Science
AAAS

Supplementary Materials for

Edge learning using a fully integrated neuro-inspired memristor chip

Wenbin Zhang *et al*.

Corresponding authors: Bin Gao, gaob1@tsinghua.edu.cn; Huaqiang Wu, wuhq@tsinghua.edu.cn

Science **381**, 1205 (2023)
DOI: 10.1126/science.ade3483

The PDF file includes:

Other Supplementary Material for this manuscript includes the following:

Materials and Methods

1. Algorithm for the STELLAR architecture

The STELLAR algorithm was developed to enable on-chip learning with memristors. It employs sign-based weight update calculations and utilizes a reconfigurable threshold in calculating the signs of errors. The STELLAR algorithm was designed as a generic approach to facilitate the implementation of on-chip learning with memristors for various neural networks. Considering that the manufactured memristor chip was designed to implement a two-layer neural network, the following explanation of the algorithm is based on this network. The details of the algorithm for other neural networks can be inferred from the following description.

A two-layer neural network with dimensions of 784 × 100 × 10 was constructed; it employed a rectified linear unit (ReLU) as an activation function for low-cost hardware implementation. Each weight value was represented as the differential conductance of a pair of memristors. The on-chip learning consisted of two stages: the forward and weight update stages (fig. S1B). The forward inference was conducted using the memristor crossbar arrays (Fig. 2A):

$$\boldsymbol{Y_1} = ReLU(\boldsymbol{W_1^T} \cdot \boldsymbol{X})$$

$$\boldsymbol{Z} = \boldsymbol{W_2^T} \cdot \boldsymbol{Y}$$

$$\boldsymbol{Y_2} = ReLU(\boldsymbol{Z})$$ (This step could be skipped)

where $\boldsymbol{X}$ is the input vector, $\boldsymbol{Y_1}$ is the 1st-layer output vector, $\boldsymbol{Z}$ is the 2nd-layer weighted-sum vector, $\boldsymbol{Y_2}$ is the 2nd-layer output vector (for the output layer, this step could be omitted according to the algorithmic configurations), and $\boldsymbol{W_1}$ and $\boldsymbol{W_2}$ are the conductance weight matrices of the 1st and 2nd layer, respectively. The weight update was performed for each input vector. The learning algorithm aimed to update the conductance optimally to minimize the loss function of the network output and target. The square loss function was used in this work; thus, the error vector was calculated as:

$$\boldsymbol{E} = \boldsymbol{T} - \boldsymbol{Y_2}$$ or

$$\boldsymbol{E} = \boldsymbol{T} - \boldsymbol{Z}$$ (W/O the ReLU at the output layer)

where $\boldsymbol{T}$ is the target vector, and $\boldsymbol{E}$ is the error vector. The signs of the errors were extracted to determine the weight update direction instead of the accurate values:

$$\boldsymbol{\Delta W_2} = \boldsymbol{SY_1} \cdot (\boldsymbol{SY_2}^{\boldsymbol{T}} \odot \boldsymbol{SE^T})$$ or

$$\boldsymbol{\Delta W_2} = \boldsymbol{SY_1} \cdot \boldsymbol{SE^T}$$ (W/O the ReLU at the output layer)

where $\boldsymbol{SY_1}$ is the sign vector of the 1st-layer output, $\boldsymbol{SY_2}$ is the sign vector of the 2nd-layer output, and $\boldsymbol{SE}$ is the ternary sign vector of the error. The elements sy_1, sy_2, and se in the sign vectors $\boldsymbol{SY_1}$, $\boldsymbol{SY_2}$, and $\boldsymbol{SE}$ were defined as follows:

$$sy_1 = \begin{cases} 1, & y_1 \geq C_1 \\ 0, & y_1 < C_1 \end{cases}$$

$$sy_2 = \begin{cases} 1, & y_2 \geq C_2 \\ 0, & y_2 < C_2 \end{cases}$$

$$se = \begin{cases} +1, & e \geq Th \\ -1, & e \leq -Th \\ 0, & else \end{cases}$$

where y_1, y_2, and e are the elements of vectors $\boldsymbol{Y_1}$, $\boldsymbol{Y_2}$, and $\boldsymbol{E}$, respectively. The C_1 and C_2 could be flexibly reconfigured. If the ReLU activation function was applied for the weighted-sum output $\boldsymbol{Z}$ vector, the C_1 value was typically the maximum among the 1st-layer outputs multiplied

by 0.4, and the C_2 value was typically set as zero. Meanwhile, if the ReLU function was omitted for the output layer, there would be no C_2, and the C_1 value typically amounted to zero. The threshold *Th* filtered out the small error values, i.e., $|se|<Th$, which prevented weight updates that were extremely sensitive and improved the convergence of the algorithm. Moreover, this whole STELLAR scheme could be extended to larger neural networks by quantizing the sy_1 and setting the threshold *Th* more flexibly.

The effectiveness of the STELLAR architecture does not rely on the specific forms of neurons and can be applied to various types of activation functions. In this work, the ReLU activation function was experimentally demonstrated with this fabricated chip because ReLU is widely adopted in current neural networks. When the neuron functions of the output layer are configured to sigmoid, tanh, etc., sy_2 will be quantized by the neuron's corresponding derivative function, instead of the value of y_2 directly. These only require standard CMOS circuitry engineering, and different kinds of neurons can be realized easily through lookup table circuits, and so on.

2. STELLAR update scheme under device asymmetric switching

Practical memristor devices suffer from non-ideal tuning behaviors, such as the nonlinearity and asymmetry of the updating curves, which hinder the development and exploration of memristor-based edge learning applications. The STELLAR update method introduced a threshold-based ternarized scheme to compute the input and output derivative vectors, which were subsequently used to calculate the sign-based weight gradients. This STELLAR scheme was beneficial to simplify the hardware design for gradient calculations and weight updates, saving hardware costs to implement a fully integrated memristor chip for edge learning. More importantly, this scheme allowed memristor chips to accommodate practical device asymmetric switching behaviors and to achieve a close approximation to the standard BP rule by omitting small weight updates with the threshold configurations.

To prove this, we first investigated how different device asymmetry factors affected the system accuracies. We established a device asymmetric switching model and chose three representative conductance-modulating curves under the pulse train (fig. S2A). Curve L1 depicted symmetric updating behavior, and curves L2 and L3 represented updating behaviors with typical and severe asymmetry, respectively. According to the test data, the memristor conductance could be modulated between 2 and 20 μS under an identical pulse train. Meanwhile, the model also incorporated device asymmetry under nonlinear conductance modulating procedure, and the conductance change ΔG regarding conductance state G was modeled and configured to simulate different memristor programming asymmetry factors. Device state variations were incorporated as Gaussian noises.

Next, we performed the simulation for MNIST image recognition with a two-layer MLP by training the weights of the last layer from random initial states. Based on the STELLAR architecture in Fig. 2A in the main text, we compared the test accuracies between zero-threshold configuration (i.e., the Manhattan rule or BP-without-verification scheme) and non-zero-threshold (i.e., the STELLAR scheme) after convergence. The result is illustrated in fig. S2B, indicating that the accuracies under the zero-threshold configuration were poor. The ideal behavior with linear and symmetric updating curve resulted in >15% accuracy drop for L1, and the device updating asymmetries for L2 and L3 led to additional 4% and 24% accuracy drops, respectively. However, by integrating on-chip learning with the STELLAR tuning scheme, the

recognition accuracy was significantly enhanced to nearly 95% for different device models, regardless of the asymmetric factor.

The system simulation results above suggested that the proposed STELLAR scheme could adapt to device asymmetric switching for a high accuracy. To understand the underlying mechanism of how the STELLAR scheme circumvented device asymmetric update, we further used a 100 × 10 memristor array to realize a perceptron model for an image classification task. During the simulation, we first randomly set ten different initial conductance distributions within the 100 × 10 array to obtain ten random initial weight distributions. Then, we fed 100 images to the array to tune the weights following the traditional stochastic BP scheme. During each iteration, the gradients of all the weights were normalized by the maximum value and recorded. The data in this flow were regarded as the ideal reference for the following simulations. Subsequently, we performed similar simulations with different memristor models and update schemes (i.e., the zero-threshold configuration and the STELLAR scheme) under the same initial weight condition. The weight gradients were also recorded during the 100 iterations, which were then used to calculate the mean square errors (MSEs) regarding the ideal reference results. The MSEs of the whole training were accumulated and then averaged over the ten random initial cases.

By investigating the MSEs under different update schemes and device asymmetries, we found that filtering out small weight changes below the threshold in the STELLAR scheme played an important role (fig. S2C). In this figure, we observed that the zero-threshold scheme suffered from device asymmetry, and a larger asymmetry factor could lead to a more serious MSE value (L3 > L2 > L1). The large MSE values indicated the gradient updates seriously departed from the BP learning trace. In contrast, the STELLAR scheme was immune to these non-ideal characteristics and achieved similar and small gradient MSE values, which decreased abruptly to 0.05 for all device models. The tendency of the changes of MSE values was well consistent with the evolution of system accuracies (fig. S2B) in terms of the effects of memristor asymmetry under zero-threshold configuration and the STELLAR scheme.

