Article

Fully hardware-implemented memristor convolutional neural network

https://doi.org/10.1038/s41586-020-1942-4

Received: 25 November 2018

Accepted: 25 October 2019

Published online: 29 January 2020

Peng Yao[1], Huaqiang Wu[1,2]*, Bin Gao[1,2], Jianshi Tang[1,2], Qingtian Zhang[1], Wenqiang Zhang[1], J. Joshua Yang[3] & He Qian[1,2]

Memristor-enabled neuromorphic computing systems provide a fast and energy-efficient approach to training neural networks[1–4]. However, convolutional neural networks (CNNs)—one of the most important models for image recognition[5]—have not yet been fully hardware-implemented using memristor crossbars, which are cross-point arrays with a memristor device at each intersection. Moreover, achieving software-comparable results is highly challenging owing to the poor yield, large variation and other non-ideal characteristics of devices[6–9]. Here we report the fabrication of high-yield, high-performance and uniform memristor crossbar arrays for the implementation of CNNs, which integrate eight 2,048-cell memristor arrays to improve parallel-computing efficiency. In addition, we propose an effective hybrid-training method to adapt to device imperfections and improve the overall system performance. We built a five-layer memristor-based CNN to perform MNIST[10] image recognition, and achieved a high accuracy of more than 96 per cent. In addition to parallel convolutions using different kernels with shared inputs, replication of multiple identical kernels in memristor arrays was demonstrated for processing different inputs in parallel. The memristor-based CNN neuromorphic system has an energy efficiency more than two orders of magnitude greater than that of state-of-the-art graphics-processing units, and is shown to be scalable to larger networks, such as residual neural networks. Our results are expected to enable a viable memristor-based non-von Neumann hardware solution for deep neural networks and edge computing.

CNNs have become one of the most important deep neural networks (DNNs)[5] and play a vital role in image-processing-related tasks, such as image recognition[11], image segmentation and object detection[12]. A typical computing procedure for a CNN involves a large number of sliding convolutional operations. In this respect, computing units that support parallel multiply–accumulate (MAC) calculations are highly desired. Such demand has led to the redesign of conventional computing systems to operate CNNs with higher performance and lower power consumption, ranging from general application platforms, such as graphics-processing units (GPUs)[13], to application-specific accelerators[14,15]. However, further improvements in computing efficiency will ultimately be constrained by the von Neumann architecture of these systems, in which the physical separation of memory and processing units results in substantial energy consumption and large latency in data shuffling between units[16]. By contrast, memristor-enabled neuromorphic computing provides a promising non-von Neumann computing paradigm in which the data are stored, thus eliminating the cost of data transfer[1,2]. By directly using Ohm's law for multiplication and Kirchhoff's law for accumulation, a memristor array is capable of implementing parallel in-memory MAC operations, leading to analogue in-memory computing with greatly improved speed and energy efficiency[3].

Studies on memristor-based neuromorphic computing have covered a broad range of topics, from device optimization to system implementation[6,17–23]. Several experimental demonstrations[4,24–28] related to practical applications of in-memory computing have been reported as well. The most recent studies report the demonstrations of two-layer[4] and three-layer[27] memristor multi-layer perceptronsfor image recognition using the MNIST (Modified National Institute of Standards and Technology) handwritten-digit database[10]. However, a complete CNN, which is essential for more complex image-recognition tasks, has not yet been demonstrated in a fully memristor-based hardware system. The reason mainly pertains to the lack of an efficient solution for the implementation[27] of a memristor-based CNN (mCNN): first, the fabricated mCNN usually suffers from a poor yield and non-uniformity of memristor crossbar arrays[4,7,8]. Second, it is difficult to achieve a performance (for example, image-recognition accuracy) comparable to software results owing to device imperfections, such as variations, conductance drift and device state locking[6–9]. Third, the key convolutional operation in CNNs is time-consuming because of the need to slide over different input patches, which is usually a sequential process and results in speed mismatch between the memristor convolver and the memristor array for fully connected vector–matrix multiplication (VMM).

[1]Institute of Microelectronics, Beijing Innovation Center for Future Chips (ICFC), Tsinghua University, Beijing, China. [2]Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University, Beijing, China. [3]Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA. *e-mail: wuhq@tsinghua.edu.cn

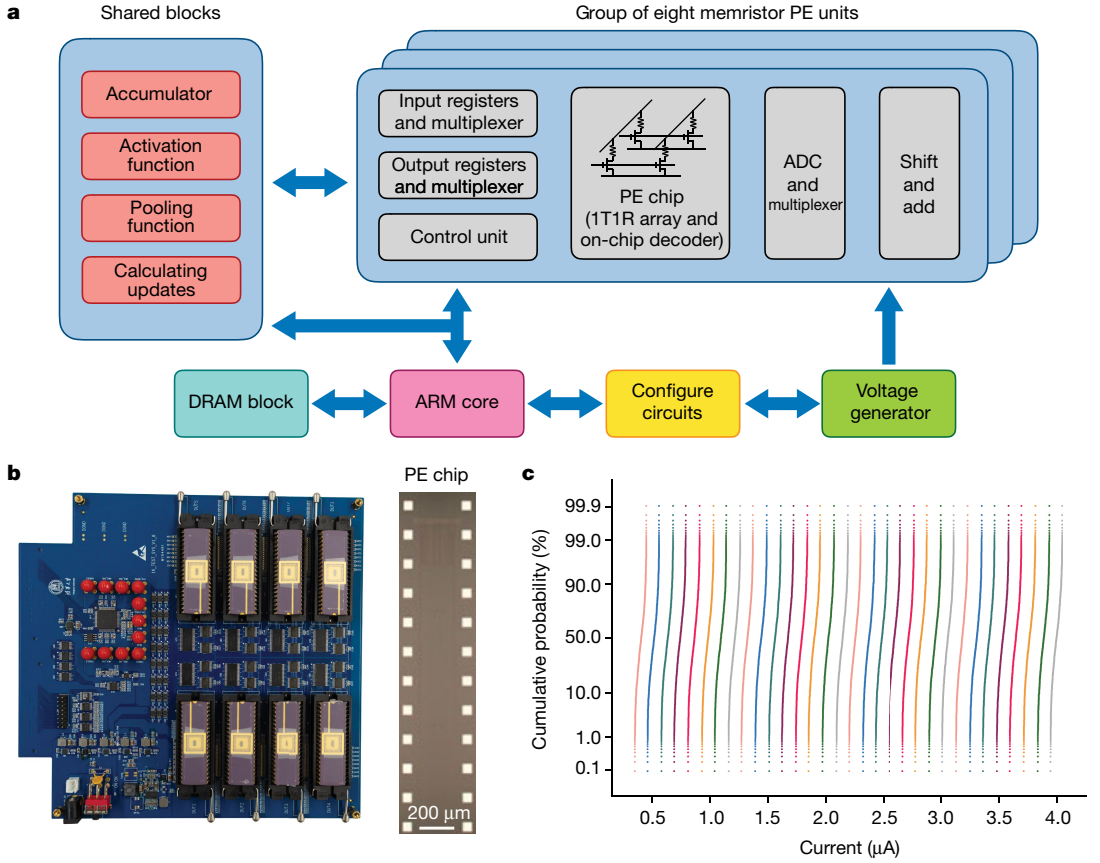


Fig. 1 | Memristor-based hardware system with reliable multi-level conductance states. a, Schematic of the system architecture with eight integrated memristor PE units and other functional blocks. DRAM, dynamic random-access memory; ARM core, control unit with ARM (Acorn RISC Machine) architecture. **b**, Left, photograph of the integrated PCB subsystem, also known as the PE board. Right, image of a partial PE chip consisting of a 2,048-memristor array and on-chip decoder circuits. **c**, Cumulative probability distribution of 1,024 cells with respect to 32 independent conductance states. The conductance is equivalently represented by the read-out current under a 0.2-V voltage pulse. For programming, the SET conditions were V_{WL} = 1.8 V for the word-line voltage, V_{BL} = 2.0 V (50 ns pulse) for the bit-line voltage and V_{SL} = 0 V for the source-line voltage; the RESET conditions were V_{WL} = 4.7 V, V_{BL} = 0 V and V_{SL} = 1.8 V (50 ns pulse).

In this study, a complete five-layer mCNN for MNIST digit image recognition was successfully demonstrated. The optimized material stacks enabled reliable and uniform analogue switching behaviours in 2,048 one-transistor–one-memristor (1T1R) arrays. With the proposed hybrid-training scheme, the experimental recognition accuracy reached 96.19% for the entire test dataset. Furthermore, replication of the convolutional kernels to three parallel memristor convolvers was implemented to reduce the mCNN latency roughly by a factor of 3. Our highly integrated neuromorphic system provides a feasible solution to substantially improve the CNN efficiency by closing the throughput gap between memristor-based convolutional computation and fully connected VMM.

Realizing a practical memristor-based neuromorphic computing system usually requires the integration of multiple memristor crossbar arrays. In general, splitting the weights into different arrays is beneficial for parallel computing, which is increasingly needed with increasing network scales. However, previous memristor-based demonstrations relied on a single array[4,24–26], mainly because of the challenge of producing highly repeatable arrays. The variability and non-ideal characteristics of memristive devices are considered as substantial hurdles to the application of neuromorphic computing[7–9].

Here we propose a versatile memristor-based computing architecture for neural networks, shown in Fig. 1a. The memristor cell uses a material stack of TiN/TaO_x/HfO_x/TiN, and shows continuous conductance-tuning capability (see Supplementary Information) in both potentiation (SET) and depression (RESET) by modulating the electric field and heat[29]. The materials and fabrication process (see Methods for details) are compatible with the conventional CMOS (complementary metal–oxide semiconductor) process, so that the memristor arrays can be conveniently built in the back end of line in a silicon fab to reduce process variations and achieve high reproducibility. The fabricated crossbar arrays exhibit uniform analogue switching behaviours under identical programming conditions. Hence, a multiple-memristor-array hardware system (see Supplementary Information) was built using a customized printed circuit board (PCB) and a field-programmable gate array evaluation board (ZC706, Xilinx). As the system schematic shows, the system mainly consists of eight memristor-based processing elements (PEs). Each PE has its own integrated 2,048-cell memristor array. Each memristor is connected to the drain terminal of a transistor, namely, in a 1T1R configuration (see Supplementary Information). The core PCB subsystem with eight memristor array chips is presented in Fig. 1b. Each memristor array (right inset of Fig. 1b) has an assembly of 128 × 16 1T1R cells. There are 128 parallel word lines and 128 source lines horizontally, and 16 bit lines vertically (see Methods for details). This array exhibits remarkably repeatable multi-level conductance states, as shown by the test results in Fig. 1c and the measured data from the remaining 2,048-cell arrays in Extended Data Fig. 1. Figure 1c shows the distribution of 1,024 memristors in 32 different conductance states, where all the curves are separated without any overlap. Identical SET and RESET pulse trains with a pulse width of 50 ns were employed in the closed-loop programming[24] operations to reach a certain conductance state. The measurement flow is described in Methods.

