STT-MRAM, or MRAM for brevity)[20–22,25,26]—the first three have frequently been used for crossbar arrays, but MRAM crossbar arrays have not been implemented for analogue ANN computing, despite simulation studies[25]. The challenge is low resistance of MRAM[27,28] (Methods), with which conventional crossbar arrays would consume considerable power, defeating the purpose of the crossbar array.

Here we fill this gap and implement an MRAM crossbar array (Fig. 1). We overcome the low-resistance issue with a new crossbar array architecture that replaces the standard current sum with a resistance sum in the analogue MAC operation. This 64 × 64 array, integrated with

[1]Samsung Advanced Institute of Technology, Samsung Electronics, Suwon-si, South Korea. [2]Foundry Business, Samsung Electronics, Yongin-si, South Korea. [3]Semiconductor R&D Center, Samsung Electronics, Hwaseong-si, South Korea. [4]John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA. ✉e-mail: donhee@seas.harvard.edu; sangjoon0919.kim@samsung.com

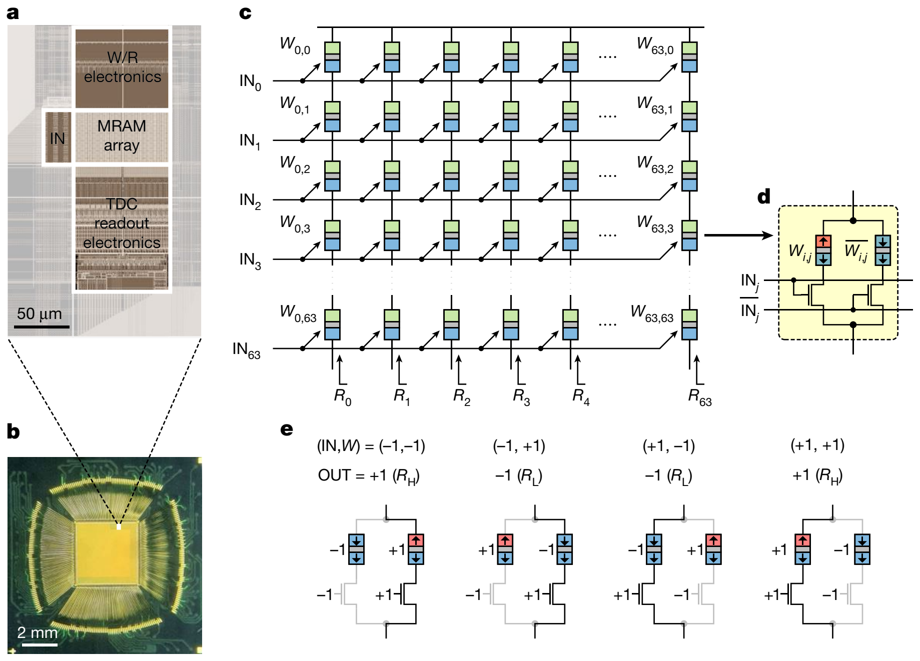


Fig. 1 | MRAM crossbar array. a, **b**, Micrograph (**b**) and layout (**a**) of the 64 × 64 MRAM crossbar array with the peripheral circuit integrated in 28-nm CMOS technology. The crossbar array is sandwiched between the write/read (W/R) electronics at the top and the TDC readout electronics at the bottom. The input data controller (IN) is on the left. **c**, **d**, MRAM crossbar array architecture. Each bit-cell (**d**), occupying a conservatively large area of 0.933 µm^2 for this fabrication, consists of two FET switches and two MTJs, and these bit-cells are connected in series to form a column in the crossbar array (**c**). In each bit-cell, the gates of the left and right FETs are driven respectively by voltage IN (either V_H (1) or V_L (−1)) and its complementary. The left and right MJT–FET paths store respectively weight W (either R_H (1) or R_L (−1)) and its complementary. Each of the resistances, R_H and R_L, adds in the FET switch resistance to the MTJ resistance. The MTJ write/read lines are omitted here for simplicity (Methods; Extended Data Fig. 2). **e**, Four configurations of a bit-cell for four possible combinations of IN and W: {IN, W} = {−1, −1}, {−1, 1}, {1, −1} and {1, 1}, and their corresponding bit-cell output resistance, which is either R_H (1) or R_L (−1). Each column that links these bit-cells in series (**c**) has a total resistance R, which is the sum of all bit-cell output resistances in the column, and serves as the output of the column.

time-to-digital converter (TDC) readout electronics in 28-nm complementary metal–oxide–semiconductor (CMOS) technology, is used for image classification and face-detection tasks. Our intention is not to compete with other memory types for in-memory computing, but to complement them. No single memory type has dominated electronics thus far, as different types bring their own merits and drawbacks. In the case of MRAM, it excels in terms of precision, energy, speed, stability and endurance[5], whereas its 1-bit nature would require an increased network size or a longer computing time to achieve a given performance[29]. In-memory computing may also develop into different applications with differing memory types. It is from this perspective that the lack of MRAM crossbar arrays for AI computing represents a critical gap, and by bridging this gap, our device may help to advance the frontiers of in-memory computing.

MRAM crossbar array

The MRAM is a current-controlled magnetic tunnel junction (MTJ) consisting of two ferromagnetic layers surrounding a thin insulator, and is located between two metal layers in CMOS technology[20]. The magnetizations of the two magnetic layers can be parallel or antiparallel. The parallel state exhibits a low resistance (R_L) across the layered structure, whereas the antiparallel state exhibits a high resistance (R_H). These two resistances representing 1 bit tend to be noticeably smaller than other NVMs[30]. In our case, $R_L \approx 13$ kΩ and $R_H \approx 26$ kΩ (Methods), and each includes the resistance of a field-effect transistor (FET) switch, as every MTJ is accompanied by a FET switch.

Owing to the small R_L and R_H, the standard crossbar array with current-sum columns would consume considerable power. To overcome this, our crossbar array introduces resistance-sum columns, with the goal still being to obtain the dot product. This architecture begins with a new bit-cell design (bit-cell refers to the element at a row–column intersection). Each bit-cell combines two paths in parallel, with each path formed by one MTJ and one FET switch in series (Fig. 1d). The FET gate of the left path is driven by a binary input voltage IN (V_L = 0 V, or V_H = 1.8 V), whereas the FET gate of the right path is driven by the voltage complementary to IN. The left MTJ–FET path stores a synaptic weight of W (R_L or R_H; each of these is the sum of the MTJ and FET switch resistances), whereas the right MTJ–FET path stores the weight complementary to W. Then IN selects either the left or right path to yield the resistance of the chosen path (R_L or R_H) as the bit-cell output. Figure 1e shows the bit-cell output for all four combinations of IN and W. If we assign 1 and −1 for V_H and V_L, and 1 and −1 for R_H and R_L, the output is the binary multiplication between IN and W. This switching-based analogue XNOR that outputs a resistance replaces the Ohm's law cross-point multiplication that outputs a current in the standard crossbar array.

Each column strings these bit-cells in series (Fig. 1c; Extended Data Fig. 2). The individual bit-cell output resistances are then summed to yield the column resistance R, which is the column output. This column resistance sum replaces the Kirchhoff's law-based column current sum in the standard crossbar array (a similar resistance sum was studied in simulations with flash memory[19]). A column R is the dot product of the input vector consisting of IN values for all rows (which we call the **IN** vector) and the vector consisting of the column W values (which we call the **W** vector). R can assume any value between 64 R_L and 64 R_H (or −64 and 64) spaced by $R_H - R_L$ (or 2), depending on **IN** and **W** vectors. This architecture lowers power consumption for small R_L and R_H.