Theoretically, in the STELLAR scheme, filtering out small updates and keeping most memristor weights untouched played the most important role in approaching the BP learning trace. This approach ensured that the selected updated cells required relatively large conductance changes, and the asymmetric tuning was merely a second-order effect to cause update errors for these cells. This effect did not affect the gradient MSE value as the selected cell number was small. By omitting the small updates, the memristor-based gradient vectors under the STELLAR update scheme approximated the BP gradient vectors more closely.

In addition, omitting small updates did not significantly affect the accuracy. We initially investigated how the weight distribution changed before and after the gradient-descent-based update (i.e., distribution of "W_{after_update} - W_{before_update}) and inputted 100 images into a 100 × 10 memristor array to tune the weights using the traditional stochastic BP scheme. Figure S2D presents the corresponding distribution of weight changes. This figure demonstrated that a significant portion of weight updates clustered around zero, indicating that the cumulative training effects resulted in only slight updates for most weights. By training from identical initial weight distributions, the mean accuracy over five runs of simulation only experienced a slight drop of 0.04% when most small weight updates (>95% of all the updates) were omitted. Additionally, similar to the scheme of filtering small updates in the STELLAR scheme, we set a threshold for the conventional BP scheme to examine how excluding partial updates affected the final weight distribution. Comparing the final weight distributions after 100 iterations under the

conventional BP scheme with and without omitting small updates, as depicted in fig. S2E, we observed nearly identical distributions, further confirming that properly configuring the threshold to exclude small weight updates did not significantly impact accuracy.

The experimental simulation results together with the theoretical analysis suggested that the STELLAR scheme could accommodate the asymmetric update of the device.

3. Comparison between STELLAR and the conventional BP algorithm

We evaluated the performance of the STELLAR scheme and conventional BP method in an on-chip learning simulation with memristors. In the STELLAR architecture, the weight update calculation was accomplished through custom-designed circuits, and the conductance tuning operations were accomplished without verification. In the BP method, the weight update calculation was accomplished via simulation using an Intel Xeon E5-2699 processor, and the conductance tuning operations were accomplished by using a write-verify scheme. We estimated the accuracies and energy consumption yielded by using the same training task for MNIST image classification. An MLP with dimensions of $784 \times 100 \times 10$ was utilized when comparing these methods. We used a behavioral memristor device model in the simulation of conductance tuning operation, and it provided a final conductance distribution after executing a SET/RESET pulse on a given initial conductance (*30*). In the BP method, the memristor was programmed to a certain margin with the target conductance as the margin middle value and a given variation as the margin upper/lower limit. The value of the variation was constant for all the conductance states and was defined as a certain percentage of the full conductance window.

We focused on the energy consumption of the weight update stage. The energy consumption required by the STELLAR architecture involved a weight update calculation utilizing custom-designed circuits and a one-time conductance tuning operation without verification. The energy consumption required by the BP method involved a weight update calculation utilizing the CPU and multiple conductance tuning operations with verification, requiring the ADC to verify the device conductance during the tuning process. We combined the experimental and simulation data to estimate the energy consumption. The energy consumption of the custom-designed circuits was obtained from the corresponding circuits used in our memristor chip with the Cadence simulator. The energy consumption of the Intel Xeon E5-2699 processor was estimated as the number of calculation operations divided by its energy efficiency (*44*). The total energy consumption of the conductance tuning operations during the entire training process was estimated as the number of tuning operations multiplied by the average energy of each operation. The number of tuning operations was derived from the on-chip learning simulation. The average energy of a write operation was estimated from the measured results yielded by our memristor chip, and the energy of a read operation was obtained from a 130-nm ADC with an 8-bit resolution (*45*).

4. Endurance requirements for the cycle-parallel conductance tuning scheme

During the procedure of cycle-parallel programming, only one programming operation (i.e., SET or RESET) was employed to tune the memristor cell of positive or negative polarity in the 2T2R weight unit to increase or decrease the weight value accordingly, instead of using the typical all-cell-updating method. This could lower the endurance requirement. We compared the specified endurance requirement in different updating schemes, i.e., cycle-parallel STELLAR updating scheme (denoted as STELLAR 1 in fig. S4), non-cycle-parallel STELLAR updating scheme (which meant both memristor cells in each 2T2R unit needed to be programmed for each

iteration, represented by STELLAR 2 in fig. S4), and conventional BP with write-verification scheme (BP w/ verify in fig. S4). During this experiment, we set a constant training epoch of three and counted the necessary pulse number during tuning the memristor weights in the whole training phase. This process was repeated ten times to obtain the final statistical results (fig. S4). Figure S4A presents that similar inference accuracies were achieved after training, and fig. S4B reveals the corresponding pulse number applied to tune the weights.

Cycle-parallel STELLAR scheme could converge as fast as conventional BP with write-verify scheme within the same training epochs; however, conventional BP with write-verify scheme required more programming pulses to program the 2T2R to the target value in each iteration and led to the strict endurance requirement of memristor device. On the other side, cycle-parallel conductance tuning consumed fewer update pulses by filtering the programming procedure for cells with small residual errors. Compared to non-cycle-parallel operation, this scheme could fully benefit from the bidirectional analog switching characteristic of memristors to save nearly half the pulses by merely updating one memristor in the 2T2R weight unit.

5. Weight configuration based on the 1T1R memristor array

Each synaptic weight was represented with a differential pair of memristor cells. The positive and the negative cells were mapped to the adjacent columns along the same row in the 1T1R array. During the weight transfer stage, if the synaptic weight was positive, the negative cell was programmed to the lowest conductance state (g_{min}), and vice-versa. Mathematically, the conductance of positive and negative cells is given as follows. For positive weights:

$$\begin{cases} g_+ = \dfrac{w}{w_{max}}(g_{max} - g_{min}) + g_{min} \\ g_- = g_{min} \end{cases}$$

and for negative weights:

$$\begin{cases} g_+ = g_{min} \\ g_- = \dfrac{|w|}{w_{max}}(g_{max} - g_{min}) + g_{min} \end{cases}$$

where g_+ and g_- are the conductance of the positive and negative cells; g_{max} and g_{min} are the maximum and minimum conductance of the memristors; $w_{\max}$ is the maximum absolute value of weights; w is the synaptic weight.

6. Circuit design of the memristor chip

The STELLAR algorithm was modified for its chip implementation (fig. S1C). The selection of the threshold was realized by configuring the output scales and target values. It was predefined to take the first 8 bits of 17-bit signed errors $\boldsymbol{E}$ when calculating the ternary signs $\boldsymbol{SE}$. Consequently, the selection of the threshold was equivalent to adjusting the error scale, which was accomplished by adjusting the output scales and target values. The output scales of $\boldsymbol{Y_1}$ and $\boldsymbol{Y_2}$ could be modulated with the resolution-adjustable ADC, whose quantization scale was determined by its resolution.

The on-chip learning of the memristor chip involved three working stages: Forward (FWD), SET, and RESET (fig. S1B). The controllers decoded the input stage selection signals to the output signals for voltage selection. The voltage selection signals were taken as the control signals of the BL/WL and SL drivers; the voltage to be applied to the memristor array was selected through the multiplexers in the drivers.

In the FWD stage, vector-matrix multiplication (VMM) operations were performed with the memristor arrays and on-chip ADCs. The ADCs could be configured to implement the ReLU function along the quantization of the analog VMM signal. The 1st-layer array utilized the 2T2R configuration to reduce the IR drop. The 2nd-layer array utilized the 1T1R configuration with the two adjacent columns representing positive and negative weights, respectively. The resolutions of the ADCs were adjustable, enabling the flexible adjustment of the output scale. A timing-dependent scheme based on the number of pulses was used in this memristor chip. If the resolutions of the 1st-layer ADCs were configured as N bits, 2^N pulses were generated. For each pulse cycle of the 1st-layer output, the 2nd-layer performed one VMM. The outputs of the 1st-layer ADC were stored in 1-bit registers and sampled simultaneously by the next memristor array as inputs. The subtraction of the 2nd-layer VMM outputs of differential column pair was accomplished with the logics connected to their associated ADCs, and the subtraction results (i.e., the output vector of $\boldsymbol{Y_2}$) were sampled simultaneously by counters and stored in output buffers. The error calculation (i.e., the error vector of $\boldsymbol{E}$) between the output and target was accomplished simultaneously with another set of counters, which used the preloaded targets as their initial values.

The signs of the 1st-layer outputs ($\boldsymbol{SY_1}$) were obtained in the FWD stage by comparing the 1st-layer outputs ($\boldsymbol{Y_1}$) with the reconfigurable value C_1. Since the memristor chip utilized a pulse-based coding scheme, this function could be realized with an AND logic gate, which takes the 1st-layer outputs and the reconfigurable value as inputs. The reconfigurable value signal was high during the C_1^{th} cycle of the 1st-layer output. In this cycle, the AND logic generated the sign signals of $\boldsymbol{SY_1}$, which were stored in the registers.