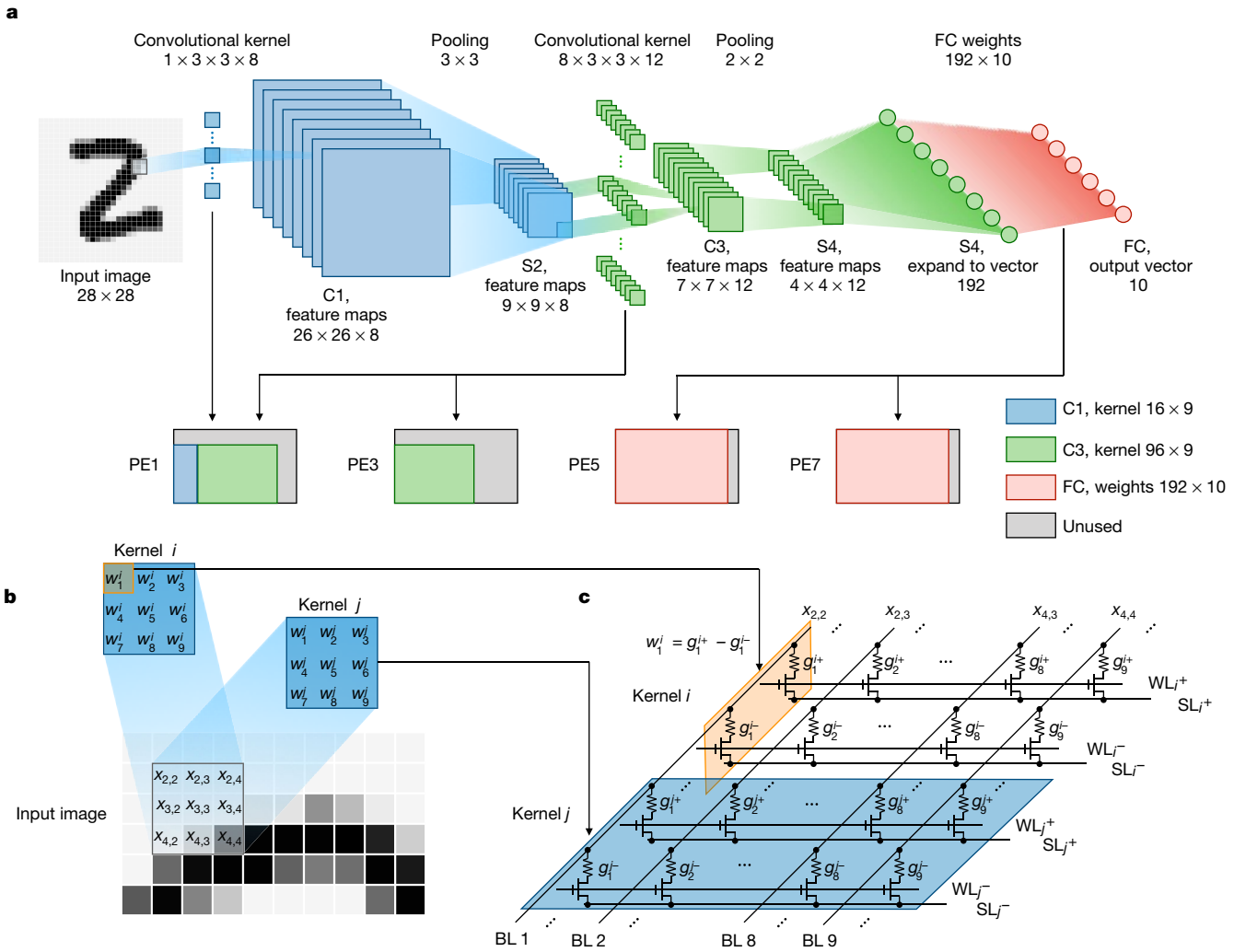


Fig. 2 | Five-layer mCNN with memristor convolver. a, Structure of the five-layer mCNN used for MNIST image recognition, with alternating convolutional (C1, C3) and subsampling (S2, S4) layers. The input is a 28 × 28 greyscale (8-bit) digit image. The mapping relations between the weights of different layers and the independent PEs are illustrated (see Methods for details). The top labels give the kernel size (input channel depth × filter height × filter width × filter batch size) for C1 and C3, the pooling size for S2 and S4 and the weight size for the FC layer (input neuron number × output neuron number). The bottom labels provide the feature map dimension (height × width × channel depth) or the vector dimension. **b**, Typical convolutional case during the slipping process. The grey box in the image confines the input patch of this sample case. $x_{m,n}$ indicates the relevant pixel at the crossing of row m and column n. Kernels i and j each have a total of 3 × 3 weights. **c**, The equivalent memristor convolver of the convolutional operation in **b**. Throughout the entire parallel computing process, all word lines (WL) are set to $V_{WL} = 4.5$ V. The injected bit line (BL) pulses are 0.2 V, and the source lines (SL) are clamped at 0 V. w represents the element value in the weight matrix, and g^+ and g^- represent the device conductance values for the positive and negative weights in the differential pair, respectively.

As shown in Fig. 2a, a five-layer CNN was constructed on a memristor-based hardware system to recognize MNIST handwritten-digit images. The detailed data flow in the CNN and the corresponding memristor mapping are described in Methods.

Realizing memristor-based convolutional operations requires performing sliding operations with various kernels. Memristor arrays are highly efficient in achieving parallel MACs under shared inputs for different kernels[22]. Figure 2b shows a typical convolution example at a particular slipping step, and Fig. 2c reveals the associated events in the 1T1R memristor array. The input value is encoded by the pulse number according to its quantized bit number (Extended Data Fig. 2). A signed kernel weight is mapped to the differential conductance of a pair of memristors. In this manner, all the weights of a kernel are mapped to two conductance rows: one row for positive weights with positive pulse inputs and the other for negative weights with equivalent negative pulse inputs. After inputting the encoded pulses into the bit lines, the output currents through the two differential source lines are sensed and accumulated. The differential current is the weighted sum corresponding to the input patch and the chosen kernel. Different kernels with different weights are mapped to different pairs of differential rows, and the entire memristor array operates MACs in parallel under the same inputs. All the desired weighted-sum results are obtained concurrently.

In typical CNN training, it is necessary to propagate the objective derivative backwards with respect to the last outputs, to determine all weight updates[10]. This task requires highly complex operations to apply encoded read pulses to source lines from back to front and layer by layer. Furthermore, it is challenging to train a complicated memristor DNN, owing to non-ideal device characteristics, such as nonlinearity and asymmetric conductance tuning[6,27]. In contrast to the pure in situ training solution, the ex situ training method appears to be a shortcut that takes advantage of existing high-performing parameters. However, inevitable hardware imperfections, such as defective devices and parasitic wire resistance and capacitance, would blur the weights and degrade the system performance

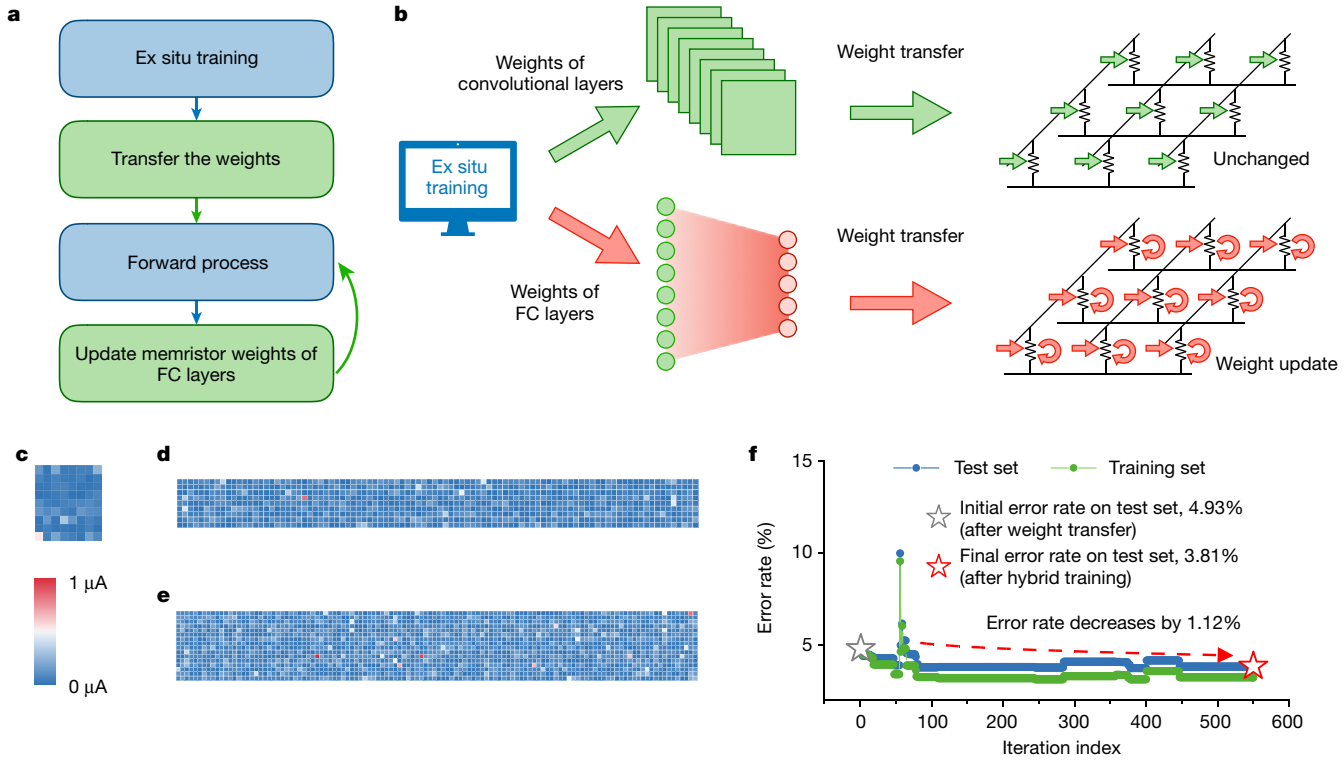


Fig. 3 | Hybrid training on the mCNN. a, Flowchart of the hybrid-training method used in this experimental demonstration. **b**, Diagram of the experimental mCNN demonstration with hybrid training. First, the system transfers the kernel weights of different convolutional layers and the 192 × 10 FC weights to the memristor PEs. Next, the system maintains the kernel weights unchanged and updates only the FC weights through in situ training. **c**–**e**, Distributions of weight-transfer error compared with the target values for the kernel weights in the C1 layer (**c**; 8 × 9 in size), C3 layer (**d**; 96 × 9 in size) and FC layer (**e**; 120 × 16 in size). The colour bar shows the absolute value of weight-transfer error. **f**, Error-rate traces over 550 hybrid-training iteration cycles. The green curve indicates the trend for the 55,000 training images, and the blue curve shows the trend for the 10,000 test images.

when transferring the ex situ learned weights to memristor conductances[4]. Therefore, ex situ training normally requires prior knowledge of the hardware situation and learns weights on the basis of this costly awareness using software.

To circumvent various non-ideal device characteristics, a hybrid-training method is proposed to implement the mCNN. The entire flowchart, illustrated in Fig. 3a, includes two stages. First a CNN model is trained ex situ, and then all the determined weights are transferred to the memristor PEs by a closed-loop writing method. In the next step, the external input propagates forwards through the mCNN, and only the last fully connected (FC) layer is trained in situ afterwards to tune the memristor conductance. It should be pointed out that the proposed hybrid learning method is different from typical transfer learning[27,30]. Hybrid training aims to accommodate the device variations in the previous layers to implement the parallel mCNN efficiently through the in situ training of the memristor weights, whereas transfer learning typically retrains the weights of the FC layers (hereafter, FC weights) using software to obtain knowledge on a new dataset.

Here the hybrid-training technique emphasizes the training of FC weights to compensate for existing device imperfections, and it could be extended as a generic system approach to address device variations and other non-ideal device characteristics by in situ tuning of some memristor weights. Hybrid training is applicable to a wide range of neural network models and could be used to address non-ideal device characteristics regardless of the specific type of memristor device. However, it is worth mentioning that, compared with traditional ex situ training, hybrid training requires fetching the training data to realize in situ conductance tuning; therefore, additional memory blocks or data-transmission modules might be required.

A memristor model is established to validate that the in situ training of only the FC layer is generally adequate for compensating for device imperfections and that it yields remarkable generalization results (see Methods for details). In this manner, hybrid training uses the advantages of both ex situ and in situ training, which complement each other.

To realize an mCNN with hybrid training (Fig. 3b), a model (Fig. 2a) was trained ex situ in Python with TensorFlow on a training set containing 55,000 images. The recognition accuracy for the 10,000 test digit images was 97.99%, which was taken as the baseline accuracy. The well trained weights were rescaled to meet the unified memristor conductance window and quantized from 32-bit floating type to 15-level fixed-point type (see Methods for details). Reasonable weight quantization caused a tolerable performance degradation; for example, the 4-bit quantization of kernel weights and 2-bit quantization of the FC weights for a typical CNN model, AlexNet, was shown to induce a 2.60% increase in the recognition error for ImageNet (a widely used image database for visual object recogoniction) classification compared with the 32-bit quantization of kernel weights and FC weights[31]. The quantization of the 15-level fixed point relaxed the conductance mapping requirements to speed up weight transfer, and ensured a high recognition accuracy of 96.92%, close to the software baseline.