We first verify the dot product operation with a d.c. measurement (Fig. 2c). To this end, we set $W = R_H$ across the entire array and apply 200 **IN** vectors for each of the 65 possible R values. Since there are over 200 ways to distribute R_H and R_L values down a column to obtain each possible R—except $R = 64\,R_L$, $63R_L + R_H$, $R_L + 63R_H$ and $64R_H$—its

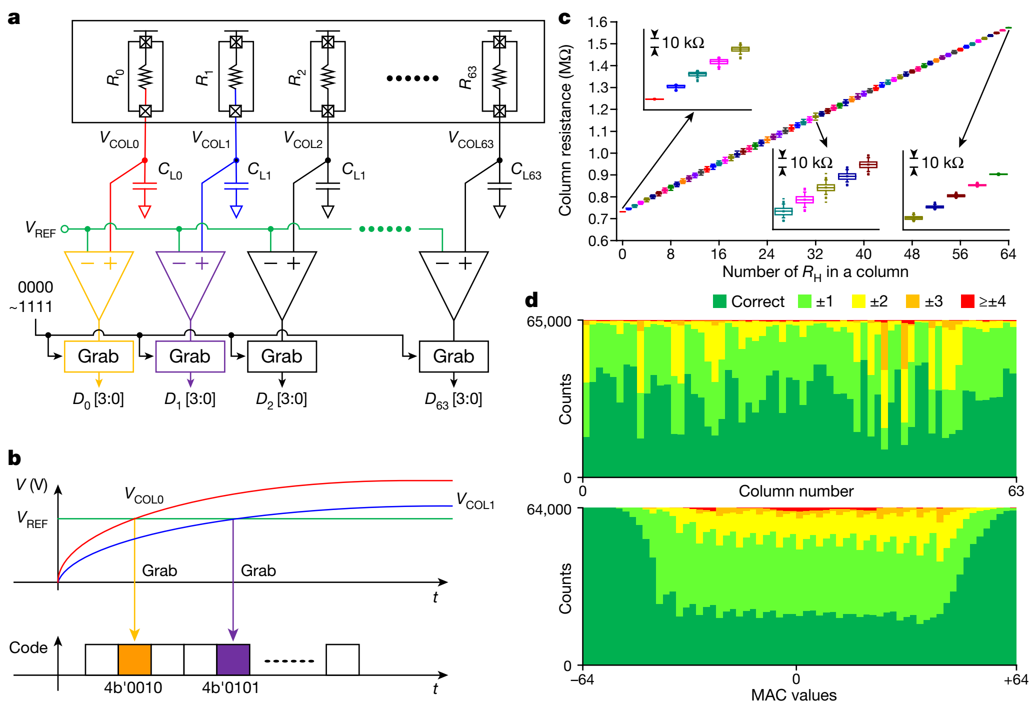


Fig. 2 | Characterization of the MRAM crossbar array. a, TDC readout electronics for the crossbar array. Each column with a total resistance *R* exhibits a time delay in charging both an explicit lumped capacitor C_L at the column end and parasitic capacitors distributed along the column (not shown). We measure the time delay for the column-end voltage V_{COL} to increase from zero to V_{REF} using a 4-bit digital counter. We extract *R* from this delay. **b**, Illustration of the delay measurements for two columns with differing *R* values, where their end voltages are depicted with respect to time, *t*. The lower *R* yields a digitized delay 0010 whereas the higher *R* yields a digitized delay 0101. **c**, *R* measured in d.c., with column-to-column variations averaged, for a broad variety of **IN** vectors with $W = R_H$ across the whole array (that is, for each bit-cell, the weight of the left MTJ–FET path is R_H and that of the right path is R_L). This d.c. measurement bypasses the TDC by shorting every reset switch (not shown) parallel to each C_L. The voltage applied over the columns is 1.0 V. Two hundred **IN** vectors were applied for each expected *R* value, as described in the text (see Methods for details). **d**, Using the TDC, we measured *R* (dot product) from individual columns for a broad variety of **IN** vectors (for each expected *R* value, we applied 1,000 **IN** vectors as described in the text) with $W = R_H$ across the whole array. Top: error distribution across all possible *R* values for each column. Bottom: error distribution across all columns for each possible *R* value.

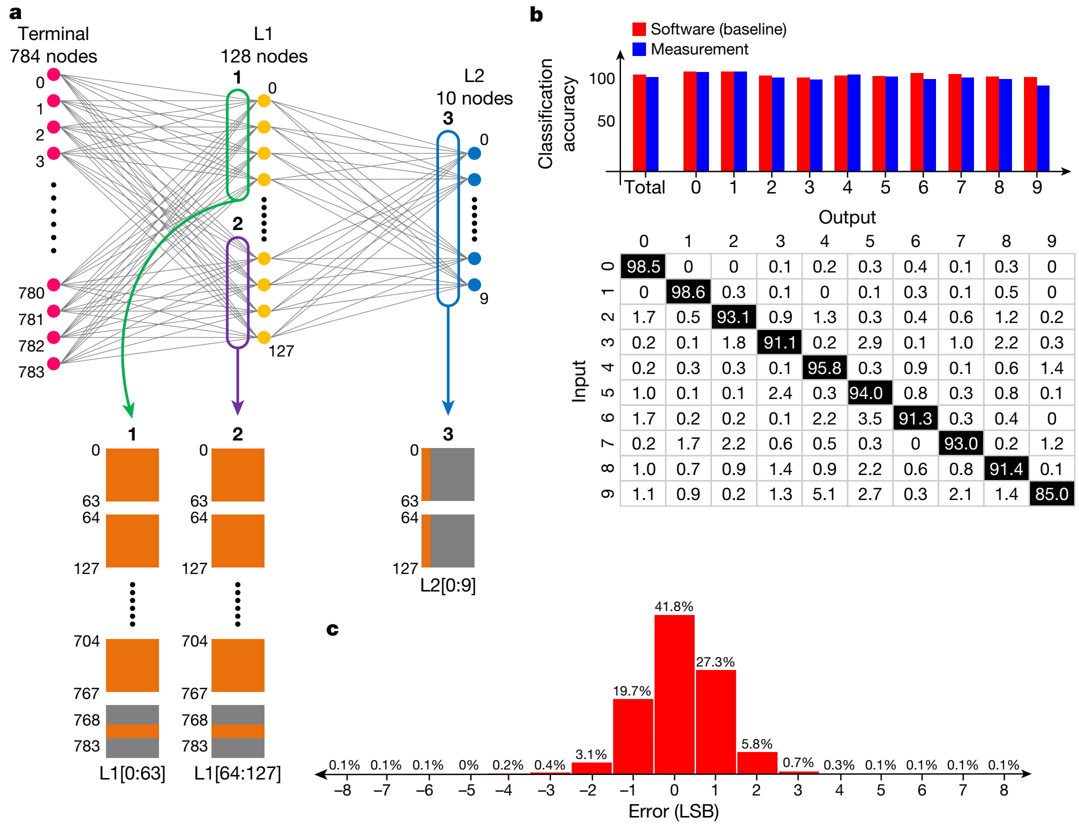
corresponding 200 **IN** vectors are chosen all randomly. For each of the four exceptions, some or all of the corresponding 200 **IN** vectors are redundant: for example, only one **IN** vector can produce $R = 64\,R_H$.

With this set-up, any given **IN** vector would ideally produce the same *R* for all columns, as the corresponding distribution of R_H and R_L values will be identical at all columns. Thus for each given **IN** vector, we measure an averaged *R* by dividing the combined d.c. current measured from all columns by 64 (Methods). Figure 2c shows *R* measured in this way for all of the **IN** vectors applied. The overall linearity confirms the dot product operation. The figure shows variations for each expected *R* except the two end points. This is because the different distributions of R_H and R_L values down a column for each expected *R*—except the two end points—yield slightly differing *R* values for two reasons. First, R_H and R_L differ from bit-cell to bit-cell due to process variations, that is, because every MTJ is slightly different in its physical properties due to imperfections in the fabrication processes, as is every FET (Methods). Second, even with no process variations, different distributions of R_H and R_L used to produce the same expected *R* lead to different biases for FETs down the column, making R_H and R_L values position dependent: this is a purely data (**IN** vector)-dependent error. At either end of Fig. 2c, only a single *R* value is measured because there is only one way R_H and R_L can be distributed in each column (only one possible **IN** vector) there.

Time-domain array readout

For ANN computation, for a given **IN** vector, we extract *R* for each column from its time delay associated with the explicit lumped capacitor $C_L \approx 33$ fF at the column end and parasitic capacitors distributed down the column (~2.1 fF per bit-cell) (Fig. 2a, b). The 4-bit TDC measures the time for the column-end voltage to rise from zero to the reference voltage V_{REF}, charging the capacitors. Different distributions of R_L and R_H values down a column (caused by differing **IN** vectors) for the same expected *R* give slightly different time delays due to the distributed capacitors. Therefore, *R* extracted from the delay contains data-dependent errors (Methods; Extended Data Fig. 3). Alternatively, analogue-to-digital converters (ADCs) could read *R* after converting it into a voltage with a current injection. Although we chose TDCs to meet our power and area constraints, ADCs could be chosen depending on design goals. While *R* is distributed from −64 to 64 spaced by 2 (6-bit), our 4-bit TDC measures *R* from only −46 to 48, truncating the outliers. This seldom affects the ANN accuracy, as *R* lies far more often in the middle of the range[31].