The SET and RESET stages involved determining the weight update direction and performing conductance tuning operations. The conductance tuning was performed in a row-by-row fashion, with a set of shift registers selecting which row to update. The calculation of the weight update direction involved the signs of the 1st-layer outputs, 2nd-layer outputs, and errors, where the first two groups of signs (i.e., the $\boldsymbol{SY_1}$ and $\boldsymbol{SY_2}$) were obtained in the FWD stage and stored in registers. The first 8 bits with the highest significance of error $\boldsymbol{E}$ were used to evaluate $\boldsymbol{SE}$ for calculating the final weight update $\boldsymbol{\Delta W_2}$.

A 2T2R memristor array was proposed to reduce the IR drop by decreasing the SL output current. Voltages with opposite polarities were applied to the two memristors in the differential pair during inference, where the conductance of these two memristors had different polarities, namely positive and negative polarity. The current through the positive and negative cells connected to the same SL could be canceled out locally. This direct subtraction in the current domain reduced the IR drop, which decreased the power consumption and boosted the computing parallelism.

The resolution-adjustable ADC consisted of three submodules: integrator, comparator, and shared digital-to-analog converter (DAC) for each layer. First, the integrator converted the SL current to a voltage. Then, the DAC generated a voltage that increased with the clock pulse. Simultaneously, the comparator compared the output voltages of the integrator and DAC and generated a timing-dependent quantized output, represented by a sequential train of pulses. The resolution of the ADC could be configured by setting the clock frequency of the DAC and the sampling frequency of output pulses. The resolution-adjustable ADC enabled a flexible trade-off between system accuracy and power consumption.

7. Fabrication of the memristor chip

The memristor arrays were monolithically integrated with CMOS peripheral circuits on a chip. Each memristor device was connected in series with a selector transistor, whose gate was taken as the effective control terminal during programming. As shown in Fig. 3C, the memristor device was sandwiched between metal-4 and metal-5 interconnects with a material stack of TiN/HfO_x/TaO_y/TiN.

The selector transistor, peripheral circuits, and bottom four metal interconnections were fabricated in a standard CMOS foundry by using a 130-nm process. The memristors and top metal interconnections were fabricated in a laboratory with a back-end-of-line process. First, the TiN bottom electrode layer, HfO_x switching layer, TaO_y capping layer, and TiN top electrode layer were sequentially deposited on the wafers from the foundry. The capping layer served as a thermally enhanced layer to modulate the distributions of the electric field and temperature in the switching layer and contributed to the improved analog switching behavior. The deposited layers were patterned with lithography and etching to form isolated 0.7 μm × 0.7 μm memristor cells. Subsequently, a SiO_2 dielectric was deposited, polished, patterned, and etched, forming vias for interconnects. Finally, the top vias were formed by sputtering tungsten and conducting chemical mechanical polishing, and the top metal interconnections were formed by sputtering aluminum and shaping it, which completed the fabrication process.

8. Measurements of the memristor devices

The test system for the memristor chip mainly integrated a field-programmable gate array (FPGA) and relevant voltage generators. The FPGA generated control commands for, sent inputs to, and read results from the memristor chip. The voltage generators provided varying voltages for the VMM operation and memristor programming. The FPGA communicated with the host PC that realized the user interface through an Ethernet connection. To facilitate the implementation of various learning tasks on the memristor chip, we mapped the high-level functions on the PC, provided as application programming interfaces (APIs), to the fundamental operations of the memristor chip. The APIs included various programming schemes, and inference/learning function on a given dataset.

The memristor device had a high yield for 5-bit (32-level) programming, with a minimum success rate of 99.69% across all the programmed conductance states (fig. S6C). During the test, ~160K memristor devices were programmed to 32 target conductance states, with ~5,000 devices assigned to each state. The conductance targets ranged within a switching window from 2 to 20 μS with a uniform interval of 0.58 μS. The margin was set as ±0.24 μS for each target conductance state. The memristor conductance was sensed with on-chip ADCs at a 0.2 V read voltage after each programming pulse.

Another 1K 1T1R memristor array was on the same wafer with the neuro-inspired computing chip, and it was used to characterize the analog switching behavior of the memristor device. Sixty-four memristor cells were measured for five cycles, consisting of alternated potentiation and depression. Potentiation referred to the evolution of conductance under an identical SET pulse train, which contained 128 SET pulses with 1.5 V and 50-ns pulse widths. Depression referred to the evolution of conductance under an identical RESET pulse train, which contained 128 RESET pulses with 1.4 V and 50-ns pulse widths. The bidirectional analog switching of the fabricated memristor device expanded the weight tuning range to avoid conductance saturation during edge learning. The memristor conductance could be gradually tuned to increase or decrease the weight even at the edge of the switching window.

9. Off-chip training and on-chip inference

We injected noise into the weights during off-chip training to improve the resilience of the network against nonidealities (*46*) in these tasks. The weights of the off-chip-trained model were transferred to the chip as the memristor conductance during the weight transfer stage. Binarized images with resolutions of 28 × 28 pixels obtained from the MNIST dataset were used as the input images. The inference was performed on 10,000 images derived from the entire test set. We investigated the change in accuracy with time by carrying out an on-chip test. The inference was performed five times each day on the entire test set within 48 days after weight transfer.

10. Implementation of the on-chip learning task

We used the MNIST dataset to demonstrate the on-chip learning task. The 1^{st}-layer weights of the off-chip-trained model were transferred to the 1^{st}-layer memristor array as conductance. The 2^{nd}-layer memristors were all programmed to the HRS via unidirectional verification, and the programming noise worked as the random initialization of the 2^{nd}-layer weights. The on-chip learning was performed during 180,000 training iterations (three epochs with the entire training set containing 60,000 images), and the inference accuracies were obtained on the 10,000 randomly selected images from the training set and all the 10,000 images from the test set, respectively.

11. Energy consumption benchmark

We measured the latency and power consumption of the memristor chip in the FWD, SET, and RESET stages. The total energy consumption for one learning iteration was the sum of the energy consumption of the forward (FWD) stage and weight update stage, where the energy consumption of the weight update stage was the average energy consumption of the SET and RESET stages. From the above measurements and calculations, we obtained that the energy consumption of our memristor chip for each on-chip learning iteration was 1.002 μJ. The detailed metrics, including the latency, power, and energy consumption of each stage in the learning mode, are listed in table S1. Instead of assessing the system as a whole, we broke down the system performance according to the cost of the subsystems to give more details and insights. The latency was recorded by capturing the duration of the flag signal on the oscilloscope for each stage. The voltage and current were carefully measured for each power domain to estimate the power dissipation, and the energy consumption was obtained based on the working time slot and corresponding power. Notably, on account of the different SET (200 ns) and RESET (50 ns) pulse width configurations in the proposed cycling parallel update, the update stage consumed an average of 85.95 and 55.95 μs for the SET cycle and RESET cycle, respectively.

We also estimated the energy consumption and computing latency of a typical digital accelerator-based (HNPU-based) system for each training iteration in the improvement learning task. This computing system generally consists of a processor (HNPU) and an off-chip main memory. During each training iteration, the core calculation operations, and memory-fetching operations both contributed to the overall energy consumption and latency. For the calculation process, the energy consumption (or computing latency) was obtained by dividing the total number of operations by the energy efficiency (or computing power) of the processor (*36*). For the memory-fetching process, the energy consumption was obtained by multiplying the data size and the energy consumption per bit, and the latency is obtained by dividing the data size by the memory bandwidth (*47*). Following this method, the energy consumption and the computing latency of the HNPU-based system could be optimistically estimated (assuming a 100%

hardware utilization rate) with the detailed parameters in table S2. Specifically, when completely running the training task within the fabricated memristor chip for MNIST image recognition, the total number of operations was 0.16 M and the transferred data size was 80.40 KB, respectively.

We assumed that an 8-bit weights configuration is used, appropriate for these kinds of edge training tasks. The HNPU-based system utilized DDR4 DRAM as external memory (*36*), so the DRAM writing/reading energy was assumed to be 45 pJ per bit based on the reported data for DDR4 (*48*). The on-board interconnect energy was assumed as 10 pJ per bit (*49*). Hence, the estimated energy consumption and computing latency of the HNPU-based system for each training iteration was 35.40 μJ and 4.24 μs, respectively. We noted that the estimated energy consumed by the HNPU-based system was 35 times larger than that of the memristor chip.

12. Motion control task of learning new samples

We demonstrated the improvement learning of new samples in the motion control task of a light-chasing car, whose control algorithm involved mapping a camera image to a control decision (i.e., a steering angle and a driving throttle) directly in an end-to-end learning approach (*50*). As illustrated in fig. S7A, a traditional approach was developed to generate control decisions for teaching neural networks and evaluating their performance. The traditional approach divided the control task into two parts, including laser spot detection and control logic generation. The camera images and corresponding control decisions were collected as datasets when the traditional approach drove the car. An end-to-end neural network was then trained with the collected samples to directly turn the images into control decisions. The obtained network was deployed on the car to drive it autonomously in real environments. In this task, the steering angles are analog signals that range from -1 to 1, corresponding to different turning angles of the steering motor from the leftmost to the rightmost. The driving throttle was a binary signal corresponding to the driving motor moving forward or stopping.