Subsequently, the quantized kernel weights of the convolutional layers and the weights of the FC layer were transferred to the corresponding memristor conductance (Fig. 2a). The weight-transfer accuracy distributions of the convolutional layers C1 and C3 and the FC layer are shown in Fig. 3c–e. The error distributions probably arise from device variations, conductance drift and state locking. The memristor hardware system still achieves a recognition accuracy of 95.07% (see Methods for details) on the 10,000 test images—a 2.92% accuracy loss compared with the baseline value of 97.99%. After an epoch of 550 training iterations (a mini-batch of 100 training images was fed into the mCNN for one iteration) on the entire training database, the recognition error rate for the 10,000 test images decreased considerably from the

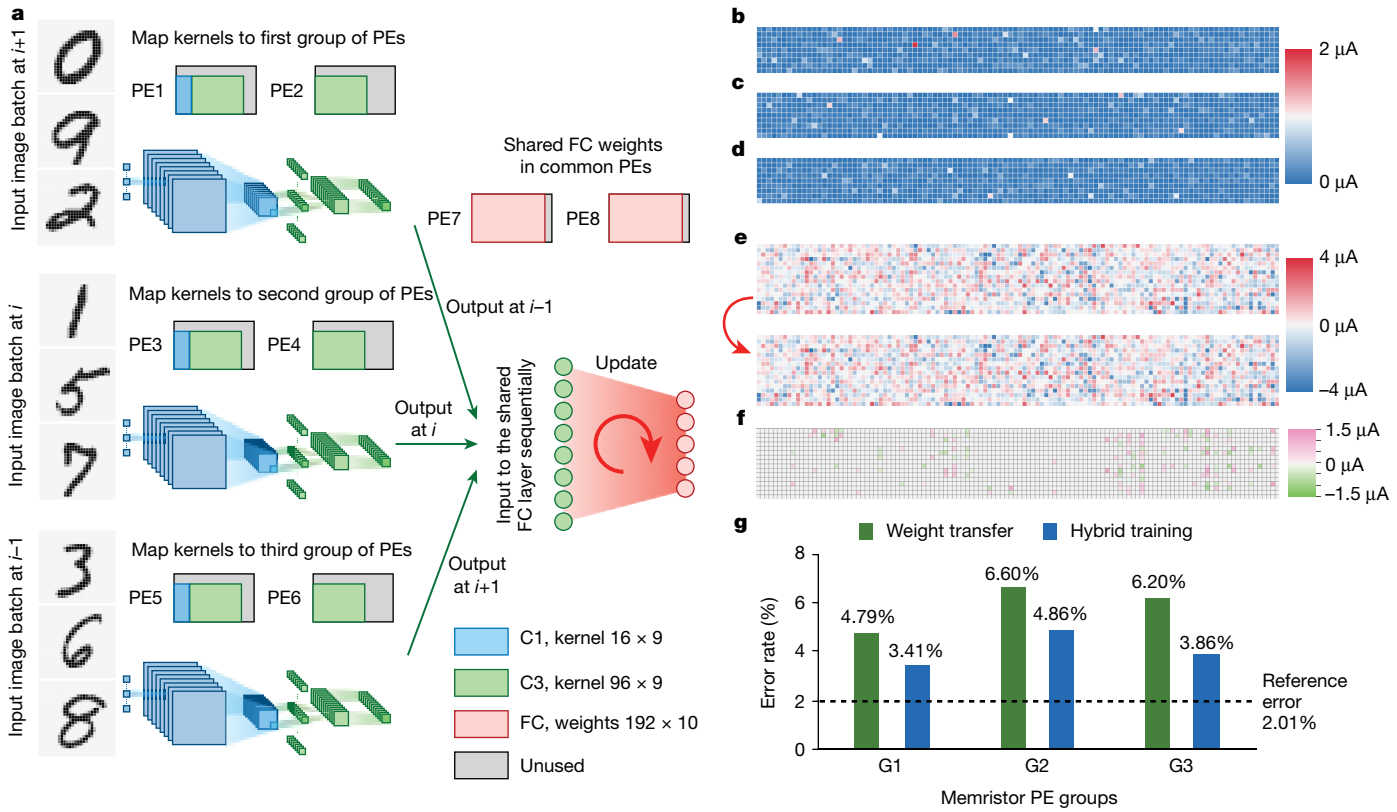


Fig. 4 | Parallel memristor convolvers with hybrid training for improving convolutional efficiency. a, Sketch of the hardware system operation flow with hybrid training used to accommodate non-ideal device characteristics for parallel memristor convolvers. Three batches of input images (handwritten digits on the left) are fed into three PE convolver groups. All the processed intermediate data are fed into the shared FC PEs to complete the in situ tuning. In the neural network schematic, the blue part represents convolutional layer C1 and subsampling layer S2, and the green part represents convolutional layer C3 and subsampling layer S4. In the PE diagram, the blue region represents the kernels of the C1 layer and the green region represents the kernels of C3 layer. **b**–**d**, Weight-transfer error distributions for the mapping from the ex situ-trained kernel weights of the C1 and C3 layers onto three different groups: G1 (**b**), G2 (**c**) and G3 (**d**). The size of the colour map is 104 × 9. The colour bar represents the error in the transferred current values at a 0.2-V read pulse. **e**, Evolution of experimental FC weight distributions (120 × 16) before (top) and after (bottom) hybrid training. **f**, Distribution of conductance-weight changes corresponding to **e**. **g**, The error rate obtained on the test set after hybrid training is considerably lower than that measured immediately after weight transfer for each convolver group.

initial value of 4.93% to the final value of 3.81% (Fig. 3f). The error rate on the training set also dropped from 4.82% to 3.21%.

In a memristor-based neuromorphic computing system, the accuracy loss is mainly attributed to two factors: first, the presence of non-ideal device characteristics, such as device variations, array yield problems and device reliability issues; second, the limited precision due to weight quantization. Even though the accuracy is not fully recovered given the limited quantization precision, the experimental results suggest that the hybrid-training method could effectively recover high recognition accuracy by accommodating device variations across different memristor crossbars. It should be emphasized that in this in situ training process, only the FC weights are updated in an analogue fashion, instead of retraining all the conductance weights. The detailed training procedure is described in Methods.

Further experiments were conducted to show the effect of conductance drift on system performance (see Methods and Extended Data Fig. 3). According to the test results, the reliability of multiple conductance states needs to be further investigated and improved by material and device engineering, which remains an active research area.

Although a memristor convolver can realize the different kernels of a convolutional layer in parallel under shared inputs (Fig. 2b), operating an mCNN remains time-consuming owing to the need to provide different patches of input during the sliding process. Considering that memristor-based MAC computing is more efficient and straightforward when used as the VMM of the FC layer, the severe speed mismatch between the memristor FC implementation and the memristor convolver[27] would induce sizeable efficiency loss. Replicating the same group of weights in multiple parallel memristor arrays appears to be a promising way to recognize an input image efficiently in an mCNN. Spatial parallelism of the memristor convolvers could expedite convolutional sliding tremendously. In practice, transferring the same weights to multiple parallel memristor convolvers is challenging because of unpredictable and inevitable device variations, conductance drift and state locking[6–9], which would induce unavoidable and random mapping error distributions. This process could result in substantial system generalization loss and is therefore considered as a major bottleneck for the realization of an efficient mCNN[27].

A five-layer CNN with three duplicated parallel convolvers on the eight memristor PEs was successfully established in our full hardware system. Hybrid training was again used to address the non-ideal device characteristics. The approach used to perform hybrid training in the parallel operating scheme is sketched in Fig. 4a. In the beginning, the ex situ trained weights were transferred to all eight memristor PEs. Specifically, the kernel weights of the C1 and C3 layers were mapped to three independent groups of PEs. All three parallel memristor convolvers were connected to common PEs of shared FC weights. The specific mapping details are shown in Fig. 4a. Figure 4b, c, d shows the accuracy distributions of the total kernel weights (both C1 and C3 layers) after the weight transfer with respect to the first, second and third groups of PEs. It is clear that inevitable mapping errors exist in each group. The subsequent in situ training of the FC weights (see Methods for details) compensates for the device imperfections naturally. We ran 100 rounds

(that is, 300 batches) to reach a stable recognition accuracy. Figure 4e illustrates the transition of the FC conductance weights before and after the in situ training, and Fig. 4f presents the related distribution of the change in FC weights. After the in situ training of the FC memristors, the error rate decreased accordingly. Figure 4g shows that the error rates with respect to the memristor PE groups G1, G2 and G3 decreased from 4.79%, 6.60% and 6.20% to 3.41%, 4.86% and 3.86%, respectively (see Extended Data Fig. 4 for results on the training set). By dividing one input into three fraction regions uniformly from top to bottom, the parallel memristor convolvers could accelerate the forward process on a single image. The three convolvers operated on their associated input parts simultaneously, and their outputs were fed together into the FC layer to complete the classification. The experimental results show that hybrid training could boost the recognition accuracy on the 10,000 test images from 93.86% to 95.83%. Moreover, we carefully evaluated the hardware performance of memristor-based neuromorphic computing using the experimental data (see Methods, Extended Data Fig. 5 and Extended Data Tables 1, 2). The performance benchmark of the memristor-based neuromorphic computing system shows 110 times better energy efficiency (11,014 GOP s^{-1} W^{-1}; 1 GOP = 10^9 operations) and 30 times better performance density (1,164 GOP s^{-1} mm^{-2}) compared with Tesla V100 GPU[27]. It should be mentioned that some necessary functional blocks (such as the pooling function, the activation function, and the routeing and buffering of data between different neural-network layers) were not considered in the comparison. These blocks could be integrated monolithically with the memristor arrays in the future and accounted for in the energy efficiency calculation.

These findings suggest that the parallel memristor convolvers are highly efficient in achieving a high recognition accuracy while greatly accelerating the mCNN. In addition, the method of replicating the same kernels to different memristor convolvers could be scalable to larger CNN models to boost the parallel computing efficiency. The associated expenditure of chip area could be minimized in the future by employing high-density integration of memristors[32,33]. A standard residual neural network, ResNET-56[11], with a compact memristor model was explored on the CIFAR-10 database and exhibited only a slight accuracy drop of 1.49% compared with the software baseline of 95.57% (see Methods and Extended Data Fig. 6).

Here, we proposed a hybrid training method to maintain high training efficiency and accuracy in a multiple-crossbar memristor CNN system. We should mention that although a small subset of the training data is sufficient in hybrid training, additional memory or data-transfer modules might be required. Moreover, a higher weight quantization precision is needed to fully recover the system accuracy, but at the cost of more hardware resources. Meanwhile, the system performance could be further enhanced by optimizing the peripheral circuits—especially the analogue-to-digital converter (ADC) blocks—and improving device reliability.

In summary, we have experimentally demonstrated a complete mCNN with hybrid training and parallel computing on multiple memristor arrays. The hybrid-training method is a generic system-level solution that accommodates non-ideal device characteristics across different memristor crossbars for various neural networks, regardless of the type of memristor device. The parallel convolution technique, which replicates weights to multiple memristor arrays, is proposed to eliminate the throughput gap between memristor-based convolutional computation and fully connected VMM. Generally, this technique could be extended to other memristor-based neuromorphic systems to efficiently boost their overall performance. The benchmark of our memristor-based neuromorphic computing system shows more than two orders of magnitude better power efficiency and one order of magnitude better performance density compared with Tesla V100 GPU. We expect that the proposed approach will enable the development of more powerful memristor-based neuromorphic systems.

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/s41586-020-1942-4.