To assess the crossbar array combined with the TDCs, we set $W = R_H$ across the entire array, and applied (for each possible *R*) 1,000 **IN** vectors (which are all different, unless *R* is one of the near-end values) and measured *R* from each column using TDCs. This leads to more than four million *R* (dot product) measurements. The top panel of Fig. 2d shows the measurement error distribution across all possible *R* values at each column (in the ANN computation we add offset digital values to some columns to compensate the column-to-column variations; Methods; Extended Data Fig. 4). The bottom panel of Fig. 2d shows the error distribution across all columns at each possible *R* value. These errors arise from the aforementioned analogue noise—process- and data-dependent variations of R_H and R_L, and data-dependent time delays—and one other source of analogue noise: the TDC non-idealities. The mean absolute error of the roughly four million dot products is 0.47 bits after the digital offset calibration. In this experiment with



	0	1	2	3	4	5	6	7	8	9
0	98.5	0	0	0.1	0.2	0.3	0.4	0.1	0.3	0
1	0	98.6	0.3	0.1	0	0.1	0.3	0.1	0.5	0
2	1.7	0.5	93.1	0.9	1.3	0.3	0.4	0.6	1.2	0.2
3	0.2	0.1	1.8	91.1	0.2	2.9	0.1	1.0	2.2	0.3
4	0.2	0.3	0.3	0.1	95.8	0.3	0.9	0.1	0.6	1.4
5	1.0	0.1	0.1	2.4	0.3	94.0	0.8	0.3	0.8	0.1
6	1.7	0.2	0.2	0.1	2.2	3.5	91.3	0.3	0.4	0
7	0.2	1.7	2.2	0.6	0.5	0.3	0	93.0	0.2	1.2
8	1.0	0.7	0.9	1.4	0.9	2.2	0.6	0.8	91.4	0.1
9	1.1	0.9	0.2	1.3	5.1	2.7	0.3	2.1	1.4	85.0

Fig. 3 | Classification of 10,000 MNIST handwritten digits with a two-layer perceptron neural network. a, Our fully connected two-layer perceptron has an input layer with 784 terminals corresponding to a 28 × 28 pixel image, a first layer with 128 neurons, and a second layer with 10 neurons corresponding to the 10 digits. For the matrix multiplication from the input to the first layer, we updated the weights of our 64 × 64 crossbar array 26 times to cover 784 inputs and 128 outputs. For the matrix multiplication from the first to the second layer, the array weights were updated twice to cover 128 inputs and 10 outputs. Each square at the bottom represents a crossbar array with a given weight update, so there are a total of 26 + 2 = 28 squares. The orange coloured region inside a given square indicates the used part of the crossbar array with the corresponding weight update and the grey coloured region indicates the unused part. **b**, Ten thousand MNIST images were classified with a 93.23 ± 0.05% accuracy (performed three times; baseline: 95.24%). Top: classification accuracy for each digit obtained by the software (red) and measurements (blue). Bottom: Per cent distribution of the ten different output digits each input digit is classified into; in each row, that is, for each input digit, one output digit is correct with the corresponding percentage shaded in black, while the other nine are false. **c**, Distribution of errors collected from the 404 million dot products produced in this classification task.

a 1.0 V supply for the TDCs, the array and TDCs dissipate 347 μW. The power efficiency is 262 tera-operations per second per watt (TOPS W^{-1}), where a single operation is referenced to a 1-bit input and a 1-bit weight in a bit-cell; with a 0.8 V TDC supply, power dissipation decreases to 225 μW while the error increases to 0.83 bits, with a 405 TOPS W^{-1} power efficiency (Methods; Extended Data Table 1; Extended Data Figs. 5 and 6).

AI computing

To apply our crossbar array to AI computing, we used a binary neural network (BNN) algorithm. The BNN algorithm accuracy[32,33] can be improved by representing each real-valued weight with multiple binary values at the cost of network size[34,35] or each real-valued input data as a sequence of multiple binary numbers at the cost of computation speed[36]. We use the latter strategy, expanding each input data into an 8-bit thermometer code, which helps to suppress noise[29] (Methods). Training is done in software, including analogue noise[37].

We classified Modified National Institute of Standards and Technology (MNIST) digits using a two-layer BNN perceptron, which we implemented by recycling the crossbar array. The perceptron had 784 terminals in the input layer to admit a 28 × 28 image, 128 neurons in the first layer and 10 neurons for the 10 digits in the second layer (Fig. 3a). Since our array size was 64 × 64, we updated the array weights 28 times (Methods). A given set of weights was used eight times due to the 8-bit thermometer-coded input data. Each time we updated the weights, we scrambled the columns to remove systematic errors. Activation, batch normalization and softmax were performed in software. In this way, the array performed all MAC operations needed to classify 10,000 MNIST images. We repeated the 10,000-image classification three times, obtaining a 93.23 ± 0.05% accuracy (Fig. 3b), the decrease in which from the 95.24% software baseline arises from the analogue noise. Errors collected from the 404 million dot products mostly fell within ± 1 least significant bit (LSB) (Fig. 3c).

The accuracy of a neural network with more than two layers should have a software baseline higher than 95.24%, the software baseline for the accuracy of the two-layer perceptron. The deeper neural network should also render the analogue noise effect less pronounced. To demonstrate, we developed an emulator based on measured errors (Methods; Extended Data Fig. 7). We first verified its fidelity by applying it to the two-layer perceptron, obtaining an accuracy of 93.18 ± 0.09%, which is within 0.2% of the hardware result (93.23 ± 0.05%). We then emulated an eight-layer Visual Geometry Group (VGG)-8 neural network, obtaining an accuracy of 98.86 ± 0.06% (baseline: 99.28%) in classifying 10,000 MNIST digits.

Face detection—detecting whether there is a face in a scene—is an emerging edge application for always-on cameras. When a face is detected, more power-demanding face authentication can be

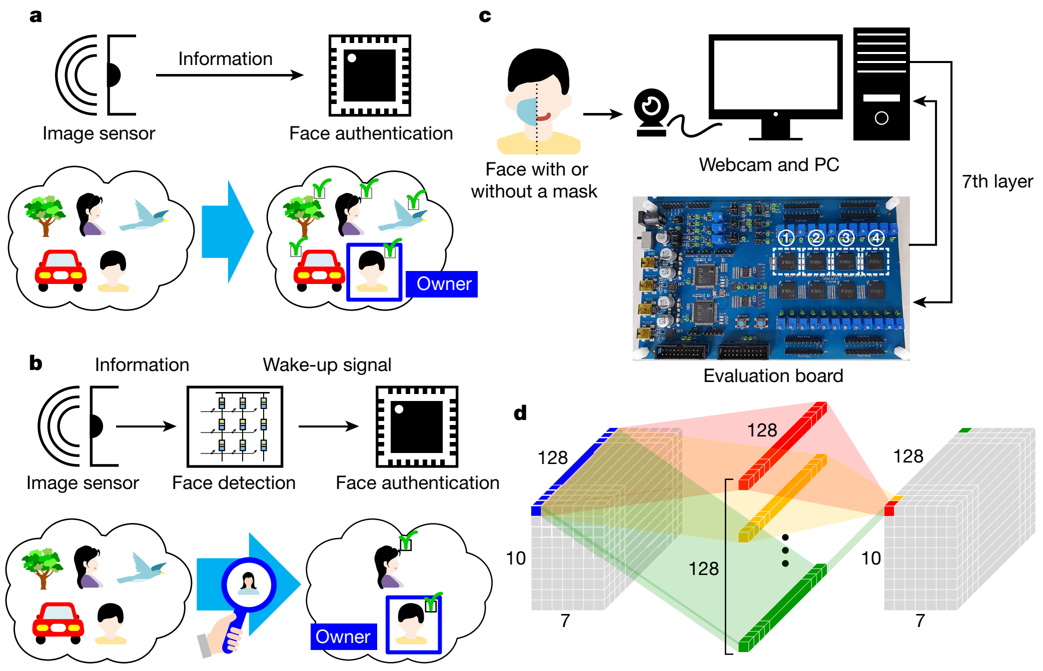


Fig. 4 | Face detection for always-on sensing applications in edge devices. **a**, **b**, Always-on cameras ultimately seek to perform face authentication (recognition of a particular individual's face, resolving fine details) to turn on an edge device. **a**, If face authentication is continually performed for all objects entering the scene, the power consumption would be too high due to heavy computation. **b**, In contrast, face detection determines whether a face is present in the scene or not (not whose face it is) and can be carried out using much less power. Therefore, if higher-power face authentication computation is activated only after a face is detected, overall much less power will be consumed in identifying a particular individual's face. **c**, **d**, Face detection experiment with a ten-layer VGG-like neural network, where the seventh layer (**d**) is implemented with four MRAM crossbar array chips in parallel (**c**). The face detection accuracy for 2,000 scenes with maskless faces and 500 scenes with masked faces is 93.4%. By combining this four-chip system with a camera (**c**), we also performed face detection in real time (Supplementary Video 1).

activated to see whether the face belongs to a particular individual (Fig. 4b). This two-step scheme consumes far less power than continually performing face authentication (Fig. 4a). We demonstrate our MRAM array for this face detection application by using it as the seventh (convolution) layer in a ten-layer VGG-like model modified from SqueezeDet[38], with all the rest layers computed in software. The seventh layer consists of a $10 \times 7 \times 128$ feature map and 128 filters, requiring 128×128 weights, which we covered with four array chips in parallel (Fig. 4c, d). The experiment detected 1,851 faces from 2,000 scenes with maskless faces (92.5% accuracy; baseline: 95.0%) and 483 faces from 500 scenes with masked faces (96.6% accuracy; baseline: 98.6%). The overall accuracy was 93.4% (baseline: 95.7%). We also detected faces in real time by combining the four array chips with a camera (Supplementary Video 1).