A multitask CNN was constructed, including convolutional layers and two FC layers, to realize the control of the light-chasing car. The dimensions of the last two FC layers were 512 × 100 × 10 to fit the array size of the fabricated memristor chip. Eight of the ten outputs predicted a regression value for the continuous steering angle, and the other two outputs classified the driving throttle, indicating whether to move or stop. The FC layers were implemented on the chip, and the convolutional layers were implemented with software. The on-chip learning was performed on 2,100 samples collected from the bright scene (i.e., the training dataset for this improvement learning task). The inference was performed on the test dataset of this improvement learning task, including 4,502 samples acquired from the dark scene and 900 samples collected from the bright scene.

A customized scoring mechanism was developed to evaluate the performance of the neural network. The rule for scoring each action (with the steering angle and driving throttle combined) is shown in fig. S7C. The final score of the neural network was the summation of the scores over the entire test dataset of the network normalized by that of the traditional approach. The snapshots of real-time driving with the memristor chip are shown in fig. S7D.

13. Image classification task of learning a new class

We demonstrated the improvement learning of a new class in the MNIST image classification task, which involved images of the digits 0 and 2–9 for off-chip training and images of the digit 1 for on-chip learning. A multilayer neural network with dimensions of 784 × 100 × 9 (Fig. 4F) was constructed during off-chip training, and this network was then transferred

to the memristor chip during the weight transfer stage. The neural network was expanded to the dimensions of 784 × 100 × 10 with a newly added output neuron for the new class. The expanded parts of the conductance weights were programmed to the HRS at the differential rows for the new class with single-side verification. Only the expanded conductance weights were updated, and the old conductance weights were frozen during on-chip learning. The on-chip learning was performed on 150 images (i.e., the training dataset for this improvement learning task) randomly selected from the training set of the total 6,742 "digit 1" images. The inference was performed every ten training iterations based on the test dataset of this improvement learning task, including the 1,000 randomly selected images from the test set of the old classes and 1,135 images from the test set of the new class.

14. Audio recognition task of learning new samples

We demonstrated the learning of new samples in a speech recognition task conducted on the AudioMNIST dataset (*51*), which consists of male and female audio samples of spoken digits (0–9). The dataset consists of 30,000 audio samples of spoken digits (0–9) provided by 48 men and 12 women, with each speaker saying each digit 50 times. Mel-frequency cepstral coefficients (MFCCs) were used as the features for the audio samples, and a two-layer perceptron with dimensions of 392 × 100 × 10 was used to classify the MFCC input. As illustrated in fig. S8A, the improvement learning of female audio samples was based on the model trained with male audio samples. First, the base model was trained off-chip with the 6,000 samples of the first 12 men and then deployed on the memristor chip. Next, the improvement learning of new samples was performed on the chip with 5,400 samples. These samples comprised the first 45 audio samples for each woman and formed the training dataset for this improvement learning task.

Figure S8B shows the evolution trends of the model accuracies for male and female audio during improvement learning. The on-chip inference was performed on the unused samples (i.e., the test dataset for this improvement learning task), which included 18,000 samples of the last 36 male and the remaining 600 female audio samples. After improvement learning with the female audio samples, the accuracy of the model for female audio increased from 85.7% to 89.5%, and the accuracy of the model for male audio slightly decreased from 91.4% to 91%.

15. Motion control task of learning a new class

We demonstrated the learning of a new class in another motion control task involving the abovementioned light-chasing car platform. In this task, improvement learning aimed to make the car learn to move backward, and both the steering angle and driving throttle were taken as the classification results of the neural network. As illustrated in fig. S9A, based on the developed model, the improvement learning was triggered once in the case of an unreachable situation. The unreachable situation refers to the data whose camera images have a light spot at their two bottom corners. In this case, the car would not reach the target (i.e., light spot) even though it turned the steering motor to the rightmost (or leftmost) setting and moved forward until it stopped. After improvement learning of the unreachable situation, whose driving throttle was labeled "move backward", the car learned to move backward to align itself to the target in this case. The behaviors of the car before and after improvement learning are illustrated in fig. S9B as the trajectories of the weight transfer and improvement learning, respectively, and are shown in Movie S2. A sequence of camera images corresponding to the trajectory after improvement learning is shown in fig. S9D. The driving throttle classification accuracies achieved for the old classes and the new class over the course of the improvement learning are shown in fig. S9C.

16. The hybrid system for running the motion control task

The CNN network used for the motion control task consisted of six convolutional layers and two FC layers (fig. S7B). Following the feature representation, each convolutional layer also included a batch normalization layer, a ReLU activation layer, and an average pooling layer. The output feature maps ($4 \times 4 \times 32$) of the last convolutional layer were finally unrolled to a 512-dimension vector, which was input to the last two FC layers to generate 10-dimension control signals.

To deploy such a network with a memristor chip for motion control, the CNN model was first trained off-chip with the collected dark scene data on conventional hardware. To alleviate the effects of device non-ideal characteristics on network accuracy, Gaussian noise was added to the weights to adapt to memristor conductance variations and drifts. Subsequently, the weights of these two FC layers were transferred to the two memristor arrays on the chip, and the convolutional layers were implemented with software due to the limited on-chip memristor capacity. Next, improvement learning on the new image (i.e., bright scene data) was performed on the chip by tuning the weights of the last FC layer. Figure S7B illustrates this whole diagram and specifies where the corresponding calculations happened. During each iteration of improvement learning, the forward phase of the convolution operations was processed with the help of a software, and the weight update process, including weight update calculation and conductance tuning, was completely realized through the memristor chip.

17. The scalability of the method to larger neural networks.

To validate that the STELLAR architecture could scale up for a larger network and realize software-comparable accuracy, we performed the simulation with a memristor ResNet20 network for CIFAR-100 image recognition. Referring to a typical improvement learning algorithm of PackNet (*52*, *53*) with floating-format weights and activations, we simulated a ResNet20 model (fig. S10A) for the CIFAR-100 image recognition with the STELLAR architecture and memristor model. The PackNet model (*53*) was constructed based on the ResNet20 to perform the improvement learning task on the CIFAR-100 dataset. This improvement learning task required 20 unseen classes to be learned after training on the remaining 80 categories from the CIFAR-100 dataset. The accuracies on the old, new, and whole datasets for this task were recorded as the baseline. A similar configuration was adopted in our simulation, despite that we set the network weights to 4 bits, activation outputs to 8 bits, and added a 5% weight noise to model the variations of memristor conductance.

The overall diagram of the improvement learning process is shown in fig. S10B. We used the first 80 classes' images (i.e., the old data) of the CIFAR-100 for offline training, and took the images of the last 20 classes (i.e., the new data) for improvement learning. During the offline training on the old data, weights and activations were quantized, and the weight noises were applied accordingly. Then, we simulated the weight-transferring process by adding Gaussian noises to the well-trained weights.

Next, the improvement learning proceeded by initializing the weights in the last FC layer for the recognition of the 20 new classes. These weights were randomly assigned by sampling from the weight distribution of the last FC layer for the 80 old classes. Then, a set of images comprising the old and new data was fed into the network to update the FC weights. Specifically, only the weights attributed to the 20 new classes were tuned following the STELLAR process,

and the remaining weights were frozen. During the weight update process of each iteration, the cycling parallel strategy was adopted.

During the simulation, the evolution of recognition accuracies on the old classes, new classes, and total classes are shown in fig. S10C. The accuracy on the old dataset gradually decreased, and the accuracy on the new data significantly increased along with the evolution of the training process. This process finally converged to an equilibrium where the accuracies on both the old and new datasets matched, realizing sufficient knowledge learning on the new data and negligible information loss on the old data.

Ultimately, the memristor-featured STELLAR architecture showed accuracies of 66.50% on the original dataset, 65.30% on the new dataset, and an overall average of 66.08% after improvement learning (fig. S10D). Compared to the baseline of the PackNet model with full-precision weights, the simulated overall accuracy slightly dropped by 1.42%. These results illustrated that the proposed STELLAR architecture could support more sophisticated networks and realize efficient improvement learning with high-precision software accuracy.

18. The energy efficiency estimation of the memristor-based learning chip

To precisely estimate the system performance for improvement learning, we established a cycle-accurate SystemC chip model for this memristor-featured STELLAR architecture (fig. S11). This tiled memristor chip model mainly consisted of CIM tiles, and each CIM tile integrated a distributed SRAM buffer, several memristor-based processing elements (PEs), and necessary digital function blocks, such as the Round Robin (RR) prioritizer, and a special function unit (SFU) for realizing activation and pooling functions. The chip also had a router block and a global controller in its totality. In each PE, the required ADC, DAC, analog buffers, drivers, and memristor array were included. In addition, the weight update unit was also added to calculate the gradient errors and control the update flow, lying alongside each CIM tile. Owing to the STELLAR scheme, these simplified update blocks contributed to negligible area and power.

The module metrics regarding the analog circuits and digital circuits were extracted from SPICE netlist simulation under UMC 28-nm technology. The SRAM buffer cost was obtained using the memory compiler. We configured this architecture to run a standard ResNet18, which features 3.6 GOPs and 11.6-M weights. The architecture setting and PE parameters are shown in detail in table S3.