1. Ielmini, D. & Wong, H.-S. P. In-memory computing with resistive switching devices. *Nat. Electron.* **1**, 333–343 (2018).
2. Wong, H.-S. P. & Salahuddin, S. Memory leads the way to better computing. *Nat. Nanotechnol.* **10**, 191–194 (2015); correction **10**, 660 (2015).
3. Williams, R. S. What's next? *Comput. Sci. Eng.* **19**, 7–13 (2017).
4. Li, C. et al. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nat. Commun.* **9**, 2385 (2018).
5. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).
6. Wu, H. et al. Device and circuit optimization of RRAM for neuromorphic computing. In *2017 IEEE Int. Electron Devices Meeting (IEDM)* 11.5.1–11.5.4 (IEEE, 2017).
7. Xia, Q. & Yang, J. J. Memristive crossbar arrays for brain-inspired computing. *Nat. Mater.* **18**, 309–323 (2019); correction **18**, 518 (2019).
8. Ding, K. et al. Phase-change heterostructure enables ultralow noise and drift for memory operation. *Science* **366**, 210–215 (2019).
9. Welser, J., Pitera, J. & Goldberg, C. Future computing hardware for AI. In *2018 IEEE Int. Electron Devices Meeting (IEDM)* 1.3.1–1.3.6 (IEEE, 2018).
10. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **86**, 2278–2324 (1998).
11. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition* 770–778 (IEEE, 2016).
12. Ren, S., He, K., Girshick, R. & Sun, J. Faster R-CNN: towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems* 91–99 (NIPS, 2015).
13. Coates, A. et al. Deep learning with COTS HPC systems. In *Proc. 30th Int. Conference on Machine Learning* 1337–1345 (PMLR, 2013).
14. Jouppi, N. P. et al. In-datacenter performance analysis of a tensor processing unit. In *Proc. 44th Int. Symposium on Computer Architecture (ISCA)* 1–12 (IEEE, 2017).
15. Chen, Y.-H., Krishna, T., Emer, J. S. & Sze, V. Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid-State Circuits* **52**, 127–138 (2017).
16. Horowitz, M. Computing's energy problem (and what we can do about it). In *2014 IEEE Int. Solid-State Circuits Conference Digest of Technical Papers (ISSCC)* 10–14 (IEEE, 2014).
17. 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).
18. Burr, G. W. et al. Neuromorphic computing using non-volatile memory. *Adv. Phys. X* **3**, 89–124 (2017).
19. Yu, S. Neuro-inspired computing with emerging nonvolatile memorys. *Proc. IEEE* **106**, 260–285 (2018).
20. Choi, S. et al. SiGe epitaxial memory for neuromorphic computing with reproducible high performance based on engineered dislocations. *Nat. Mater.* **17**, 335–340 (2018).
21. Burr, G. W. et al. Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element. *IEEE Trans. Electron Dev.* **62**, 3498–3507 (2015).
22. Gao, L., Chen, P.-Y. & Yu, S. Demonstration of convolution kernel operation on resistive cross-point array. *IEEE Electron Device Lett.* **37**, 870–873 (2016).
23. Kumar, S., Strachan, J. P. & Williams, R. S. Chaotic dynamics in nanoscale NbO_2 Mott memristors for analogue computing. *Nature* **548**, 318–321 (2017).
24. Yao, P. et al. Face classification using electronic synapses. *Nat. Commun.* **8**, 15199 (2017).
25. Prezioso, M. et al. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* **521**, 61–64 (2015).
26. Sheridan, P. M. et al. Sparse coding with memristor networks. *Nat. Nanotechnol.* **12**, 784–789 (2017).
27. Ambrogio, S. et al. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature* **558**, 60–67 (2018).
28. Serb, A. et al. Unsupervised learning in probabilistic neural networks with multi-state metal-oxide memristive synapses. *Nat. Commun.* **7**, 12611 (2016).
29. Gao, B. et al. Modeling disorder effect of the oxygen vacancy distribution in filamentary analog RRAM for neuromorphic computing. In *2017 IEEE Int. Electron Devices Meeting (IEDM)* 4.4.1–4.4.4 (IEEE, 2017).
30. Donahue, J. et al. DeCAF: a deep convolutional activation feature for generic visual recognition. In *2014 Int. Conference on Machine Learning* 647–655 (ACM, 2014).
31. Han, S., Mao, H. & Dally, W. J. Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. In *2016 International Conference on Learning Representations (ICLR)* (2016).
32. Xu, X. et al. Fully CMOS-compatible 3D vertical RRAM with self-aligned self-selective cell enabling sub-5-nm scaling. In *2016 IEEE Symposium on VLSI Technology* 84–85 (IEEE, 2016).
33. Pi, S. et al. Memristor crossbar arrays with 6-nm half-pitch and 2-nm critical dimension. *Nat. Nanotechnol.* **14**, 35–39 (2019).

Methods

Fabrication of 1T1R memristor array

The fabricated memristor array has a 1T1R structure (see Supplementary Information) in which the memristor stacks are TiN/TaO_x/HfO_x/TiN. This array has a high operation speed of ~10 ns, a high yield (99.99%) and robust endurance performance.

All transistors and major metal interconnections and vias are fabricated in a standard CMOS foundry. The technology node is 130 nm. The back end of line—that is, the procedure used to complete the memristor stacks and the remaining top metal interconnections and vias—is processed in the laboratory. The bottom electrode layer of TiN, the switching layer of HfO_x, the capping layer of TaO_x and the top electrode layer of TiN are deposited sequentially after receiving the wafers from the foundry. The capping layer is used as a thermally enhanced layer[34] to modulate the distribution of the electric field and heat in the switching layer for improved device behaviour. Afterwards, a lithographic process is adopted to form isolated 0.5 μm × 0.5 μm memristor stacks. Then, the SiO_2 dielectric is added and polished. The final steps of etching the vias, depositing aluminium and shaping the remaining interconnection patterns are performed to complete the fabrication process.

Structure of memristor array

A PE chip (Fig. 1b) integrates on-chip encoder circuits and a 128 × 16 1T1R memristor array (see Supplementary Information). The memristor array is constructed by connecting the top electrodes of 128 memristor devices on the same column (that is, bit line) and the 16 transistor sources on the same row (that is, source line). The transistor gate ports facilitate fine memristor-conductance modulation by controlling the device's compliance current with a specific applied gate voltage. The gates in a row are connected to the same line (that is, word line), which is parallel to the source line. This memristor array acts as a pseudo-crossbar of two-port memristors by operating all transistors in the deep-triode region.

Measurements of multi-level conductance states

To measure the reliability of multi-level conductance (see Fig. 1c) in the array, we used a closed-loop writing method with identical SET and RESET pulses. During the test, we supplied the programming pulses to 1,024 randomly chosen memristors from the array to reach 32 individual conductance targets. These target states were distributed within the switching window from 2 μS (that is, 0.4 μA at 0.2-V read voltage) to 20 μS (that is, 4 μA at 0.2-V read voltage) with a uniform interval of 0.58 μS (that is, 116 nA at 0.2-V read voltage). For any desired conductance state, such as I_t at a 0.2-V read voltage, we established the maximum programming pulse number to be 500. In addition, we set the defined target margin parameter ΔI to be ±50 nA. When writing an individual cell to this conductance I_t from any initial state, we continuously applied operating pulses up to the maximum pulse number, and the real-time conductance value was sensed as I_{read} at a 0.2-V read voltage after each programming pulse. If I_{read} was within the desired range, from $I_t - \Delta I$ to $I_t + \Delta I$, the procedure ended successfully. Otherwise, a subsequent SET or RESET pulse was applied accordingly (see Supplementary Information). This entire process was conducted repeatedly over the chosen memristors for the 32 conductance targets. The low-conductance switching range and succinct operation with identical programming pulses could be used to simplify the system design and achieve low-power monolithic integration.

Structure of the five-layer CNN

As shown in Fig. 2a, a C1 layer measuring 26 × 26 × 8 (weight × height × depth) is acquired after convolution with kernel weights measuring 1 × 3 × 3 × 8 (depth × weight × height × batch). The result is subsampled by a pooling layer (S2), that uses a 3 × 3 max-pooling operation over the input with a sliding stride of 3. Then, a C3 layer is formed with 12 stacked feature maps after convolution with the 8 × 3 × 3 × 12 kernels. Another pooling layer (S4, 4 × 4 × 12) is subsequently formed using a 2 × 2 max-pooling operation with a stride of 2. Then, the expanding 192-element vector is passed into the FC layer to obtain the final 10 probability outputs, determining the class to which the input belongs. The inset (dashed box) clarifies how to map the total weights of different layers to memristor PEs of the hardware system. In the experimental demonstration, 9 of 16 memristors in a row were used to realize a 3 × 3 kernel, and the residual cells remained unused. Hence, the 1 × 3 × 3 × 8 kernel weights of the C1 layer required 16 differential rows of memristors (PE1), and the 8 × 3 × 3 × 12 kernel weights of the C3 layer required 192 differential rows of memristors (PE1 and PE3). Owing to the limited number of memristors per row (that is, 16), we split the total 192 weights connected to an output neuron in the FC layer into 24 differential rows and gathered all the corresponding currents of the 12 positive weight rows and 12 negative weight rows (see Supplementary Information). Thus, we were able to map the total FC weights to PE5 (120 rows) and PE7 (120 rows) to carry out the equivalent VMM of the FC layer.

mCNN demonstration

A typical CNN model is created by stacking convolutional and pooling layers repeatedly in series, followed by one or two FC layers at the end. Here we implemented a complete five-layer CNN with our memristor-based hardware system to recognize MNIST handwritten-digit images. The CNN model employed is shown in Fig. 2a. The model contains two convolutional layers, two pooling layers and one FC layer. The max-pooling and ReLU (rectified linear unit) activation functions are employed. The images in this dataset are categorized into 10 classes numbered 0 to 9. The input layer has 784 neurons, which is consistent with the number of pixels in the 28 × 28 input image. There are eight 3 × 3 kernel weights for the first convolutional layer (C1 layer in Fig. 2a) and twelve 3 × 3 × 8 kernel weights for the second convolutional layer (C3 layer in Fig. 2a). The convolutional operation is conducted by calculating the weight sums between the shared local kernel and the generated input patch of the input layer during continuous sliding with a fixed stride step. This operation could be decomposed into parallel MAC operations, which are naturally amenable to a memristor-based in-memory-computing architecture. The input patch is unrolled into a nine-dimensional vector. The hardware system then drives nine channels of pulses accordingly to be supplied to nine bit lines simultaneously. A weight is represented by two differential 1T1R memristor conductances, and thereby a kernel is mapped throughout to the corresponding positive and negative weight rows. The difference in the cumulative flowing currents through these two related source lines is precisely the desired weighted sum of the kernel weights and the input patch. The elements of the second pooling layer (S4 layer in Fig. 2a) are flattened and expended as a 192-dimensional vector to be passed into the last FC layer, and then the weighted-sum values are fed as the input of the softmax function to compute the classification probability. In this manner, the system leads to a map from the original digit image to the ten output probabilities of the last layer. Each output neuron is associated with a defined digital class. The largest among the outputs indicates that the CNN classifies the input image to the matching category accordingly. The associated pooling and ReLU activation functions, as well as the update-calculating modules (such as those computing softmax outputs and weight gradients), were realized by running the codes on ARM cores.

Hybrid training on a subset of the training images

We trained the five-layer CNN model in Python and reached 97.99% recognition accuracy on the test set. The extracted memristor compact model was then used to validate that in situ learning of the FC conductance weights is generally adequate for tolerating device imperfections. After transferring the weights, the recognition accuracy dropped from

97.99% to 95.63% owing to the non-ideal memristor characteristics. Afterwards, all possible combinations of different layers of the weights were tuned—that is, we tried to train the FC weights only, the weights of C1 only, the weights of C3 only, the weights of the FC layer and the C3 layer together, etc. Five epochs of measurements were conducted on the entire training set for the arch trial. As shown in Extended Data Fig. 4b, tuning the FC conductance weights only is most efficient for regaining a remarkable generalization result. Essentially, this approach guarantees a high recognition accuracy and simplifies the original end-to-end training flow by skipping the backward propagation.

Furthermore, we experimentally validated that only a small subset of the training data is sufficient to recover the initial system accuracy using hybrid training, which helps to minimize the hardware resources needed for fetching training data. A five-layer CNN (shown in Fig. 2a) was employed to demonstrate that only 10% of the training dataset is enough to regain the high recognition accuracy of the system. Similarly to the experimental procedure, the trained weights were first transferred to the memristor PEs, and during the transfer some mapping errors were intentionally added by replacing 10% of the target weights with random values; accordingly, the recognition accuracy was reduced to 80.66%. Then 5,500 training images were randomly chosen from the total training dataset (that is, 10% of the 55,000 training images) to update the weights of the FC layer. After performing hybrid training as described in the main text, the accuracy was increased up to 94.40% after ten training epochs. To prove the robustness of our hybrid training technique, the experiment was conducted two more times, and the result is shown in Extended Data Fig. 6c.

Furthermore, a typical ResNET-56 model was used to validate that a small subset of the total training dataset is adequate for recovering the high initial accuracy of the system using hybrid training. The initial accuracy achieved using software was 95.57% (training with 32-bit single-precision floating-point weights), which was degraded to 89.64% after the quantization using 15-level weights. Subsequently, the quantized weights were mapped to the memristor arrays with the established device model in the weight-transfer stage, and the recognition accuracy dropped accordingly to 79.76%. Afterwards, we evaluated the system accuracy after hybrid training using 3% of the total training dataset, that is, 1,500 images from a total of 50,000 training samples. During the simulation, ten trials were made. The final result is plotted in Extended Data Fig. 6d, which depicts the recognition accuracy associated with the key phases of the whole simulation process. It was found that a small subset (3%) of the training data is enough to guarantee a high recognition accuracy (92%)—a 3.57% precision decline against the software result. This simulation result is consistent with the experimental results described above.