Outlook

We have demonstrated an MRAM crossbar array, adding the MRAM to other volume-producible NVMs in pursuit of low-power AI via in-memory computing. Opportunities and challenges lie ahead. Our 64×64 array is much smaller than digital MRAM memory macros. This size constraint, particularly the row number, arises from analogue noise, which would accumulate in the column R to eventually surpass the readout LSB. Readout electronics may also limit the array size (Methods). Constructing a system-on-a-chip AI processor that integrates many such limited-size arrays with data converters and digital electronics will be an important challenge for MRAMs, just as for other NVMs. From a broader perspective, memory crossbar arrays could be used not only for ANN computing but also offer a potential platform for downloading biological neuronal networks to mimic the brain[39]. The MRAM crossbar array demonstrated here thus widens the platform choices for such biomimicry applications.

Online content

Any methods, additional references, Nature Research reporting summaries, source data, extended data, supplementary information, acknowledgements, peer review information; details of author contributions and competing interests; and statements of data and code availability are available at https://doi.org/10.1038/s41586-021-04196-6.

1. Horowitz, M. Computing's energy problem (and what we can do about it). In *Proc. International Solid-State Circuits Conference (ISSCC)* 10–14 (IEEE, 2014).
2. 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).
3. Song, J. et al. An 11.5TOPS/W 1024-MAC butterfly structure dual-core sparsity-aware neural processing unit in 8nm flagship mobile SoC. In *2019 IEEE Int. Solid-State Circuits Conference Digest of Technical Papers (ISSCC)* 130–131 (IEEE, 2019).
4. Sebastian, A. et al. Memory devices and applications for in-memory computing. *Nat. Nanotechnol.* **15**, 529–544 (2020).
5. Wang, Z. et al. Resistive switching materials for information processing. *Nat. Rev. Mater.* **5**, 173–195 (2020).
6. Ielmini, D. & Wong, H. P. In-memory computing with resistive switching devices. *Nat. Electron.* **1**, 333–343 (2018).
7. Verma, N. et al. In-memory computing: advances and prospects. *IEEE Solid-State Circuits Mag.* **11**, 43–55 (2019).
8. Woo, J. et al. Improved synaptic behavior under identical pulses using AlO_x/HfO_2 bilayer RRAM array for neuromorphic systems. *IEEE Electron Device Lett.* **37**, 994–997 (2016).
9. Yao, P. et al. Face classification using electronic synapses. *Nat. Commun.* **8**, 15199 (2017).
10. Wu, H. et al. Device and circuit optimization of RRAM for neuromorphic computing. In *2017 IEEE International Electron Devices Meeting* 11.5.1–11.5.4 (IEEE, 2017).
11. Li, C. et al. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nat. Commun.* **9**, 2385 (2018).
12. Chen, W. et al. CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors. *Nat. Electron.* **2**, 420–428 (2019).
13. Yao, P. et al. Fully hardware-implemented memristor convolutional neural network. *Nature* **577**, 641–646 (2020).
14. Le Gallo, M. et al. Mixed-precision in-memory computing. *Nat. Electron.* **1**, 246–253 (2018).
15. Ambrogio, S. et al. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature* **558**, 60–67 (2018).

16. Merrikh-Bayat, F. et al. High-performance mixed-signal neurocomputing with nanoscale floating-gate memory cell arrays. *IEEE Trans Neural Netw. Learn. Syst.* **29**, 4782–4790 (2018).
17. Wang, P. et al. Three-dimensional NAND flash for vector-matrix multiplication. *IEEE Trans. VLSI Syst.* **27**, 988–991 (2019).
18. Xiang, Y. et al. Efficient and robust spike-driven deep convolutional neural networks based on NOR flash computing array. *IEEE Trans. Electron Dev.* **67**, 2329–2335 (2020).
19. Lin, Y.-Y. et al. A novel voltage-accumulation vector-matrix multiplication architecture using resistor-shunted floating gate flash memory device for low-power and high-density neural network applications. In *2018 IEEE International Electron Devices Meeting* 2.4.1–2.4.4 (IEEE, 2018).
20. Song, Y. J. et al. Demonstration of highly manufacturable STT-MRAM embedded in 28nm logic. In *2018 IEEE International Electron Devices Meeting* 18.2.1–18.2.4 (IEEE, 2018).
21. Lee, Y. K. et al. Embedded STT-MRAM in 28-nm FDSOI logic process for industrial MCU/IoT application. In *2018 IEEE Symposium on VLSI Technology* 181–182 (IEEE, 2018).
22. Wei, L. et al. A 7Mb STT-MRAM in 22FFL FinFET technology with 4ns read sensing time at 0.9V using write-verify-write scheme and offset-cancellation sensing technique. In *2019 IEEE Int. Solid-State Circuits Conference Digest of Technical Papers* 214–216 (IEEE, 2019).
23. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).
24. Yu, S. Neuro-inspired computing with emerging nonvolatile memory. *Proc. IEEE* **106**, 260–285 (2018).
25. Patil, A. D. et al. An MRAM-based deep in-memory architecture for deep neural networks. In *2019 IEEE International Symposium on Circuits and Systems* (IEEE, 2019).
26. Zabihi, M. et al. In-memory processing on the spintronic CRAM: from hardware design to application mapping. *IEEE Trans. Comput.* **68**, 1159–1173 (2019).
27. Kang, S. H. Embedded STT-MRAM for energy-efficient and cost-effective mobile systems. In *2014 IEEE Symposium on VLSI Technology* (IEEE, 2014).
28. Zeng, Z. M. et al. Effect of resistance-area product on spin-transfer switching in MgO-based magnetic tunnel junction memory cells. *Appl. Phys. Lett.* **98**, 072512 (2011).
29. Kim, H. & Kwon, S.-W. Full-precision neural networks approximation based on temporal domain binary MAC operations. US patent 17/085,300.
30. Hung, J.-M. et al. Challenges and trends in developing nonvolatile memory-enabled computing chips for intelligent edge devices. *IEEE Trans. Electron Dev.* **67**, 1444–1453 (2020).
31. Jiang, Z., Yin, S., Seo, J. & Seok, M. C3SRAM: an in-memory-computing SRAM macro based on robust capacitive coupling computing mechanism. *IEEE J. Solid-State Circuits* **55**, 1888–1897 (2020).
32. Hubara, I. et al. Binarized neural networks. In *Advances in Neural Information Processing Systems* 4107–4115 (NeurIPS, 2016).
33. Rastegari, M., Ordonez, V., Redmon, J. & Farhadi, A. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *2016 European Conference on Computer Vision* 525–542 (2016).
34. Lin, X., Zhao, C. & Pan, W. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems* 345–353 (NeurIPS, 2017).
35. Zhuang, B. et al. Structured binary neural networks for accurate image classification and semantic segmentation. In *2019 IEEE Conference on Computer Vision and Pattern Recognition* 413–422 (IEEE, 2019).
36. Shafiee, A. et al. ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture* 14–26 (IEEE, 2016).
37. Liu, B. et al. Digital-assisted noise-eliminating training for memristor crossbar-based analog neuromorphic computing engine. In *2013 50th ACM/EDAC/IEEE Design Automation Conference* 1–6 (IEEE, 2013).
38. Wu, B., Iandola, F., Jin, P. H. & Keutzer, K. SqueezeDet: unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *2017 IEEE Conference on Computer Vision and Pattern Recognition* 129–137 (IEEE, 2017).
39. Ham, D., Park, H., Hwang, S. & Kim, K. Neuromorphic electronics based on copying and pasting the brain. *Nat. Electron.* **4**, 635–644 (2021).