Using the 8-bit activation precision and the straightforward mapping strategy, the chip model was benchmarked to run the improvement learning task in an end-to-end manner based on the ResNet18 network. The simulated chip could achieve a latency of 0.87 ms for each image, denoting a frame rate of 1,152 frames per second. The frame rate could be further improved with optimized mapping and schedule schemes, such as replicating the weights to multiple PEs to speed up the system. At the same time, the energy efficiency was as high as 18.5 TOPS/W (4-bit input, analog weight, and 8-bit output), which was about seven times higher than that of HNPU (2.64 TOPS/W for 8-bit operations). Moreover, the proposed STELLAR scheme is fully compatible with present optimizations in memristor CIM inference accelerators. Hence, the energy efficiency could reach above 200 TOPS/W (*41*) by further optimizing the memristor array structure and peripheral circuits. These results prove that the memristor-based learning chip could achieve higher energy efficiency than a digital accelerator.

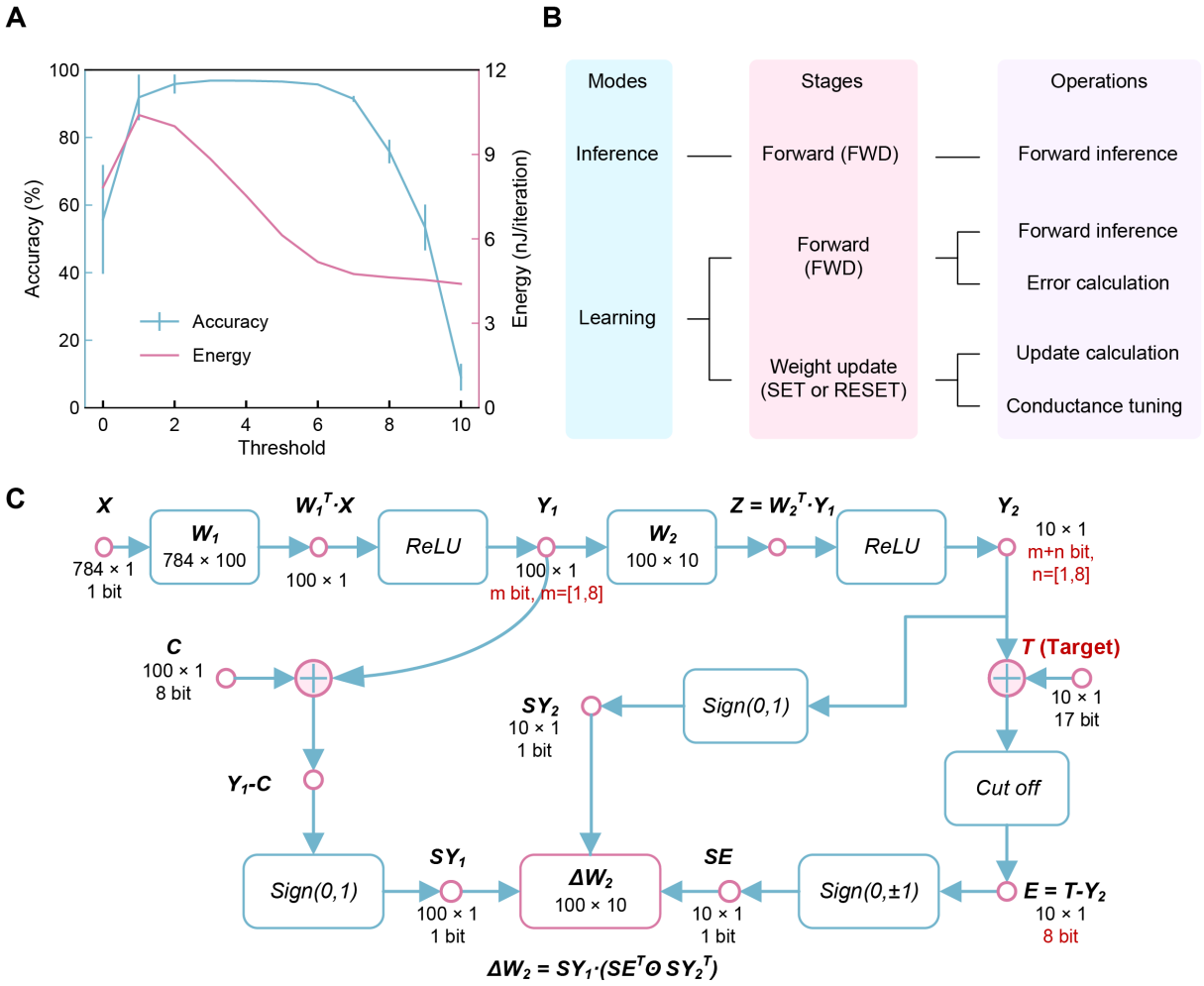

Fig. S1. Details of the algorithm for the STELLAR architecture. (**A**) The role of threshold in the STELLAR algorithm, showing the effects of different thresholds on classification accuracy and energy consumption of on-chip learning. (**B**) Dendrogram of the chip's working modes and the corresponding stages and operations. The forward stage involves a forward inference operation and an error calculation operation. The weight update stage involves a weight update calculation and a conductance tuning operation. (**C**) Schematic of the modified algorithm for chip implementation. m and n, ranging from 1 to 8, represented the resolutions of the ADCs of the 1st and 2nd layers, respectively.

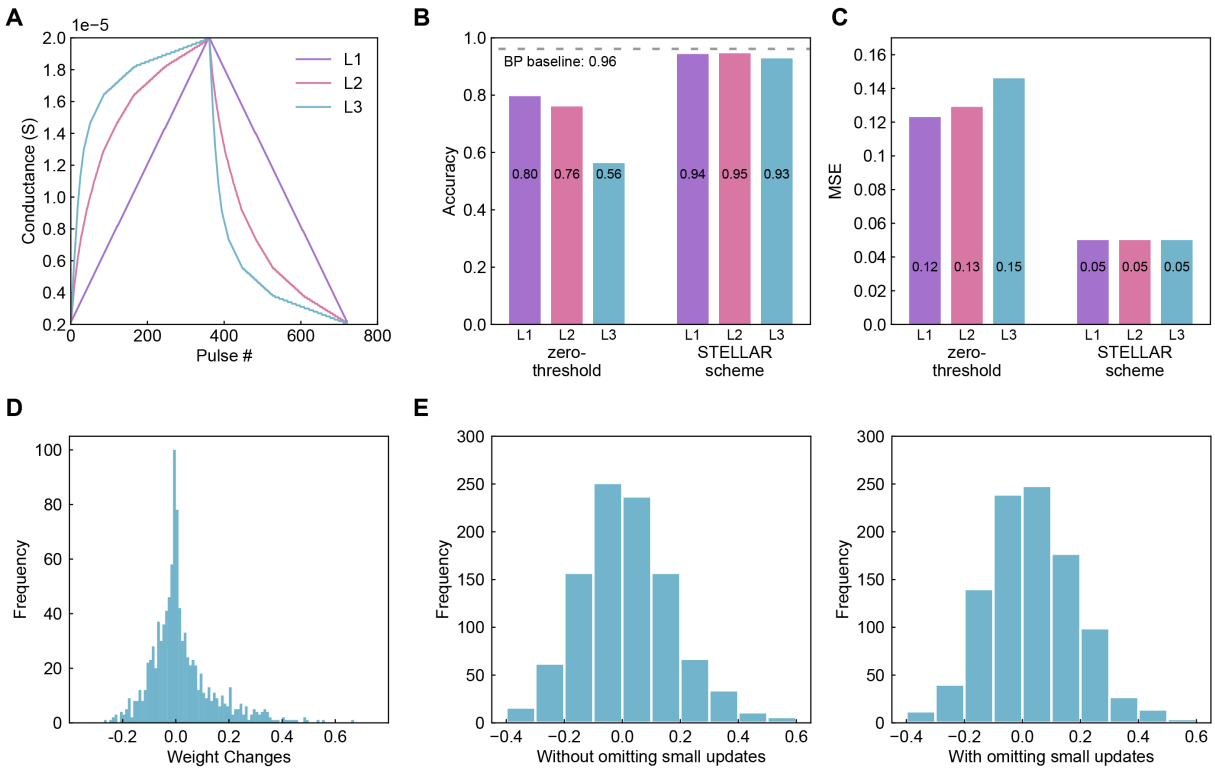

Fig. S2. Performance of the STELLAR update scheme under different device asymmetric switching behaviors. (**A**) Three nonlinear device switching models with different updating asymmetry factors. (**B**) The comparison among inference accuracies under three asymmetric switching models for zero-threshold configuration and the STELLAR scheme. (**C**) Comparison of gradient MSE between zero-threshold configuration and the STELLAR scheme based on the ideal device model (L1) and asymmetric device models (L2 & L3). (**D**) The distribution of weight changes. (**E**) Comparison of the final weight distributions after 100 iterations under the conventional BP scheme with and without omitting small updates.