The 15-level conductance weight

A 4-bit weight is generally sufficient to achieve a high recognition accuracy for CNNs[31,35]. In this work, an approximate 15-level fixed-point weight was adopted as the differential conductance of a pair of 8-level memristors. The smaller number of conductance states needed within the switching window leads to faster weight transfer because a larger target margin is permitted in the closed-loop writing. Writing an arbitrary 15-level fixed-point number to a differential pair of memristors obviously calls for a consistent ability to distinguish among eight conductance states in each device. In addition, such writing requires that these states be separated within the switching window over the same interval. During the corresponding experiment, the conductance was programmed from 2.5 μS (0.5 μA at a 0.2-V read pulse) up to 20 μS (4 μA at a 0.2-V read pulse) with a constant step of 2.5 μS. The equivalent 15-level weight of the memristor pair was thus referred to the 15 individual differential conductance values that were uniformly distributed from negative 17.5 μS (2.5 μS–20 μS) to positive 17.5 μS (20 μS–2.5 μS). Moreover, the effect of read disturbance on the 15-level conductance weights after applying 10^6 read pulses (0.2 V) is investigated in Extended Data Fig. 7. The experimental data from the array-level tests show that the read operations with the 0.2-V pulse do not disturb the conductance states markedly or systematically.

Estimation of number of programming pulses and programming currents

It is critical to assess the required number of programming pulses in the closed-loop programming system to benchmark the system performance. To estimate the number of programming pulses required to stably converge the memristor to a desired conductance state, we randomly selected 24 rows of 1T1R memristor devices and programmed them to high conductance states, that is, >20 μS (4.0 μA at a 0.2-V read pulse). Afterwards, we divided these devices to eight groups, each with three rows. These eight groups of memristors were correspondingly written to eight different conductance states, from 0.5 μA to 4.0 μA with a uniform interval of 0.5 μA under a read voltage of 0.2 V. The error margin was set as ±100 nA for the eight states. Then the required pulse numbers were analysed statistically on the basis of the measured data, and they are shown in Extended Data Fig. 8a, b.

Even though the test only provides a rough estimation on the required number of programming pulses, it indicates that it strongly depends on the gap between the starting conductance and the desired state. The larger the gap is, the more pulses are needed. Besides, a higher programming resolution—for example, a greater number of required quantized conductance states within the switching window or a smaller desired error margin—would also require a larger number of pulses.

In addition, writing currents are crucial for system design, especially for the calculation of the system energy. However, the programming currents cannot be deduced directly based on the reading currents and writing voltages owing to the nonlinear current–voltage curve. To estimate the programming currents accurately, we swept the d.c. voltage on a single 1T1R cell to measure the write current.

The result is shown in Extended Data Fig. 8c, d. The SET current is around 60 μA at 1.5 V and the RESET current is around 45 μA at −1.2 V. Both voltages are smaller than those measured during the pulse programming process in the array (that is, 2.0 V for SET pulse and −1.8 V for RESET pulse). This is because the 50-ns pulse width used for pulse programming is much shorter than the voltage duration in the d.c. test.

Evaluation of recognition accuracy

Although we have successfully demonstrated the mCNN using parallel operations, the test system crashes easily for long running periods owing to unstable interface connections—for example, the UART interface between the upper computer and lower computer and the FMC connector between the ZC706 board and the customized PE board (Supplementary Information). Besides, the specific implementation of the test system—such as the quantity and speed of the commercial ADC chips—is not optimized for a high-performance design. To facilitate a reliable accuracy analysis within a stable connection period, in this study the conductance of each memristor in different PEs is written first. Then, the current of each memristor is sensed, and this value is consequently used to calculate the recognition accuracy using the ARM core of the test system. The computation process is similar to that realized by the hardware.

Learning and tuning of FC weights

During the second phase of hybrid training, we adopted in situ learning to adjust the FC memristor weights. A stochastic gradient descent (SGD)[10] with a batch size of 100 was used. Even though this mini-batch SGD technique may require extra memory resources to store the intermediate results, it could increase the converging speed and mitigate the overfitting issue. In addition, the memory overhead could be minimized by using the proposed hybrid training method to update the FC weights only and eliminate the demand for storing the intermediate results of all convolutional layers.

For a single iteration cycle, the 100 images drawn from the 55,000 training images were fed into the mCNN and processed from the initial to the final output layer. Then, the gradients of the objective function (here, the cross-entropy loss function) with respect to the FC weighted-sum outputs were determined using the softmax probabilities and the associated true image labels. Later, the quantitative updates of the FC weights were calculated from the intermediate FC inputs and the gradients as follows:

$$\Delta W = \eta \sum_{i=1}^{100} \mathbf{V}_i \times \boldsymbol{\delta}_i \quad (1)$$

Here, the learning rate η is a constant; ΔW describes the desired updates of the weight matrix; $\mathbf{V}_i$ is the intermediate 192-dimensional column vector injected into the FC layer; $\boldsymbol{\delta}_i$ is the calculated ten-dimensional row vector representing the objective derivatives of the FC outputs; and i represents the image index in the batch of 100 images. The accumulated weight updates determine the conductance changes that are ultimately needed on the basis of the following threshold learning rule[36]:

$$\Delta W_{m,n} = \begin{cases} \Delta W_{m,n} & |\Delta W_{m,n}| \geq \mathrm{Th} \\ 0 & |\Delta W_{m,n}| < \mathrm{Th} \end{cases} \quad (2)$$

where $\Delta W_{m,n}$ represents the update cell at the cross point of row m and column n in the weight-update matrix, and Th represents the predefined threshold value used to determine whether the corresponding memristor needs to be programmed. In this study, Th was equal to 1.5 μS (that is, 0.3 μA at a 0.2-V read pulse). This threshold learning rule tends to reduce the number of programming operations by filtering out the original tiny updates, and results in training acceleration and energy saving. Then, parallel programming of the FC memristors could be conducted row by row[24] to achieve the desired updates accordingly. The closed-loop writing method was introduced to circumvent the nonlinear and asymmetric conductance tuning issue, which could be addressed by exploring new basic weight units[27] and programming schemes[4]. Alternatively, if the device performance (for example, the linearity and symmetry) could be further improved, faithful in situ updating could be used with the SGD updating method. This could be more energy- and latency-efficient by encoding the residual error from the output side and the input data from the input side to the corresponding programming pulses directly.

Degradation of conductance weight with time

In hybrid training, the kernel weights were programmed only during the weight-transfer stage. Thus, we set up this experiment by writing all the convolutional kernel weights onto two memristor PEs. After programming all the conductance weights smoothly, we read out these weights to assess how the conductance weights evolved within 30 days.

Extended Data Fig. 3a illustrates how the differential conductance weights (represented by the current read at 0.2 V) drifted with time. The cluster of grey curves in Extended Data Fig. 3a includes the evolution traces of all the conductance weights, where each line represents one individual weight. In the foreground, three typical evolution traces of the conductance weights are highlighted to show the general trend. Because the conductance weights were quantized and programmed using 15 levels, we divided all the weight cells in Extended Data Fig. 3a to these 15 different weight levels, and obtained the mean weight values for each level statistically, as shown in Extended Data Fig. 3b. It can be seen that the 15 levels are still accessible and there is no overlapping between adjacent levels over time.

Extended Data Fig. 3a indicates that the majority of cells can still maintain the weights well, even though there are some tail cells exhibiting noticeable weight drift with time. However, these tail cells could degrade the system accuracy, which will be discussed in the next section.

Hybrid training could be used to address the device reliability issue caused by conductance drifts to some extent, instead of adopting the expensive reprogramming strategy. However, the reliability of the multiple conductance states needs to be further investigated[37] and improved by device and material engineering[38]. The performance of memristor-based neuromorphic systems would benefit considerably from the improvement of device reliability and other non-ideal device characteristics.

Effect of conductance weight degradation on recognition accuracy

By repeating the experiment described in Fig. 3, we investigated how the drifts of the conductance weights affect the system recognition accuracy after hybrid training. The inference accuracy and conductance weights were recorded at 10, 30, 60, 90 and 120 min after hybrid training.

Extended Data Fig. 3c illustrates how the system accuracy changes during the experiment. Similarly to Extended Data Fig. 3a, in Extended Data Fig. 3d we plot the state evolution curves of all the involved weights, including the convolutional kernels and the weights of the FC layer, and three typical lines. Most of the weight states are maintained well within the first 2 h after hybrid training; however, the conductance drifts of the tail cells lead to apparent accuracy degradation.

Training process in parallel memristor convolvers

After transferring the weights, three fetched batches of training images were passed into the three convolver copies separately. By applying the input signal as described in the previous section, we captured three independent batches of interim output of the S4 layer and organized them as the input to the FC layer in a pipeline fashion. The training scheme sets the constraint that a batch of intermediate outputs will not be supplied as input until the previous batch has been used to calculate the desired weight updates and the corresponding FC memristor conductances have been well tuned. The desired updates of the FC weights with respect to the first input batch were calculated according to equation (1), and the relevant memristor conductances were modulated following the threshold learning rule of equation (2). Then, the FC conductances were updated after inputting the second input batch based on the well tuned FC weights of the previous phase. Afterwards, the third batch was used to tune the FC conductance weights sequentially. During this updating stage, another three batches were drawn from the training database and fed into the unoccupied memristor convolvers in parallel. These operations were repeated until the system converged to a stable recognition accuracy.

Benchmarking of system metrics

We evaluated the hardware performance of the memristor-based neuromorphic computing system using the experimental data. Based on the calculation results, we can conclude that the system can achieve 110 times better energy efficiency and 30 times better performance density compared with Tesla V100 GPU.

To benchmark the performance of the memristor-based neuromorphic computing system, we propose a neural processing unit architecture (shown in Extended Data Fig. 5) corresponding to the structure in Fig. 1a. It consists of multiple memristor tiles and each tile contains four memristor cores. The memristor core comprises one 128 × 128 memristor array and all the essential peripheral circuits, including drivers, multiplexer (MUX), MUX controller, sample-and-hold blocks and ADCs. Using the macro core, the typical energy efficiency and performance density are assessed by combining the experimental data (measured from the fabricated memristors at a 130-nm technology node) and simulation data obtained with the simulator XPEsim[39].

In the memristor macro core, we maximize the computing parallelism by connecting two sample-and-hold blocks (S & H groups 1 and 2 in Extended Data Fig. 5) to each column of the array. Every four columns

share one common ADC converter to save chip area and power. When applying a 1-bit read pulse to all rows at 20 MHz, one of the S & H groups is turned on and is connected to the source lines in parallel to convert the accumulated current to voltage. During the next 1-bit read period, the S & H blocks are redirected to another S & H group. Meanwhile, at the beginning of this read phase, the stable voltage signals of the previous S & H group are passed to the ADC block through the control of the MUX-based data path, where every four stable voltages are converted in turn to a digital signal by the shared ADC at the MUX. The 8-bit ADC completes four conversations during the 1-bit inference stage and consumes 2.55 pJ of energy for each conversion. In this manner, there is no idle period for the ADCs and the input pulses are continuously fed into the array.

The detailed metrics, including the energy, latency and area of each block, are listed in Extended Data Table 1, which indicates the system performance for an input of a 1-bit read pulse (0.2 V, 50 ns). In Extended Data Table 1, the memristor-related metrics are evaluated with the measured memristor (130 nm technology node) characteristics. The parameters associated with the other peripheral circuitry blocks are extracted using the simulated circuits at the 65-nm technology node, except for the S & H block[40] and the 8-bit ADC block[41]. When inferencing with a 0.2-V, 50-ns read pulse and considering all the 32 ADCs and other circuitry blocks, the energy consumption for 1-bit computing is 371.89 pJ, and the total occupied chip area is 63,801.94 μm^2/90.69% = 0.0704 mm^2 (the area efficiency is 90.69% for the layout of the macro blocks). Hence, we can assess the metrics of the memristor-based neuromorphic computing system when inputting an 8-bit integer by evaluating the performance, power, area, energy efficiency and performance density, as shown in Extended Data Table 2.

From the above calculations, we obtained an energy efficiency of 11,014 GOP s^{-1} W^{-1} and a performance density of 1,164 GOP s^{-1} mm^{-2}. Compared with the metrics of Tesla V100 GPU[27] (that is, energy efficiency of 100 GOP s^{-1} W^{-1} and performance density of 37 GOP s^{-1} mm^{-2} for 16-bit floating-point number computing), the memristor-based neuromorphic computing system shows roughly 110 times better energy efficiency and 30 times better performance density. It should be mentioned that some necessary functional blocks—such as the pooling function, the activation function, and the routeing and buffering of data between different neural-network layers—were not considered in the comparison. Our system performance could be further improved by using more advanced technology nodes and optimizing the computing architecture and peripheral circuits.