Methods

Conventional memory crossbar array

A conventional memory crossbar array is shown in Extended Data Fig. 1a. A memory at each cross-point stores a synaptic weight as its conductance value. A voltage vector representing the input data is fed to the rows and, in response, currents flow from the array's columns, which form an output current vector. For input voltage V_j at the jth row, the memory at the cross-point of the jth row and the ith column with a conductance $G_{i,j}$ generates a current $I_{i,j} = G_{i,j}V_j$ via Ohm's law and all such cross-point currents along the ith column are summed via Kirchhoff's law to produce the following total current at the ith column:

$$I_i = \sum_j G_{i,j}V_j \tag{1}$$

This is the dot product between the input voltage vector and the column weight vector. Therefore, the output current vector consisting of all column currents is a multiplication of the array's weight matrix and the input voltage vector, as written out below (n is the number of columns and m is the number of rows):

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ \vdots \\ I_n \end{bmatrix} = \begin{bmatrix} G_{1,1} & G_{1,2} & G_{1,3} & \cdots & G_{1,m} \\ G_{2,1} & G_{2,2} & G_{2,3} & \cdots & G_{2,m} \\ G_{3,1} & G_{3,2} & G_{3,3} & \cdots & G_{3,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ G_{n,1} & G_{n,2} & G_{n,3} & \cdots & G_{n,m} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ \vdots \\ V_m \end{bmatrix} \tag{2}$$

Such vector–matrix multiplications are prevalent in neural network computation (Extended Data Fig. 1b).

MTJ resistance

When measured from all 8,192 MTJ–FET paths across the crossbar array, $R_H \approx 26$ kΩ (standard deviation $\sigma \approx 2.0$ kΩ) and $R_L \approx 13$ kΩ ($\sigma \approx 1.6$ kΩ). Each of these values includes the FET switch resistance, as every MTJ is accompanied by a FET switch. While the MTJ resistances could in principle be increased by changing device geometry, it is difficult in practice. For example, while reducing the MTJ area can increase its resistance, the area is already minimized for the main market: digital memory. Increasing the thickness of the insulator sandwiched between the two magnetic layers could also increase the resistance but such a structure would demand a higher write voltage or current, which may not be possible in the CMOS technology in which the MRAM is embedded.

MTJ write/read operation

For each column, two data lines are prepared (as shown in Extended Data Fig. 2a) to write and read MTJs. Extended Data Fig. 2b shows an example of the write operation: the left MTJ in the top second bit-cell is connected to the two data lines by turning on relevant switches. The left data line connected to $V_{WRITE} = 1.5$ V and the right data line connected to ground write what corresponds to R_L to the selected MTJ. The reversed voltage (not shown in the figure), that is, the left data line connected to ground and the right data line connected to V_{WRITE}, writes what corresponds to R_H to the selected MTJ. Extended Data Fig. 2c shows the read operation on the left MTJ in the top second bit-cell. The left data line is connected to a current source I_{READ} while the right data line is connected to a fixed common voltage V_{CM}. The voltage developed at the current source node is $R_{[1],left} \times I_{READ} + V_{CM}$, where $R_{[1],left}$ is the resistance of the second bit-cell's left MTJ–FET path. Comparison of this to V_{READ} determines whether $R_{[1],left}$ is R_H or R_L.

Crossbar array weight update

Weights trained in software are programmed into the crossbar array by repeating the MTJ write procedure described in MTJ write/read operation section above, row by row. For a given row, all left MTJ–FET paths of bit-cells are selected first, and then their respective W weights are written simultaneously. Subsequently, all right MTJ–FET paths of bit-cells of the same row are selected, and their respective W-complementary weights are written. Each of the two writes takes one clock cycle (clock frequency: 11.1 MHz). We repeat this programming procedure row by row. Since $V_{WRITE} = 1.5$ V and each writing current for a given bit-cell path is well below 100 μA, the energy expenditure for writing the entire array is less than ~110 nJ.

Details of the d.c. measurements

We apply a 1.0-V supply at the top of all columns and set $W = R_H$ all across the array. We apply 13,000 (mostly different) **IN** vectors as described in the main text. For any given **IN** vector, we measure 64 column d.c. currents altogether, bypassing the lumped column-end capacitors by short-circuiting the reset switches parallel to them (in this d.c. measurement, our circuit allows us to measure only the total current combining all 64 columns, while in the time domain it allows the measurement of individual column R using the TDCs). Once the total d.c. current is measured, we divide it by 64 to obtain the per-column R. A given **IN** vector gives exactly the same column distribution of R_H and R_L values across all columns, thus the expected R corresponding to the **IN** vector is the same for all columns. In other words, the divide-by-64 operation to obtain the per-column R only averages the non-ideal column-to-column variations. Figure 2c displays this per-column R measured for all **IN** vectors applied.

In Fig. 2c, the deviations of the measured R for a given expected R are due both to the process- and data-dependent variations of R_H and R_L, as explained in the main text, but due to the aforementioned averaging (the divide-by-64 operation), the systematic data-dependent variations are more pronounced than the process variations. For instance, in the middle point of Fig. 2c, the measured standard deviation (which subsumes both process- and data-dependent variations of R_H and R_L) is 4.9 kΩ, whereas the pure process variation effect simulated at the same point gives a standard deviation of up to 1.4 kΩ. For the actual ANN computing done with the TDC, however, each column R is measured individually without the divide-by-64 operation, and thus the process- and data-dependent variations are more on a par with each other, both contributing to the errors reported in Fig. 2d.

Data-dependent delay and associated error

In ANN computation, we extract the dot product R of a column from the measured time delay of the column. The delay arises from R, the explicit lumped capacitor $C_L \approx 33$ fF at the column end, and the parasitic capacitors distributed along the column ($C_p \approx 2.1$ fF per bit-cell). The effect of the distributed parasitic capacitance is as appreciable as the lumped capacitance in determining the delay. The Elmore model gives the following time constant for the length-N column

$$\tau = \sum_{k=1}^{N} kR_kC_P + \sum_{k=1}^{N} R_kC_L, \tag{3}$$

where R_k is the resistance of the kth bit-cell, determined by the IN voltage and the W weight of the bit-cell, and is either R_H or R_L. At the same time, the column resistance sum R is given by

$$R = \sum_{k=1}^{N} R_k \tag{4}$$

If R_k is R_H for all k, or R_L for all k, τ of equation (3) is proportional to R of equation (4) with a proportional constant, or effective capacitance, of

$$C = (N+1)(C_P/2) + C_L \tag{5}$$

But for a general distribution of R_k along the column, τ of equation (3) is not precisely proportional to R of equation (4). Since our goal is to extract the true R of equation (4) from the time delay proportional to τ of equation (3), and our extraction scheme assumes a linear relationship between τ and R, the central question is how much the correct nonlinear τ vs R relationship defined by equations (3) and (4) deviates from the approximate linear relationship $\tau = RC$ that our readout scheme assumes. Note that this deviation, or error, originates from the data dependent time delay seen in equation (3), which is the effect of the distributed capacitors.

To assess, after setting $W = R_H$ for the entire column, we calculate the estimated $R = \tau/C$ from equations (3) and (5) and the true R of equation (4) for a broad variety of **IN** vectors, which produce many different distributions of R_H and R_L along the column. We apply 200 different **IN** vectors to create 200 different combinations of R_H and R_L along the column for each true R of equation (4) (we have already discussed how we deal with the special situations that occur at near-end values of R in the main text). Extended Data Fig. 3 shows the estimated R vs the true R obtained for the large number of **IN** vectors. Here we have used fixed values of R_H and R_L, ignoring the process- and data-dependent variations of R_H and R_L, to single out the error due purely to the distributed capacitance effect. As can be seen from Extended Data Fig. 3, the linear approximation is actually quite good, but deviations do exist for each true R value due to the data-dependent delay of equation (3) stemming from the distributed capacitors, except at the two end R values. If we use a bigger lumped capacitor C_L, this error becomes smaller, but it comes at the expense of an increased area.

In summary, the distributed capacitance gives rise to data-dependent time delay, even when individual R_H and R_L values are assumed to be data independent, causing an error in the R extraction from the delay. This error is on a par with the error due to the process- and data-dependent variations of R_H and R_L seen in Fig. 2c (the d.c. data of Fig. 2c do not involve any capacitance effect). In the actual crossbar array operation with the TDC readout, all of these analogue noises—process- and data-dependent variations of R_H and R_L, and data-dependent delays—will manifest in a mixed fashion.

Results of digital offset application

Extended Data Fig. 4 shows the error distribution modified from Fig. 2d, after applying digital offsets to select columns.