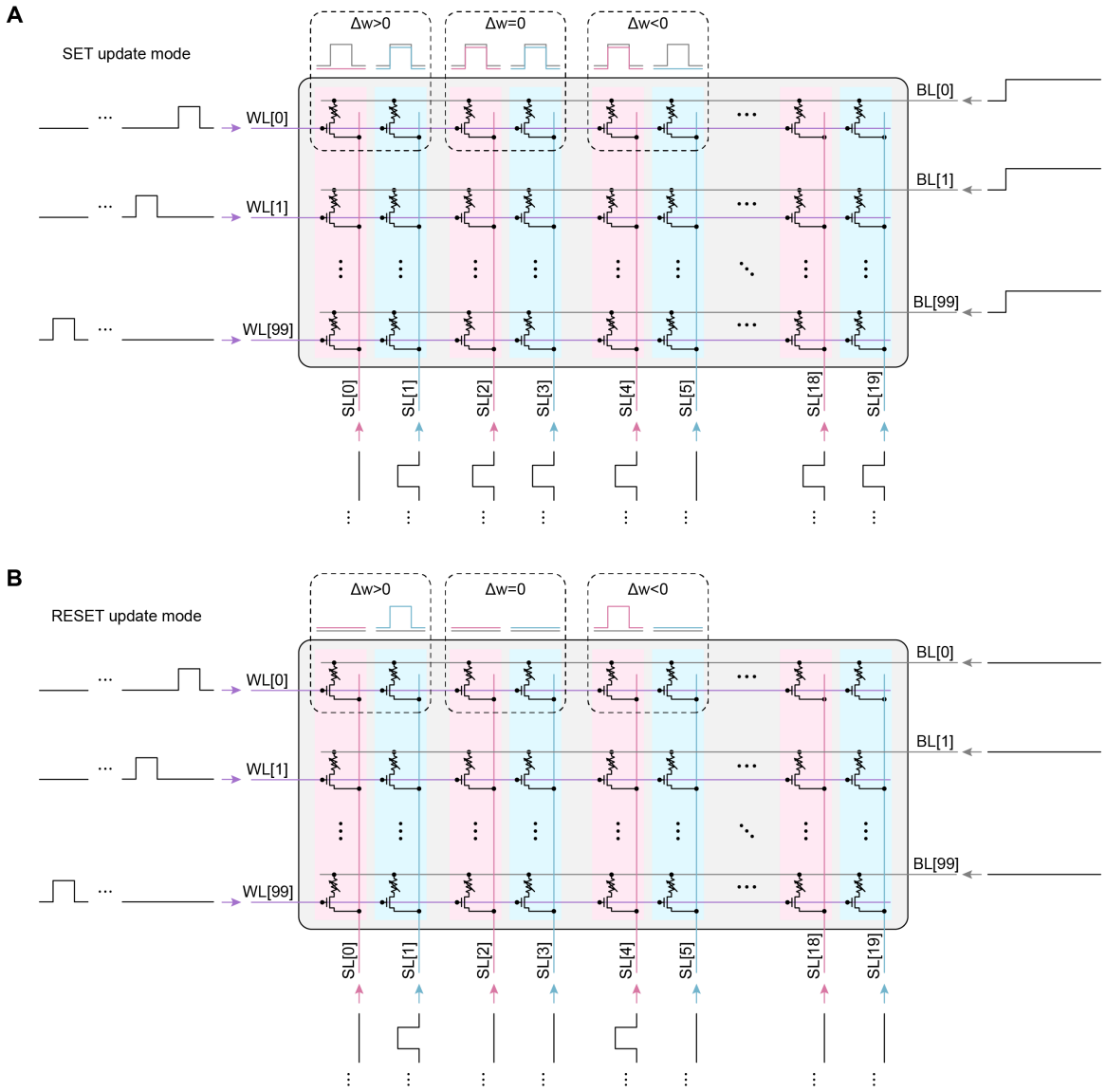

Fig. S3. Schematics of array operations during update modes. Schematics of array operations in SET update modes (**A**) and RESET update modes (**B**). The WL signals select which row to operate in turns, and the SL signals determine which memristors to operate on the selected row.

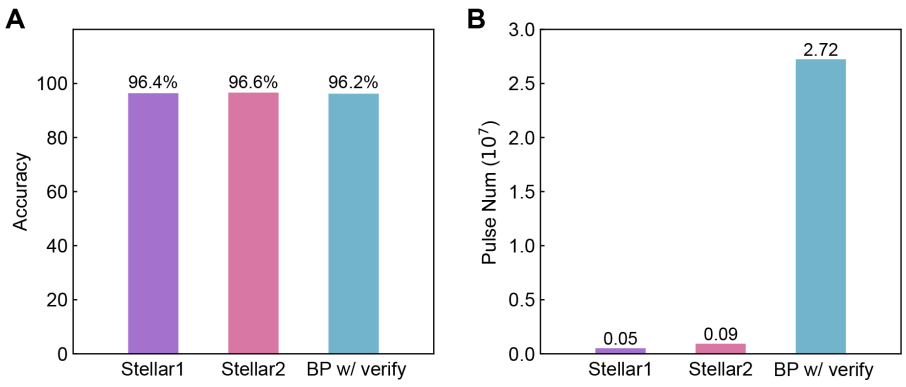

Fig. S4. Endurance requirement for different training schemes during the on-chip learning process. (**A**) Comparison of inference accuracies after training. (**B**) Comparison of required programming pulse numbers during training.

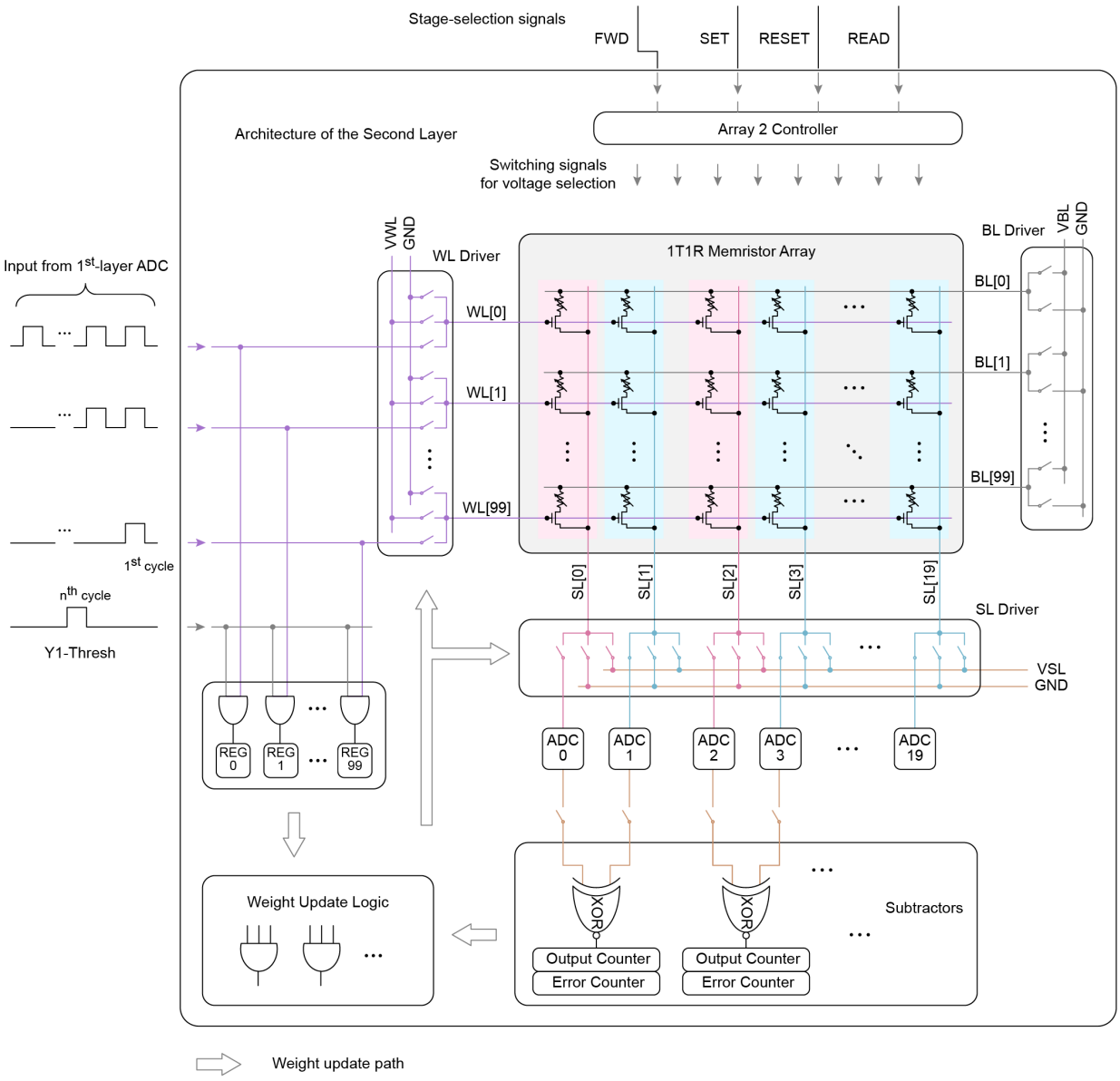

Fig. S5. Architecture of the 2nd layer of the memristor chip.