Furthermore, in Extended Data Table 1, we break down the power consumption of each circuitry block in the macro core during the 1-bit inference period. The ADC accounts for 14.4 times the power of the memristor array; however, this number is expected to decrease when the memristor array size increases. For example, the ADC blocks (52 mW) would only account for 1.8 times the power of a 1,024 × 1,024 memristor array (29 mW). This result suggests that both the array size and the ADC optimization should be carefully considered to achieve the best computing efficiency of memristor-based neuromorphic systems.

Scalability demonstration using the ResNET model and the CIFAR-10 database

Replicating the same kernels to different memristor arrays is a crucial approach to improving the efficiency of memristor-based convolvers. This replicating method could mitigate the speed mismatch between the convolutional layer and the FC layer, and overcome the difference in the amount of convolution operations among different convolutional layers. In practice, we can duplicate a certain part of the kernels to realize efficient acceleration. For example, the first convolutional layer in a CNN normally contributes the greatest number of sliding convolutional operations because it has the largest input size; therefore, it causes the largest convolutional computing latency compared with the other layers. In the respect, it is reasonable to replicate only the kernels on the first convolutional layer. Further studies are required to optimize the replicating strategy in the architecture design to yield the desired system performance.

To validate that the approach of replicating the same kernel to different memristor replicas, combined with the hybrid training method, is scalable to larger networks in the presence of intrinsic device variability, a standard residual neural network, ResNET-56, was tested on the CIFAR-10[42] database. We used the compact model incorporating the device variability to simulate the real device performance. Taking the programing error into consideration, a Gaussian distribution was employed to model it as δ[nA] ~ N(0 nA, 108 nA). Here δ is the programming error compared to the target conductance, and $N(\mu, \sigma)$ denotes a Gaussian distribution with mean value μ and standard deviation σ. The statistical parameters were extracted from the measurement results. During the simulation, the memristor was programmed at eight different levels during the weight-transfer stage. We then realized the equivalent 15-level weight by the differential technique as in the experiments. The kernels in the first convolutional layer were replicated into four copies of memristor arrays in the ResNET-56 model. Theoretically, this ResNET-56 model requires 782 memristor arrays with a size of 144 × 16 to implement all the weights.

The initial accuracy achieved by the software was 95.57%, which was degraded to 89.64% after the quantization of the 15-level weights. Subsequently, the quantized weights were mapped to the memristor arrays in the weight-transfer stage, and the recognition accuracy further decreased to 80.06%. However, after the in situ training of the FC layer, the accuracy ultimately recovered to 94.08%—that is, a slight degradation of 1.49% compared with the baseline of 95.57%, as shown in Extended Data Fig. 6a. Extended Data Fig. 6b presents the error rates for the replicated G1, G2, G3 and G4 groups, which decreased from the initial values of 20.24%, 19.83%, 19.58% and 19.84% to 6.11%, 5.84%, 5.87% and 6.34%, respectively.

Data availability

The datasets that we used for benchmarking are publicly available[10,42]. The training methods are provided in refs. [10,36]. The experimental setups are detailed in the text. Other data that support the findings of this study are available from the corresponding author upon reasonable request.

Code availability

The simulator XPEsim used here is publicly available[39]. The codes used for the simulations described in Methods are available from the corresponding author upon reasonable request.

34. Wu, W. et al. A methodology to improve linearity of analog RRAM for neuromorphic computing. In *2018 IEEE Symposium on VLSI Technology* 103–104 (IEEE, 2018).
35. Cai, Y. et al. Training low bitwidth convolutional neural network on RRAM. In *Proc. 23rd Asia and South Pacific Design Automation Conference* 117–122 (IEEE, 2018).
36. Zhang, Q. et al. Sign backpropagation: an on-chip learning algorithm for analog RRAM neuromorphic computing systems. *Neural Netw.* **108** 217–223 (2018).
37. Zhao, M. et al. Investigation of statistical retention of filamentary analog RRAM for neuromophic computing. In *2017 IEEE Int. Electron Devices Meeting (IEDM)* 39.34.31–39.34.34 (IEEE, 2017).
38. Kim, W. et al. Confined PCM-based analog synaptic devices offering low resistance-drift and 1000 programmable states for deep learning. In *2019 Symposium on VLSI Technology* T66–T67 (IEEE, 2019).
39. Zhang, W. et al. Design guidelines of RRAM-based neural-processing unit: a joint device–circuit–algorithm analysis. In *2019 56th ACM/IEEE Design Automation Conference (DAC)* 63.1 (IEEE, 2019).
40. O'Halloran, M. & Sarpeshkar, R. A 10-nW 12-bit accurate analog storage cell with 10-aA leakage. *IEEE J. Solid-State Circuits* **39**, 1985–1996 (2004).
41. Kull, L. et al. A 3.1 mW 8b 1.2 GS/s single-channel asynchronous SAR ADC with alternate comparators for enhanced speed in 32 nm digital SOI CMOS. *IEEE J. Solid-State Circuits* **48**, 3049–3058 (2013).
42. Krizhevsky, A. & Hinton, G. *Learning Multiple Layers of Features From Tiny Images*. Technical report (University of Toronto, 2009); https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Acknowledgements This work is supported in part by the National Natural Science Foundation of China (61851404), the Beijing Municipal Science and Technology Project (Z191100007519008), the National Key R&D Program of China (2016YFA0201801), the Huawei Project (YBN2019075015) and the National Young Thousand Talents Plan.

Author contributions P.Y., H.W. and B.G. conceived and designed the experiments. P.Y. set up the hardware platform and conducted the experiments. Q.Z. performed the simulation work. W.Z. benchmarked the system performance. All authors discussed the results. P.Y., H.W., B.G., J.T. and J.J.Y. contributed to the writing and editing of the manuscript. H.W. and H.Q. supervised the project.

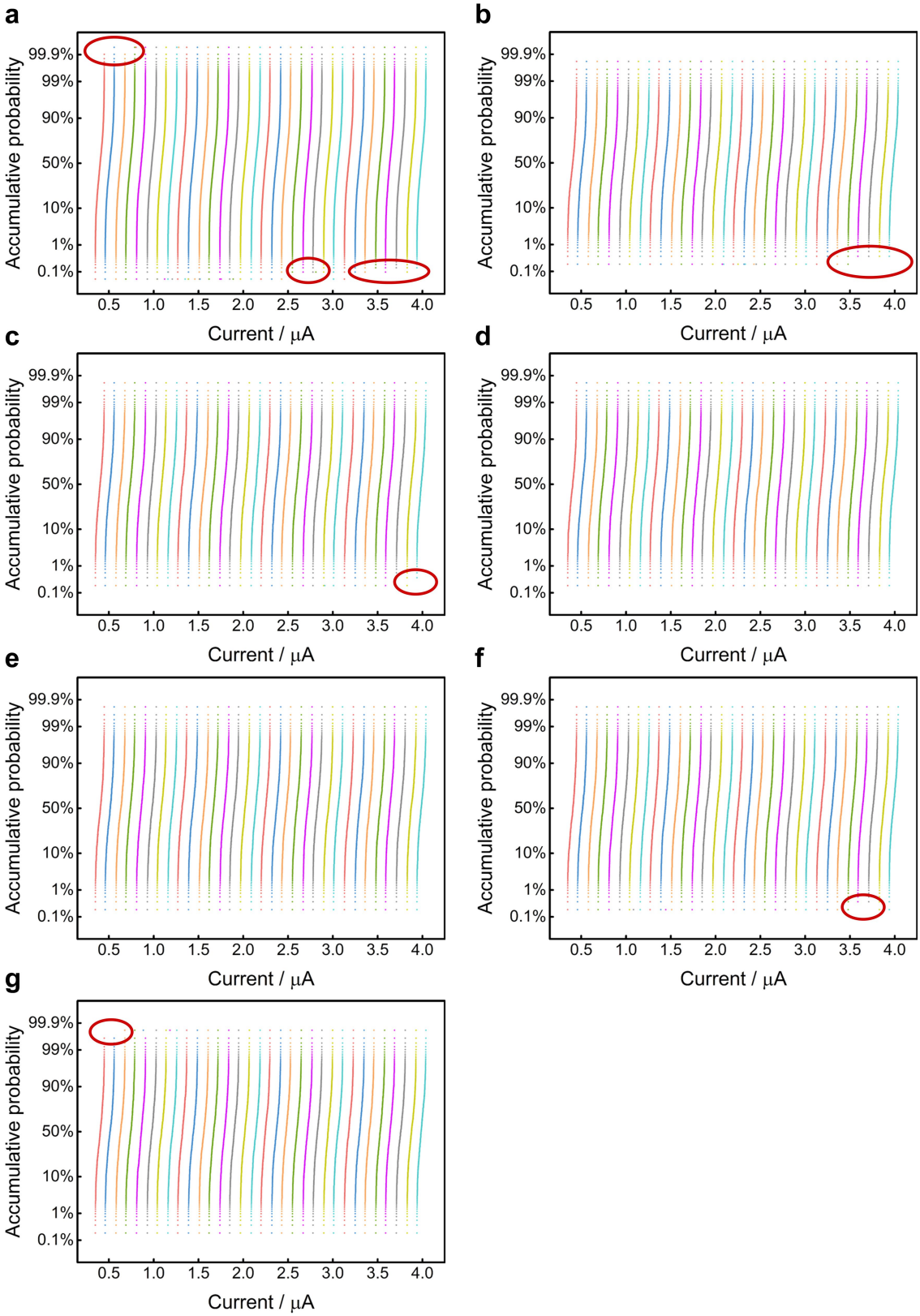
Competing interests The authors declare no competing interests.

Additional information

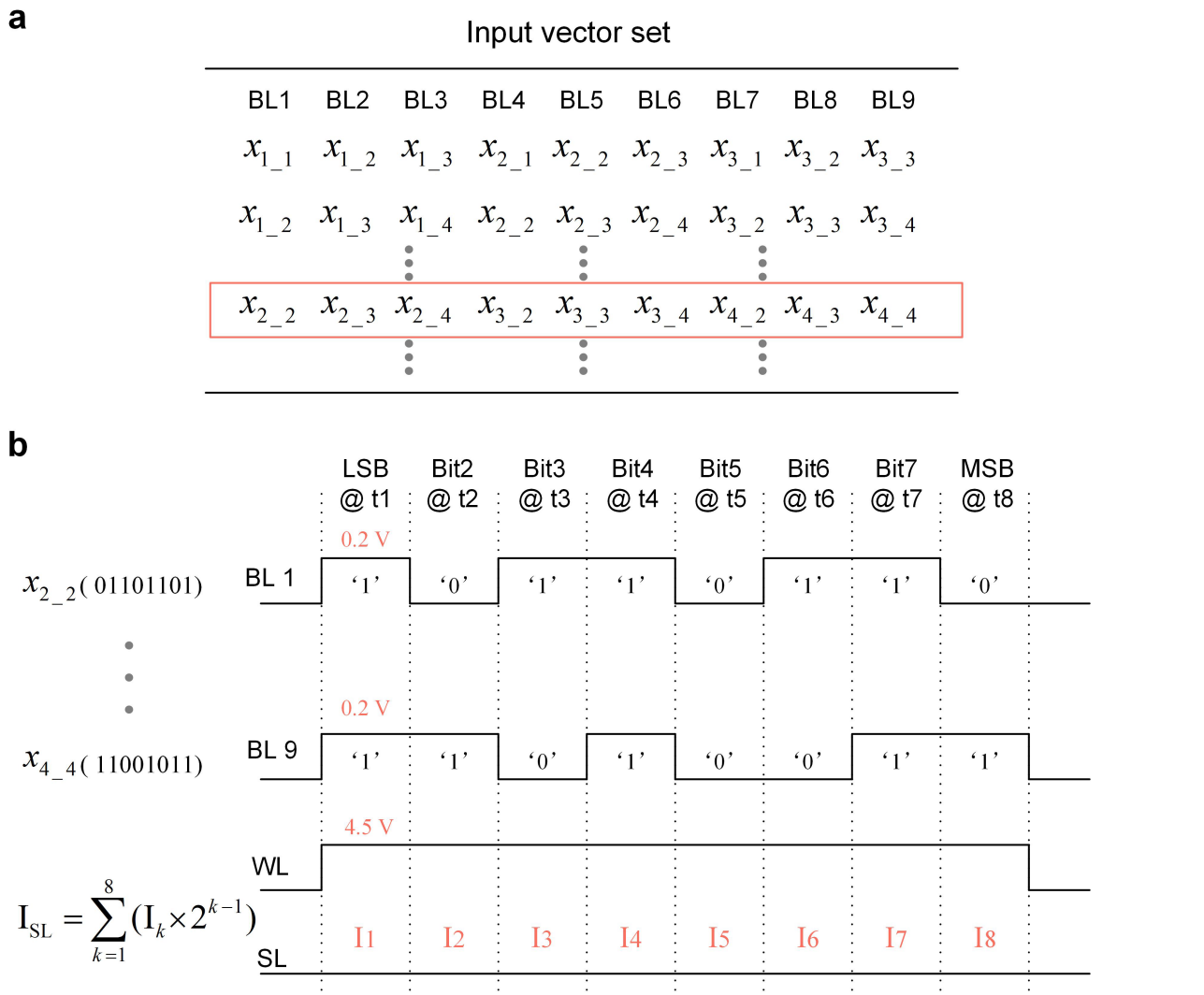
Supplementary information is available for this paper at https://doi.org/10.1038/s41586-020-1942-4.