Operating frequency

The time delay for the column-end voltage to reach the reference voltage (V_{REF}) rising from zero ranges from 13 ns to 29 ns. The operating frequency in principle can be set at the inverse of twice the maximum time delay, which is 17.2 MHz. But in practice, digital control delay should be also taken into account. This sets the operating frequency at the lower 11.1 MHz.

Power consumption and efficiency

The power dissipation measured for each building block is shown in Extended Data Table 1. The TOPS W^{-1} figure of merit in the table is calculated as follows. As each bit-cell performs two operations per clock cycle (one for switching-based binary multiplication and one for accumulation) with the clocking at 11.1 MHz, there are a total of $64 \times 64 \times 2 \times 11.1$ M operations per second. Dividing this number by the power consumption measured for both the crossbar array and the readout electronics, we obtain 405 TOPS W^{-1} (0.8-V supply for TDC readout electronics) or 262 TOPS W^{-1} (1.0-V supply for TDC readout electronics).

Supply and frequency dependence of performance

Power efficiency and mean absolute errors of dot products measured at various TDC supply voltages and operating frequencies are shown in Extended Data Fig. 5. A lower supply voltage increases the power efficiency at the expense of increased errors (Extended Data Fig. 5a). The error increases sharply above 15 MHz, because the clock duration is too short to cover the time delay (Extended Data Fig. 5b).

Measurement set-up

The MRAM crossbar array chip is controlled by a micro controller unit (MCU) on an evaluation board via a serial peripheral interface (SPI) (Extended Data Fig. 6). The board is controlled by a PC (Python) via a USB connection.

BNN algorithm

Imagine time-expanding each input data into a 7-bit thermometer code, while using 1-bit weights. Then, the final MAC result will be the simple sum of seven intermediate MAC results that stream out serially. Let the non-idealities of the crossbar array and readout electronics add noise with a power of σ_n^2 to each of the seven intermediate MAC results. Then the noise of the final MAC result will be $7\sigma_n^2$ given the simple sum. To compare, if we time-expand each input data into a conventional 3-bit binary code (which has the same number of levels as the 7-bit thermometer code), the final MAC result will again be an accumulation of three intermediate MAC results, with each accompanied by the aforementioned noise with power σ_n^2. But in this case, to produce the final MAC result, the 3 individual MAC results are added with weights 2^2, 2^1 and 2^0. Therefore, the power of the noise accompanying the 3 individual MAC results will be added with weights $(2^2)^2$, $(2^1)^2$, $(2^0)^2$ to yield the total noise power of $(4^2 + 2^2 + 1^2)\sigma_n^2 = 21\sigma_n^2$ for the final MAC result. As seen, as compared to the 3-bit conventional binary code, the 7-bit thermometer code spends 2.33× more time but obtains 3× less noise. Above, we have chosen 7-bit thermometer code to compare it to the 3-bit conventional binary code. Our actual BNN algorithm time-expands each input data into an 8-bit thermometer code, while using 1-bit weights (as an aside, we train weights first in real-values, and second re-train them into 1-bit values: details of this two-step BNN training method can be found in ref. [40]).

Recycling of the single MRAM crossbar array to implement the two-layer perceptron

The perceptron has an input layer of 784 terminals, the first layer of 128 neurons, and the second layer of 10 neurons. Our crossbar array can take 64 inputs and produce 64 outputs at a given time. Therefore, for the input-to-first layer matrix multiplication, we need to update the array weight 13 times to cover the 784 input terminals, and 2 more times to cover the 128 neurons, so a total of 26 updates are required. For the first-to-second layer matrix multiplication, the array needs to be updated only 2 times to cover for the 128 inputs, while only 10 out of 64 outputs are used. Since our BNN algorithm time-expands a real-valued input into 8-bit thermometer codes, the crossbar array with a given set of weights is used 8 times. Therefore, for the classification of 10,000 MNIST images, which we repeat 3 times, the total number of dot product operations executed by the recycled MRAM crossbar array is $3 \times (10{,}000 \times 26 \times 8 \times 64 + 10{,}000 \times 2 \times 8 \times 10) \approx 404$ million.

Emulator

By applying a broad variety of **IN** and **W** vectors to the crossbar array, which produce many different R_H–R_L distributions for each column, we amass 4,160,000 dot products digitally converted by TDCs and their errors. Nonetheless, since the number of ways R_H and R_L can be distributed along a column is enormous ($>10^{38}$), R_H–R_L column distributions that occur during the image classification emulation will only be found in our measurement dataset with low probability. Given this, we sort our 4,160,000 error data (which are digital-offset calibrated) into 4,160 different groups based on two indices: the column index, $N_C = 1, 2, \ldots, 64$, and another number index, $N_\Delta = -32, -31, \ldots, 32, 32$, defined as the difference between the number of R_H values in the column top half and that in the column bottom half (these 4,160 data groups are not equally populated, with the $N_\Delta = 0$ group being most populated). In each data group, we organize its errors into a histogram, with examples shown in Extended Data Fig. 7 for the $N_C = 1$ and $N_\Delta = -16$ group and for the $N_C = 1$ and $N_\Delta = 16$ group. In the image classification emulation, for each dot product between an **IN** vector and a **W** vector, we find the data group it belongs to according to N_C and N_Δ, and add to the software dot product an error probabilistically chosen from this data group according to its error histogram. The key idea of this emulation lies in that the error data groups defined by N_C and N_Δ exhibit reasonably distinct error characteristics. For example, the two data groups of Extended Data Fig. 7 have error distributions biased in opposite directions. The emulator also models in skews observed in the measurements. Since this is a model, its validity is to be tested by comparing its result to the hardware result: when we apply this emulator to the two-layer perceptron of the main text, we obtain a 93.18 ± 0.09% accuracy (repeated three times), which is within 0.2% of the hardware result of 93.23 ± 0.05%.

MRAM crossbar array size limitation due to the TDC readout scheme

In addition to the analogue noise that limits the array size, the TDC readout scheme may also limit the array size. If N of the $N \times N$ crossbar array scales linearly along with the lumped capacitor, the delay time constant, the number of operations and the array area all scale quadratically. If V_{REF} is set at the delay time constant, the operating frequency scales inverse quadratically. The power consumption of the crossbar array associated with the capacitor charging then remains constant. Therefore, the throughput and TOPS W^{-1} remain the same, while TOPS mm^{-2} scales inverse quadratically. In reality, the scaling consideration is more complex: for example, the frequency does not have to change so markedly as we can adjust V_{REF}, and the area is determined by not just the crossbar array but also the peripheral electronics.

Data availability

The data that support the findings of this study are available from the corresponding authors upon reasonable request.

Code availability

Computer codes are available from the corresponding authors on reasonable request.

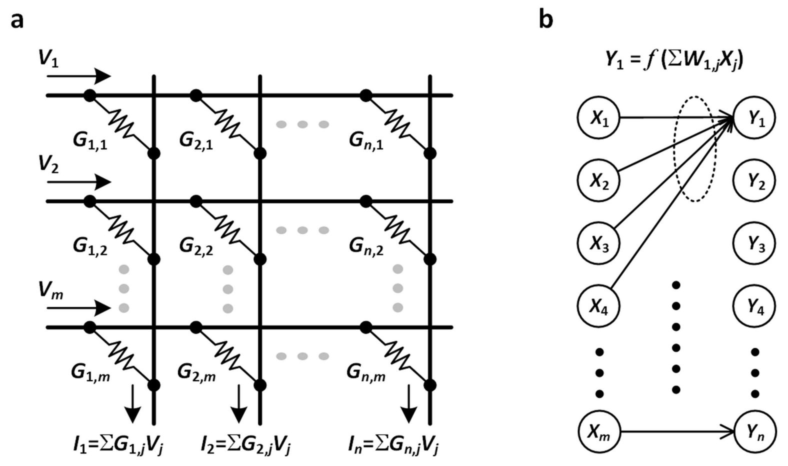
40. Wang, P. et al. Two-step quantization for low-bit neural networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition* 4376–4384 (IEEE, 2018).

Acknowledgements We thank G. Jin (Corporate President of Samsung Electronics), S. Hwang (CEO of Samsung SDS), E. Shim (Corporate EVP of Samsung Electronics) and G. Jeong (Corporate EVP of Samsung Electronics) for technical discussions and support.