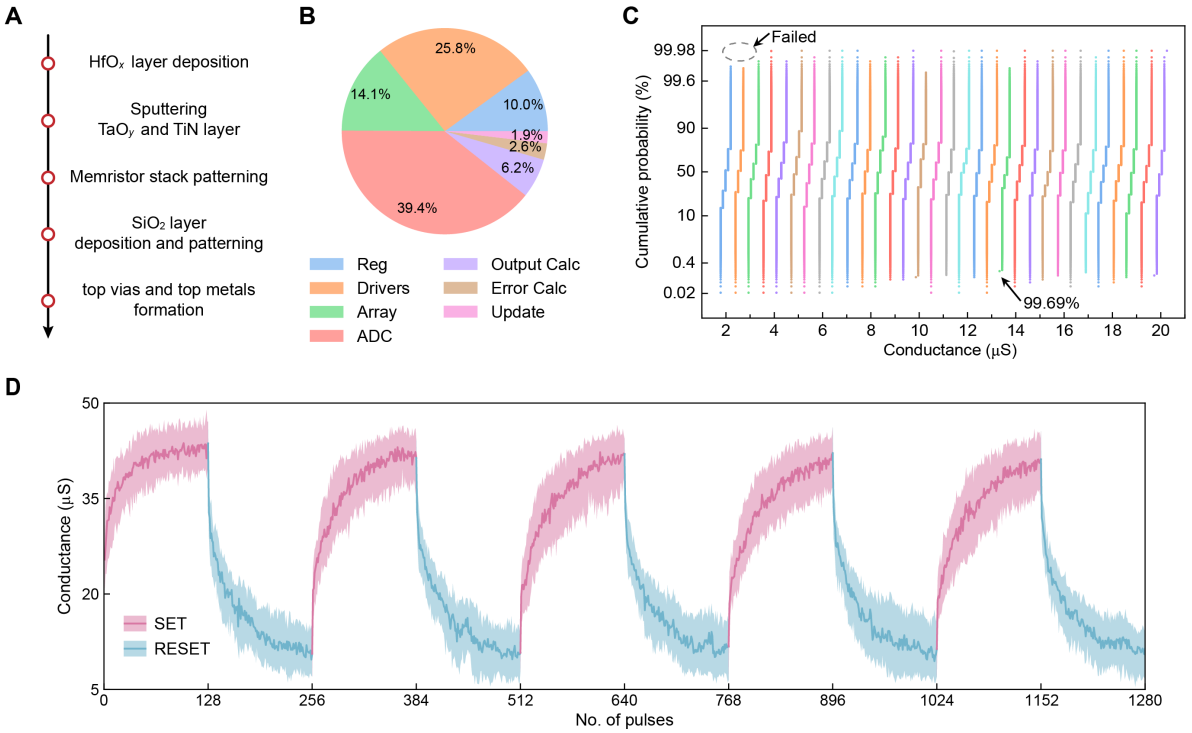

Fig. S6. Fabrication and characterization of the memristor devices. (**A**) Process flow of the fabrication of the memristor devices. (**B**) Chip area breakdown. (**C**) Cumulative probability distribution of ~160K memristors with respect to 32 equally distributed conductance states. The memristors outside of the target range were not shown, as marked by the grey circle. The minimum success rate among the 32 conductance states was 99.69%. (**D**) Analog switching behaviors of 64 memristors under identical pulse trains. The dark lines represent the median values of the conductance, and the light colors fill the regions between the lower and upper quartiles.

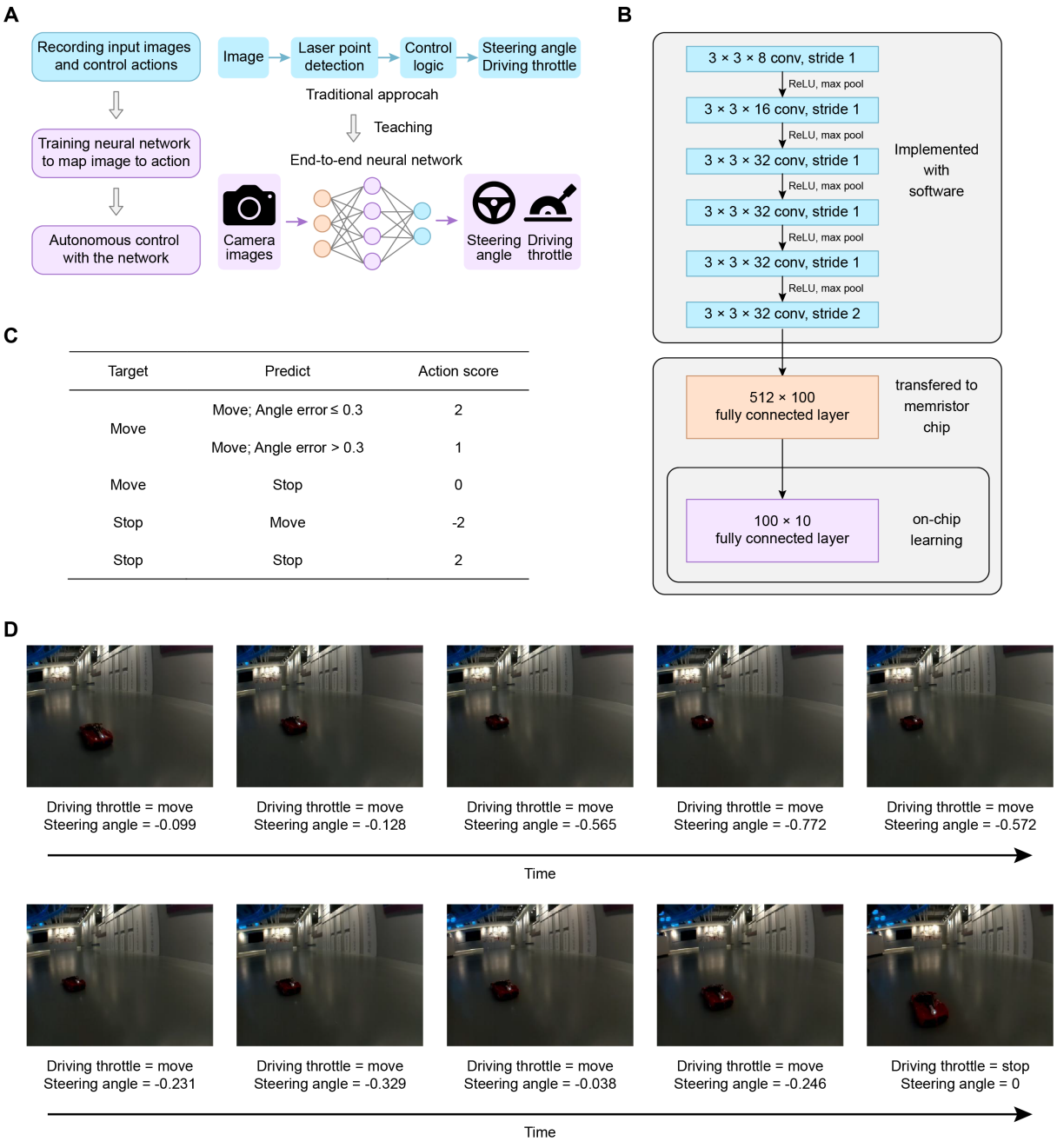

Fig. S7. Motion control task of learning new samples. (**A**) Diagram of the realization of end-to-end autonomous control. (**B**) The diagram of the CNN model and computing locations for the motion control task. (**C**) Table of the scoring rule for different control decisions under the consideration of both the steering angle and driving throttle. Taking the first row as an example, for a certain network prediction, if both the predicted throttle and corresponding labeled throttle indicated 'move', and the difference between the predicted and labeled angles did not exceed 0.3, the score assigned to this predicted control decision would be 2. (**D**) A sequence of input camera

images and their corresponding control decisions produced during real-time driving by the network deployed on the memristor chip.

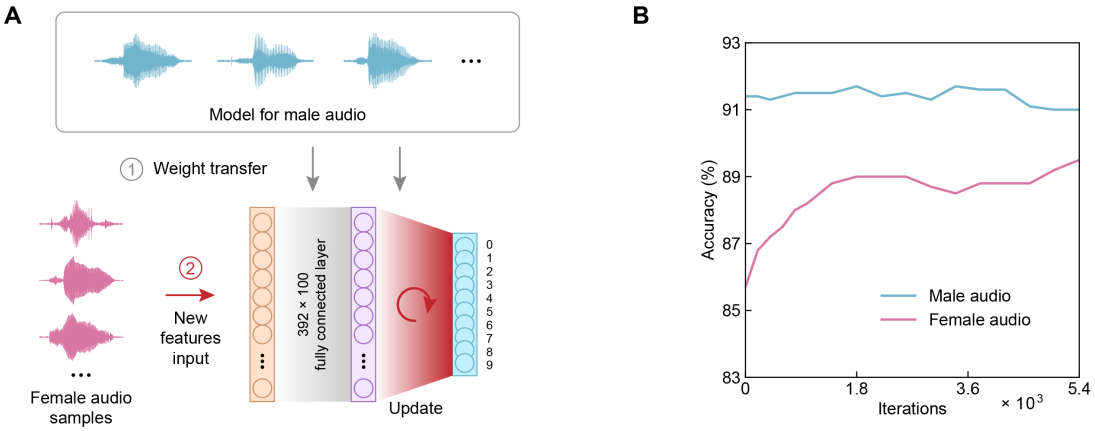

Fig. S8. Audio recognition task of learning new samples. (**A**) Illustration of the improvement learning of new samples in the audio recognition task. (**B**) Changes in the recognition accuracies achieved for male and female audio over the course of the on-chip improvement learning for female audio samples.