Correspondence and requests for materials should be addressed to H.W.

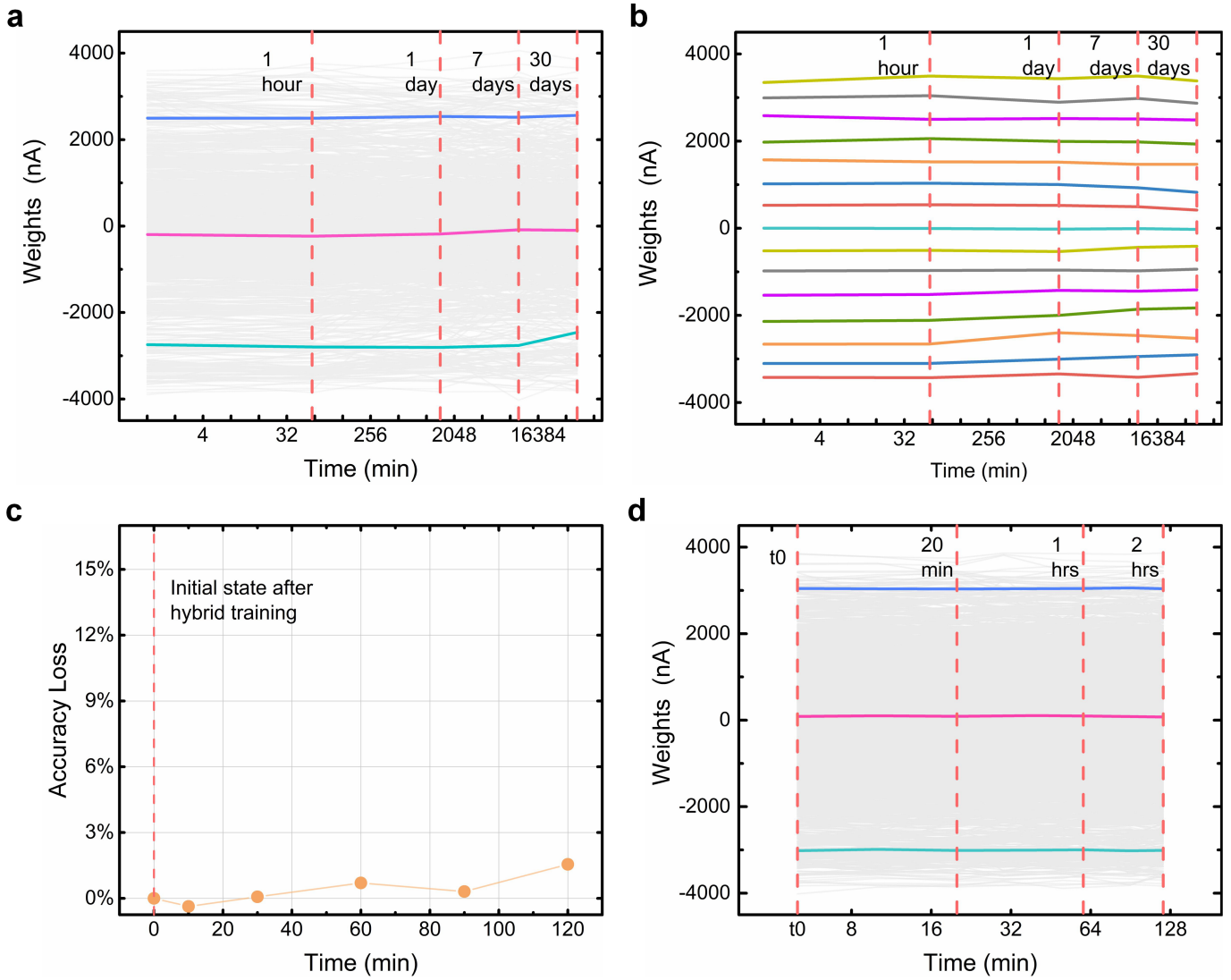
Peer review information *Nature* thanks Darsen Lu and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.



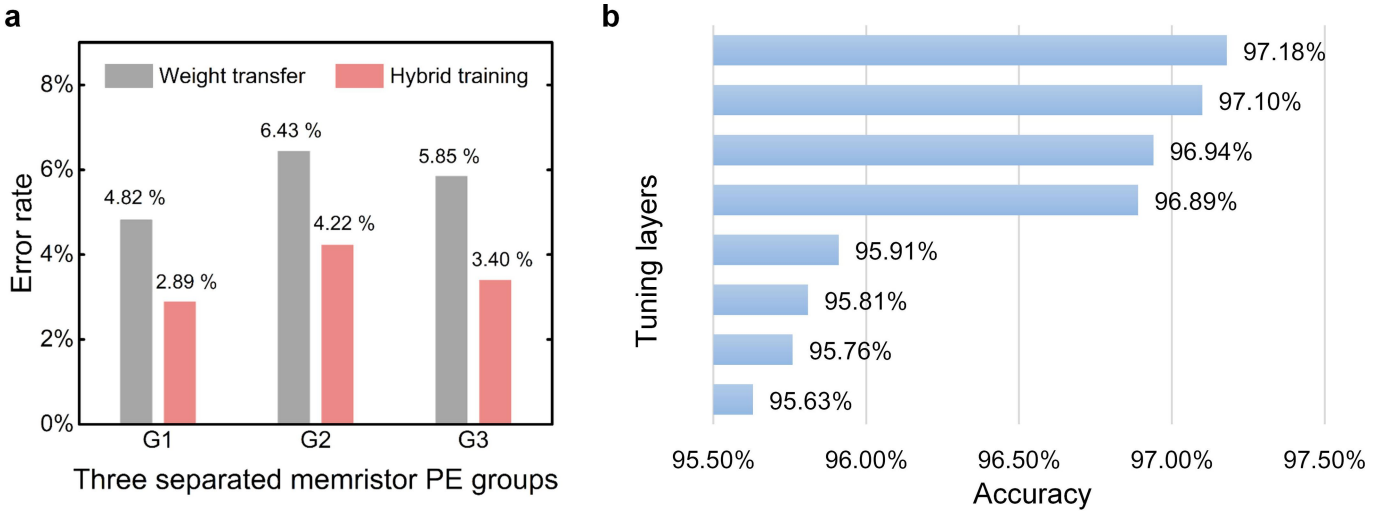
Extended Data Fig. 1 | Cumulative probability distribution of memristor conductance for the remaining seven 2,048-cell arrays. The red circles highlight abnormal data points, which deviate from their target conductance ranges owing to device variations. **a**, The 32-level conductance distribution on an entire 2,048-cell array. **b**–**g**, Conductance distributions for the first 32 rows of memristors (namely, 512 devices) for each of the remaining 2,048-cell arrays. A small number of writing errors were observed during the programming procedure (red circles), which could be attributed to device variations. These results show good consistency with Fig. 1c.



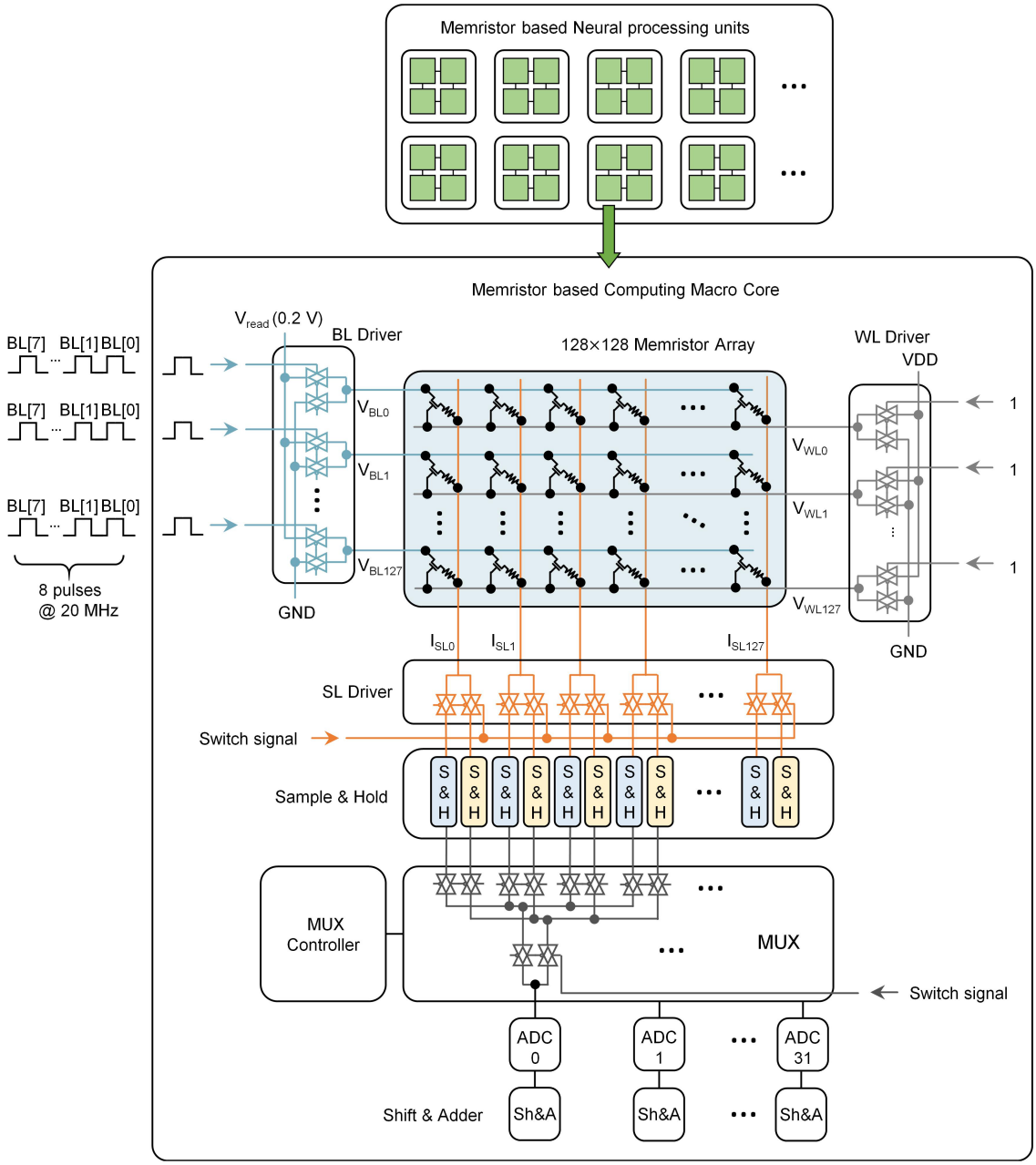
Extended Data Fig. 2 | Input patch set generated during the sliding process and input waveforms during the convolution. a, Input nine-dimensional vectors unrolled from the input 3 × 3 patch set. x_{m_n} indicates the relevant pixel at the crossing of row *m* and column *n*. The input patches are generated during the sliding convolutional process over the input feature planes and are subsequently injected into the memristor weight array. For a specific input vector, each element is encoded as the corresponding input pulse applied on the associated bit line. The red box indicates the current input vector, in agreement with the case illustrated in **b**. **b**, Input waveform sample in a memristor-based convolutional operation. Each element (an 8-bit binary number) in the input vector is encoded as sequential pulses over eight time intervals ($t1, t2, \ldots, t8$). For a particular period t_k, bit *k* determines whether a 0-V pulse or a 0.2-V pulse is used. Each '1' at a certain bit location implies the existence of a 0.2-V read pulse in the corresponding time interval, and a '0' indicates a 0-V read pulse. The corresponding output current I_k is sensed, and this quantized value is then left-shifted by $k-1$. Finally, the quantized and shifted results with respect to the same source line over the eight time intervals are summed together (I_{SL} in the inset equation). The difference between every two I_{SL} values from a pair of differential source lines is considered to be the expected weighted-sum result.