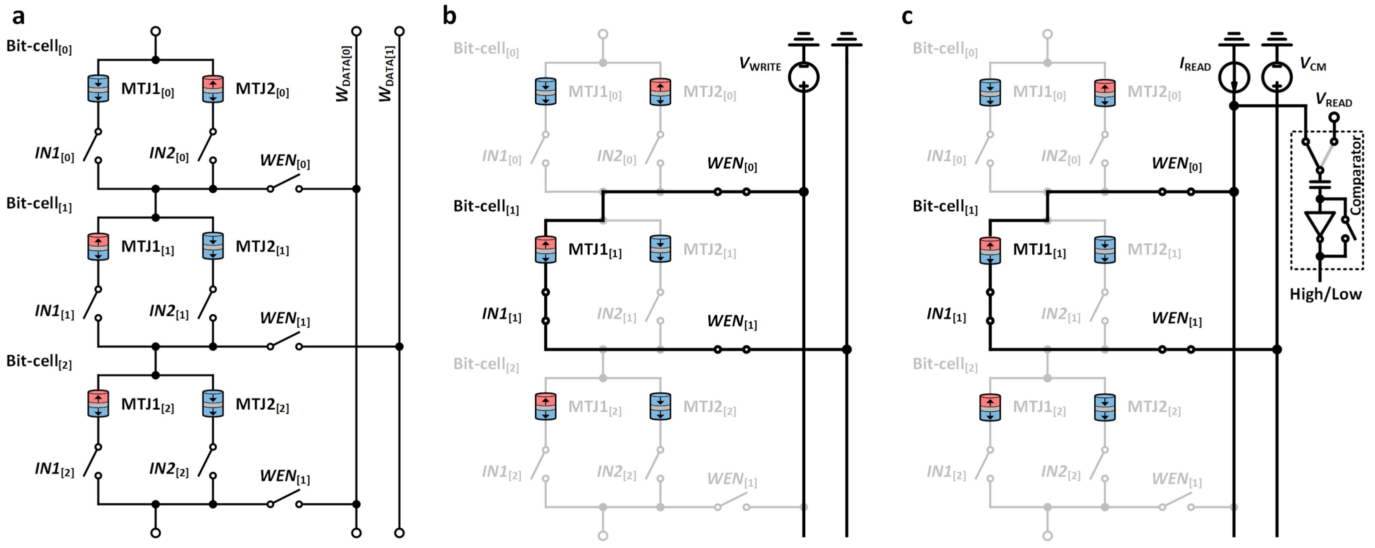
Author contributions S.J., G.-H.K., Y.S., D.H. and S.J.K. devised this work. S.J., H.L., S.M. and Y.J. designed the analogue circuits, and S.-W.K. and M.K. designed the digital circuits for the CMOS chips. S.H., B.K., B.S., Kilho Lee, Kangho Lee, G.-H.K. and Y.S. characterized the MTJs and optimized the fabrication steps for the MRAM crossbar array. S.J. and Y.J. developed evaluation boards. S.J., H.L., S.M. and W.Y. evaluated the MRAM crossbar array. H.K. and C.C. designed and trained the two-layer, eight-layer and ten-layer neural networks. S.J., S.K.Y. and W.Y. performed the MNIST classification experiments. S.J., S.K.Y., S.J.K. and D.H. analysed dot product errors. S.K.Y. and H.K. developed the emulator. S.-W.K. and M.K. developed the face detection system and performed the face detection experiments. S.J., H.L., S.K.Y., S.J.K. and D.H. wrote the paper. G.-H.K., Y.S., C.C., D.H. and S.J.K. supervised this work. All authors read and discussed the manuscript.

Competing interests The authors declare no competing interests.

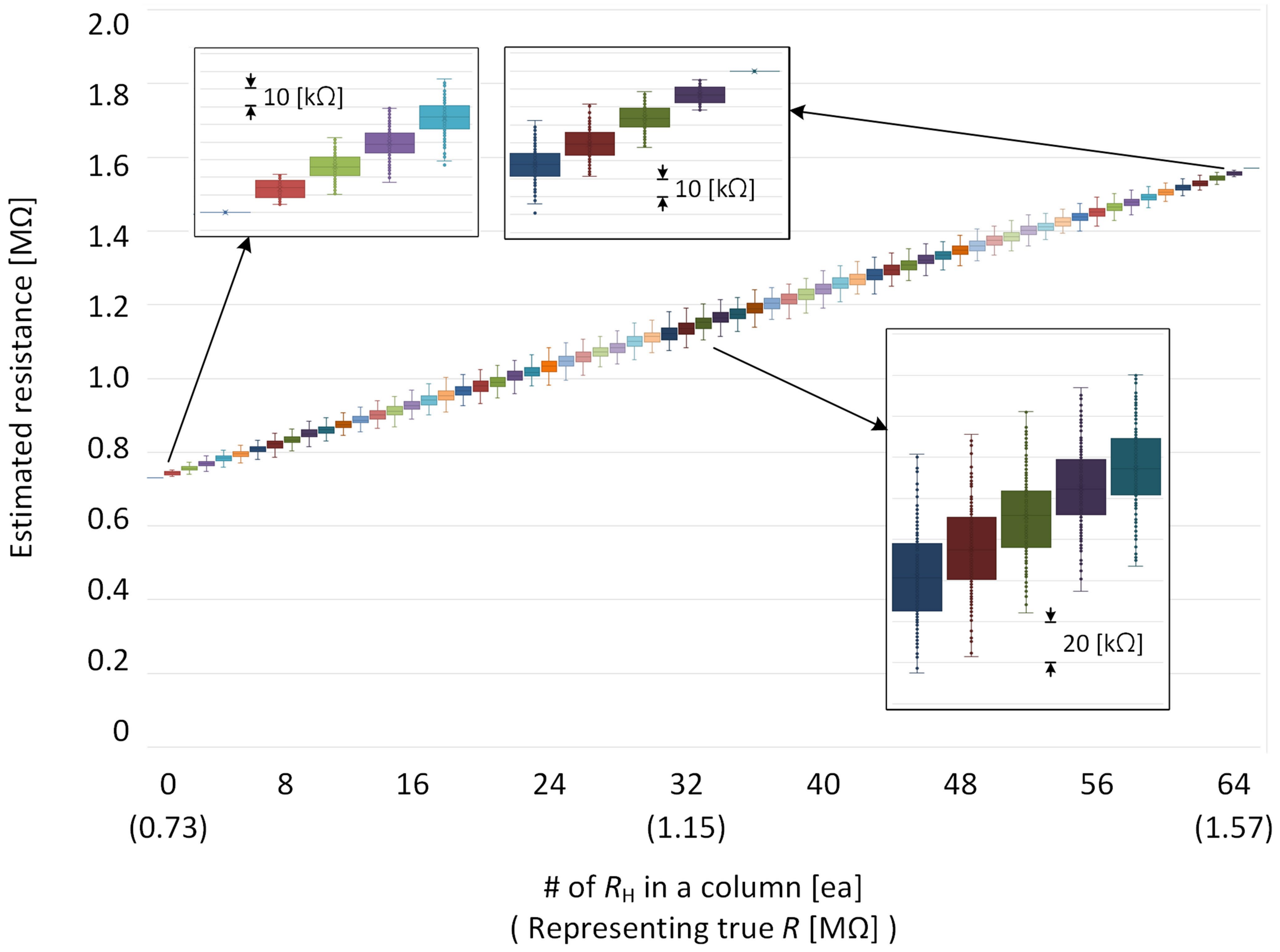
Additional information
Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s41586-021-04196-6.
Correspondence and requests for materials should be addressed to Donhee Ham or Sang Joon Kim.
Peer review information *Nature* thanks the anonymous reviewer(s) for their contribution to the peer review of this work.
Reprints and permissions information is available at http://www.nature.com/reprints.



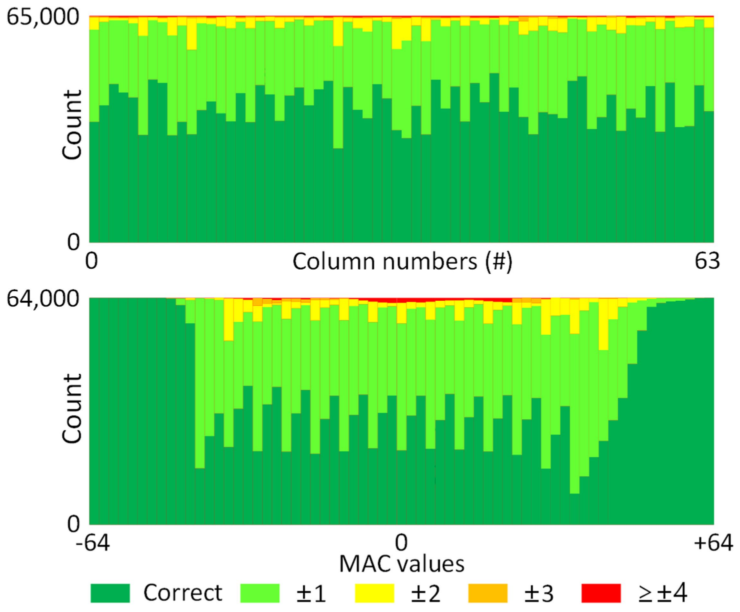
Extended Data Fig. 1 | Conventional memory crossbar array for ANN computing. a, Conventional memory crossbar array to perform analogue vector–matrix multiplication. **b**, Vector–matrix multiplication is prevalent in ANN computing: it is used to transfer data from a layer to the next.



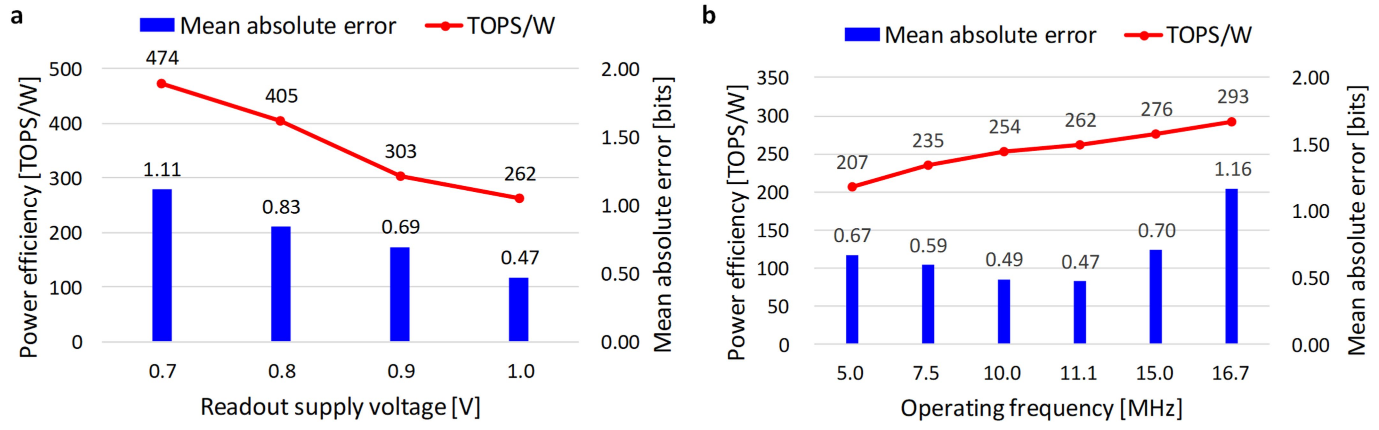
Extended Data Fig. 2 | MTJ write/read operation. a, For each column, two write/read data lines were added with access switches. **b**, Example of write operation. **c**, Example of read operation.



Extended Data Fig. 3 | Estimated *R* from data-dependent delay and associated error. Estimated $R = \tau/C$ calculated from equations (3) and (5) versus true R calculated from equation (4) for a broad variety of **IN** vectors, with $W = R_H$ for the entire column.

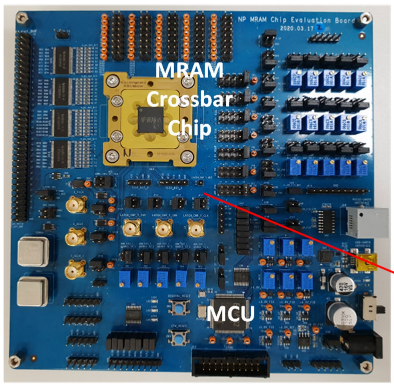


Extended Data Fig. 4 | Error distribution after digital offsets. Error distribution modified from Fig. 2d after applying digital offsets to select columns.

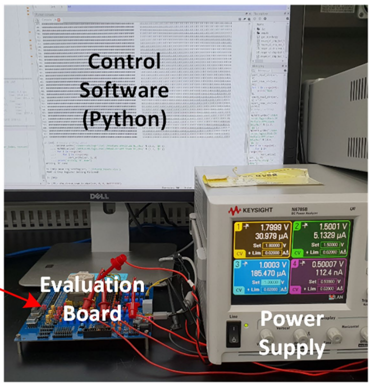


Extended Data Fig. 5 | Performance with varying conditions. a, Measured power efficiency and mean absolute error of the dot products as a function of the supply voltage of the TDC readout electronics for an 11.1 MHz operating frequency. **b**, Measured power efficiency and mean absolute error of the dot products as a function of the operating frequency for a 1.0 V supply voltage for the TDC readout electronics. For both **a** and **b**, each mean absolute error is obtained from 1,600 dot products as in Option 1 in Extended Data Table 1, except for the mean absolute error in the case of the 1.0 V TDC readout supply and the 11.1 MHz operating frequency, for which the error is calculated from ~4 million dot products (this Option 2 in Extended Data Table 1).

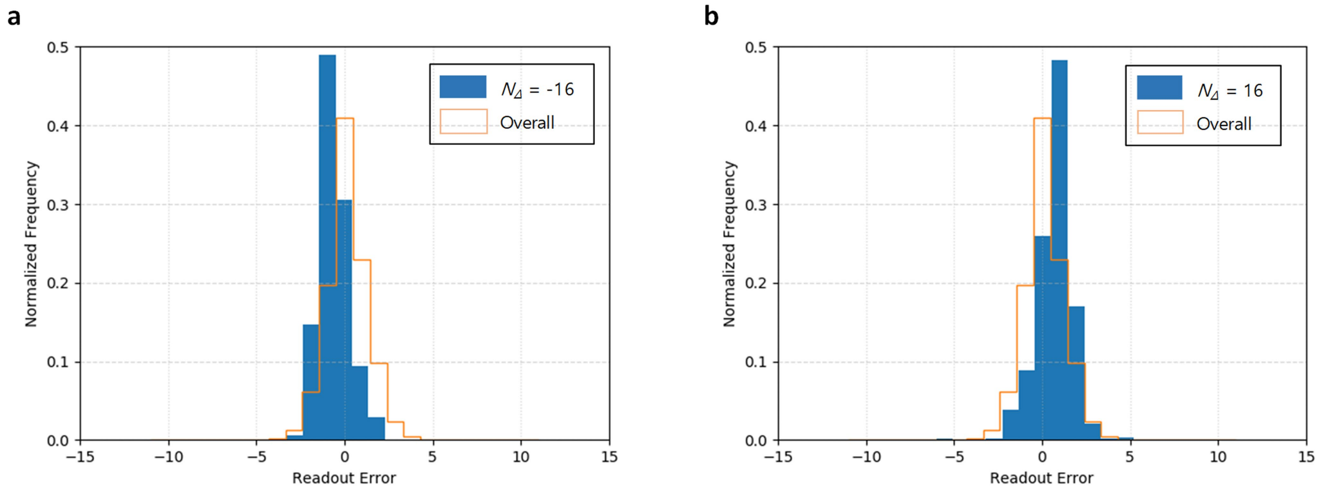
a



b



Extended Data Fig. 6 | Measurement set-up. a, Evaluation board containing voltage regulators, clock generators, an MCU and the MRAM crossbar array chip. **b**, The MCU communicates with the PC via USB.



Extended Data Fig. 7 | Distribution of dot product errors. a, $N_C = 1$ and $N_\Delta = -16$ data group. **b**, $N_C = 1$ and $N_\Delta = 16$ data group.

Extended Data Table 1 | Measured performance summary

Measurement condition			
Operating frequency		11.1 MHz	
Supply voltages	Input driver	1.8 V	
	MRAM array	1.0 V	
	TDC readout	0.8 V (**Option 1**) or 1.0 V (**Option 2**)	
Power consumption and efficiency			
		Option 1	Option 2
Power diss. (µW)	Input driver	60.7	60.1
	MRAM array	42.0	44.0
	TDC readout	122.0	242.7
	Total	224.7	346.8
Effi-ciency	TOPS/W	405	262
	TOPS/mm^2	4.43	4.43
Readout accuracy			
		Option 1	Option 2
Data with no errors		37.2%	60.0%
Data with ±1-bit errors		45.1%	35.3%
Data with ±2-bit errors		14.6%	3.9%
Mean absolute error		0.83 bits	0.47 bits
Note		Errors above are collected from 1,600 dot products (25 dot products per column with randomly chosen inputs and weights) and digital offset calibrated.	Errors above are collected from the ~4 million dot-products from the experiment corresponding to Fig. 2d, but after digital offset calibration.
		Mean absolute error above is the sum of the absolute errors of all dot-product outputs divided by the number of dot products.	