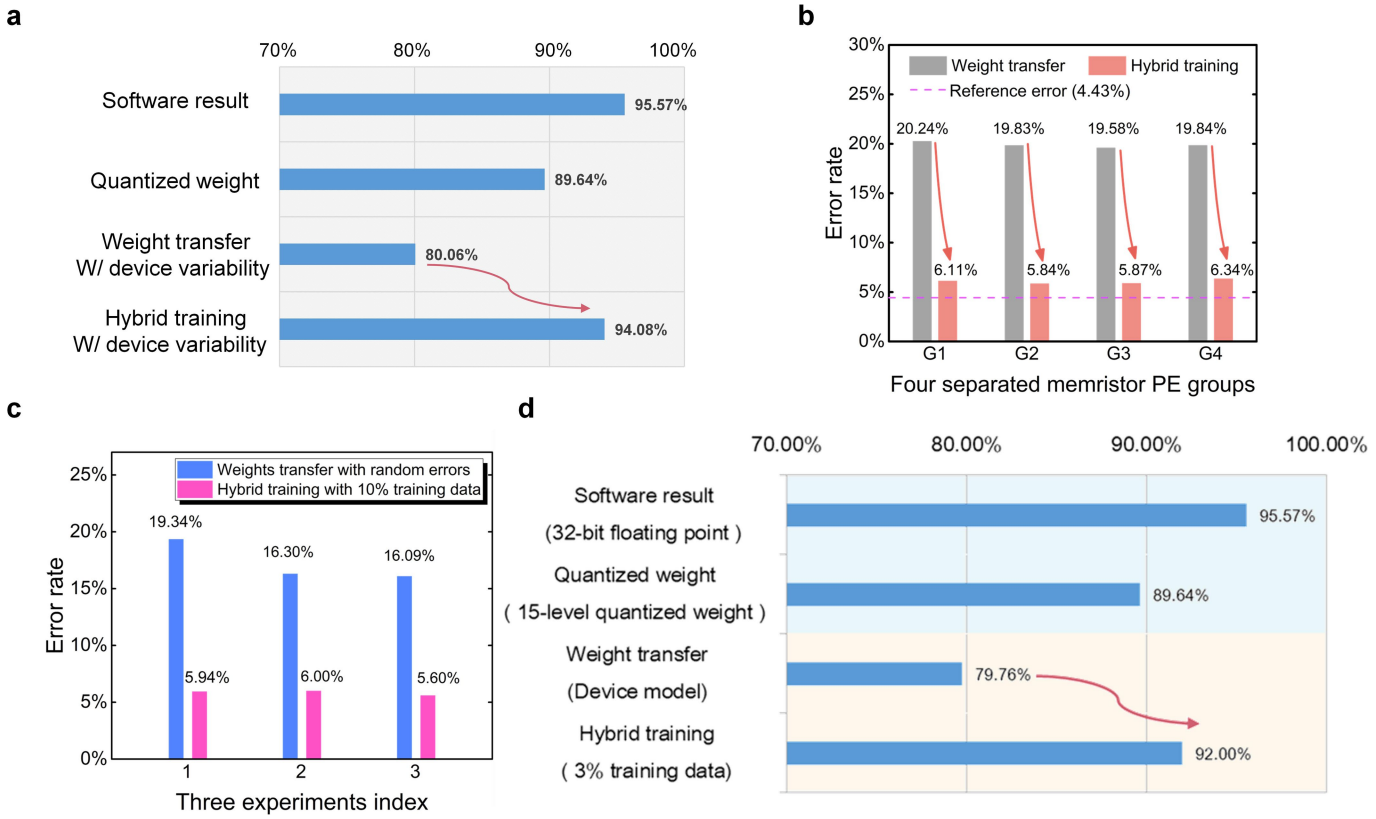
Extended Data Fig. 3 | Drift of conductance weights in time and associated degradation in system accuracy. a, Changes in the conductance weights with time, over 30 days after the transfer. The grey lines present the changing traces of all the cell weights, and the three coloured lines depict representative evolution trends. **b**, Mean weight value for the cells that belong to each of the 15 levels according to **a**. The 15 coloured traces show the 15 mean-value evolution traces as a function of time. **c**, Profile of accuracy loss during the experiment. The overall trend of the accuracy loss indicates how the conductance weight drifts deteriorate the recognition accuracy over time after hybrid training. Compared with the initial state, the recognition accuracy increases by 0.37% at $t = 10$ min, owing to random device-state fluctuations. **d**, Evolution of the weights of the weight cells considered in **c** over 2 h. t_0 denotes the moment when the hybrid training is completed. The grey lines show the changing traces of the states of the cells, and the three coloured lines depict representative evolution trends.



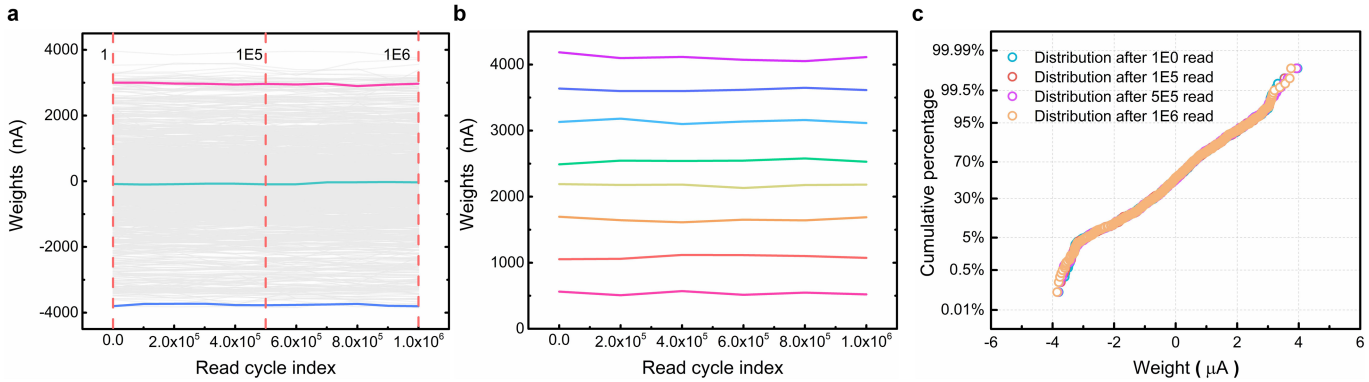
Extended Data Fig. 4 | Experimental accuracy of parallel memristor convolvers after hybrid training, and simulated training efficiency of different combinations of tuning layers. a, The error rate on the entire training set after hybrid training drops substantially compared with that achieved after weight transfer for any individual convolver group. The error rates with respect to the G1, G2 and G3 groups decrease from 4.82%, 6.43% and 5.85% to 2.89%, 4.22% and 3.40%, respectively, after hybrid training. **b**, Simulation results for all combinations of tuning weights for different layers using hybrid training and the five-layer CNN.



Extended Data Fig. 5 | Architecture of the simulated memristor-based neural processing unit and relevant circuit modules in the macro core.



Extended Data Fig. 6 | Scalability of the joint strategy. The joint strategy combines the hybrid training method and the parallel computing technique of replicating the same kernels. We show that a small subset of training data is sufficient for hybrid training. **a**, Recognition accuracies at different stages of the simulation process. During the simulation with ResNET-56, the kernel weights of the first convolutional layer are replicated to four groups of memristor arrays. **b**, After hybrid training the error rate on the test set drops substantially compared with that obtained immediately after weight transfer using each convolver group. **c**, The error rates drop considerably after hybrid training using 10% of the training data in the experiment with the five-layer CNN.The three experimental results show good consistency. **d**, Recognition accuracies at different stages of the simulation with ResNET-56. A high level of accuracy is achieved even when using 3% of the training data (1,500 training images) to update the weights of the FC layer. The mean accuracy for 10 trials is 92.00% after hybrid training, and the standard deviation is 0.8%.

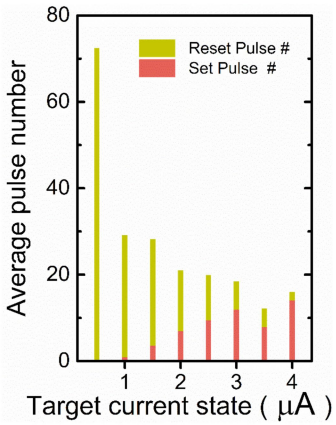


Extended Data Fig. 7 | Effects of read disturbance. To investigate this effect, we set up this experiment by writing all the convolutional kernel weights to two memristor PEs. After programming all the conductance weights smoothly, we physically apply 1,000,000 read pulses (0.2 V) on all weight cells to see how the read operations disturb the weight states. **a**, Changes in the states of the 936 conductance weightswhile cycling read operations. The grey lines give the changing traces of the states of all cells, and the three coloured lines depict representative evolution trends. **b**, Conductance evolution of eight memristor states during 10^6 read cycles. **c**, Distributions of weight states after 1, 10^5, 5×10^5 and 10^6 read cycles.

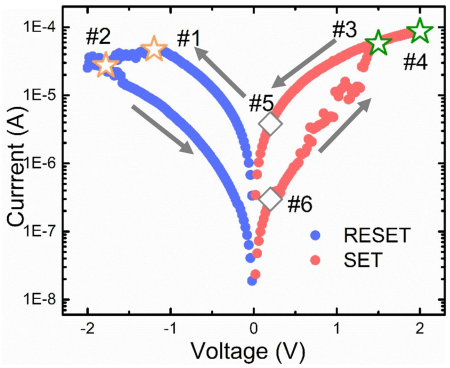
a

Group #	Target State / uA (0.2 V Read)	Average SET Pulse #	Average RESET Pulse #	Average Pulse # in Total
1	0.5 ± 0.1	0.2	72.2	72.4
2	1.0 ± 0.1	0.9	28.3	29.2
3	1.5 ± 0.1	3.6	24.6	28.2
4	2.0 ± 0.1	7.0	14.0	21.0
5	2.5 ± 0.1	9.5	10.4	19.9
6	3.0 ± 0.1	11.9	6.6	18.5
7	3.5 ± 0.1	7.9	4.3	12.2
8	4.0 ± 0.1	14.0	2.0	16.0

b



c



d

Data point Index	Description	Voltage (V)	Current (uA)
#1	RESET current	-1.2	45
#2	RESET pulse voltage	-1.8	25
#3	SET current	1.5	60
#4	SET pulse voltage	2.0	85
#5	READ @ LRS	0.2	4
#6	READ @ HRS	0.2	0.4

Extended Data Fig. 8 | Test results of the required programming pulse number and programming currents. a, Average pulse number required to reach each target conductance state. All the initial states were programmed to >4.0 μA. **b**, Stacked histogram distribution corresponding to the data in **a**. **c**, Current–voltage curve obtained during a d.c. voltage sweep. RESET and SET currents are measured at points #1 and #3, respectively. The conditions of RESET and SET pulses in this study are marked by points #2 and #4, respectively. Point #5 labels the read current at the low-resistance state (LRS) and point #6 labels the read current at the high-resistance state (HRS). **d**, Typical programming parameters. The programming current is 60 μA at 1.5 V during the SET process and 45 μA at −1.2 V during RESET.

Extended Data Table 1 | Detailed metrics of each circuitry module in the macro core with 1-bit input

Module	Area/um^2	Latency/ns	Energy/pJ
Array	1107.56	-	22.59
BL Driver	2104.36	0.002	1.99
WL Driver	2104.36	0.008	1.99
SL Driver	1686.3	0.38	0.11
Sample & Hold	10	-	0.13
Mux	1741.9	0.008	2.37
Mux Decoder	69.84	6.10	0.05
ADC	48000	-	326.4
Shift & Adder	6977.62	0.19	16.26
Sum	**63801.94**	**-**	**371.89**

BL, bit line; WL, word line; SL, source line.

Extended Data Table 2 | Benchmark metrics of a single macro core with 8-bit input

Performance	(128 × 128 × 2) ops / (8 × 50 ns) = 81.92 GOP s^{-1}
Power	371.89 pJ / 50 ns = 7.438 mW
Area	0.0704 mm^2
Energy efficiency	81.92 GOP s^{-1} / 7.438 mW = 11,014 GOP s^{-1} W^{-1}
Performance density	81.92 GOP s^{-1} / 0.0704 mm^2 = 1,164 GOP s^{-1} mm^{-2}

ops, operations.