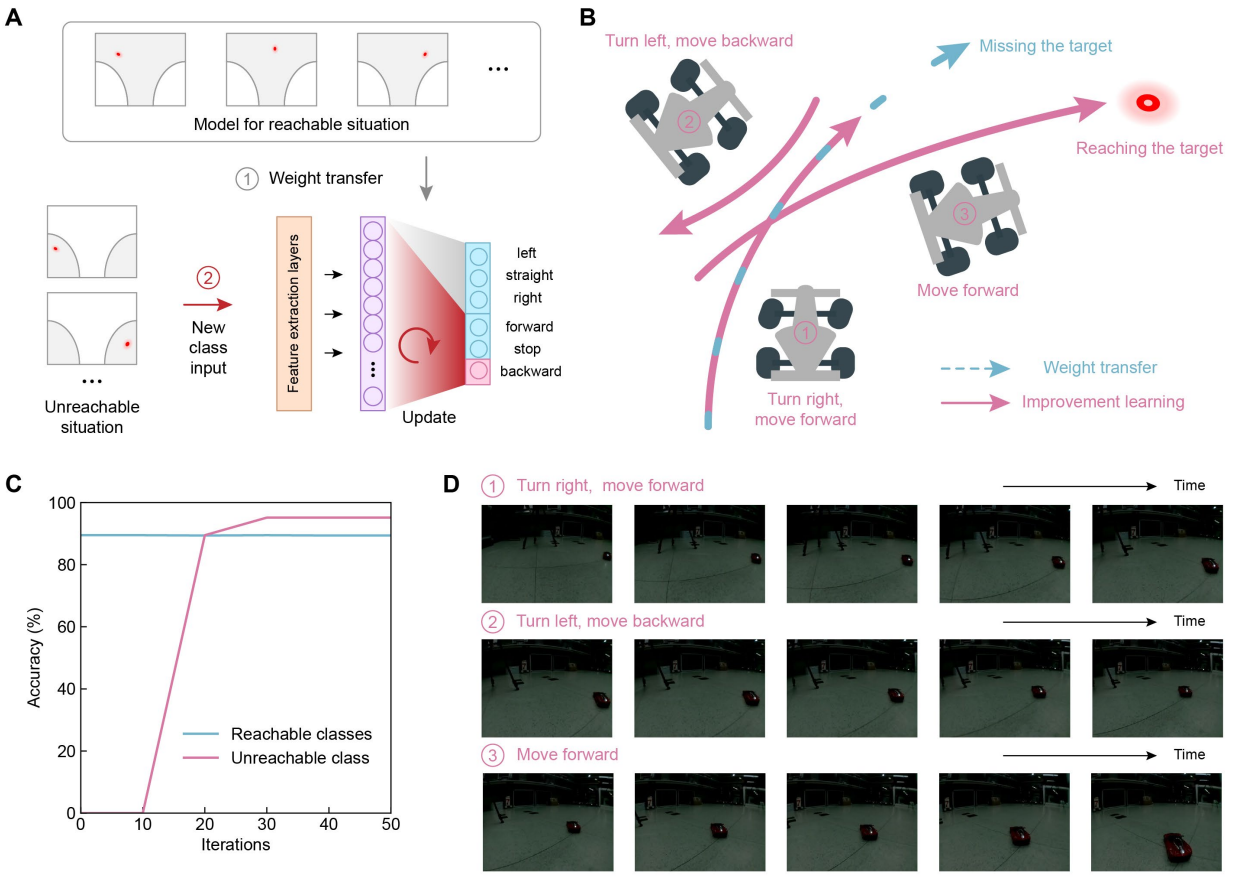

Fig. S9. Motion control task of learning a new class. (**A**) Illustration of the improvement learning of the new class in the light-chasing car task. (**B**) Illustration of the car's movements and trajectories before and after improvement learning of the new class. Before improvement learning, the car moved forward even though it was going to miss its target. After improvement learning, the car moved backward to align itself with the target and then moved forward to reach the target. (**C**) Changes in the throttle classification accuracies for the old classes and the new class over the course of the on-chip improvement learning for the new class. (**D**) A sequence of camera images corresponding to the trajectory produced after improvement learning shown in (B). The car initially moved forward until it was about to miss the target, then moved backward to align itself with the target, and finally moved forward again to reach the target.

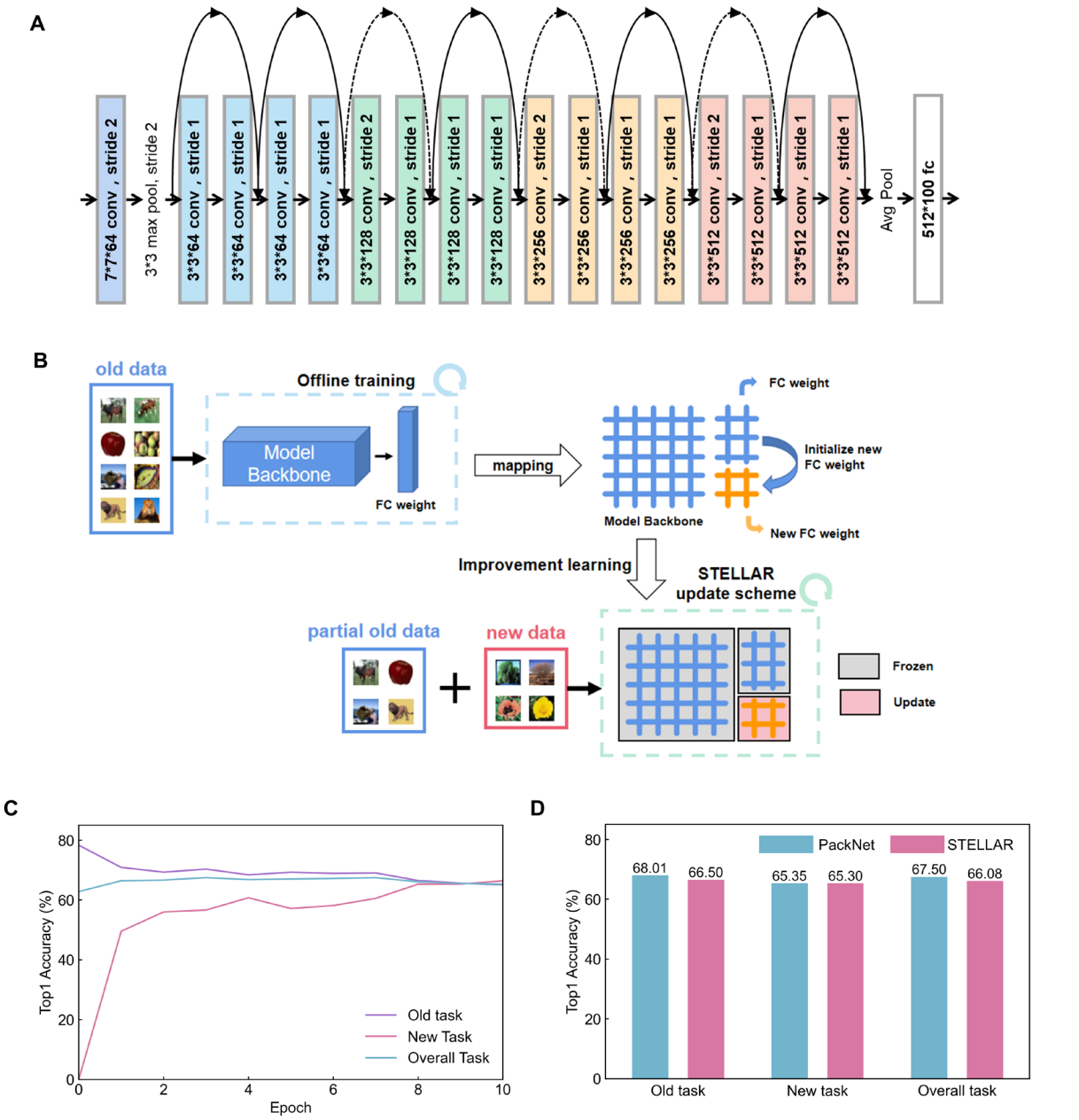

Fig. S10. Scalability of the STELLAR scheme. (**A**) The network diagram of the adopted ResNet20 model. (**B**) The complete procedure for how to conduct the improvement learning simulation in the STELLAR architecture for the ResNet20 model. (**C**) The evolution of recognition accuracies on the old, new, and total classes. (**D**) Improvement learning accuracy comparison between the STELLAR-architecture-based memristor ResNet20 and floating-precision PackNet.

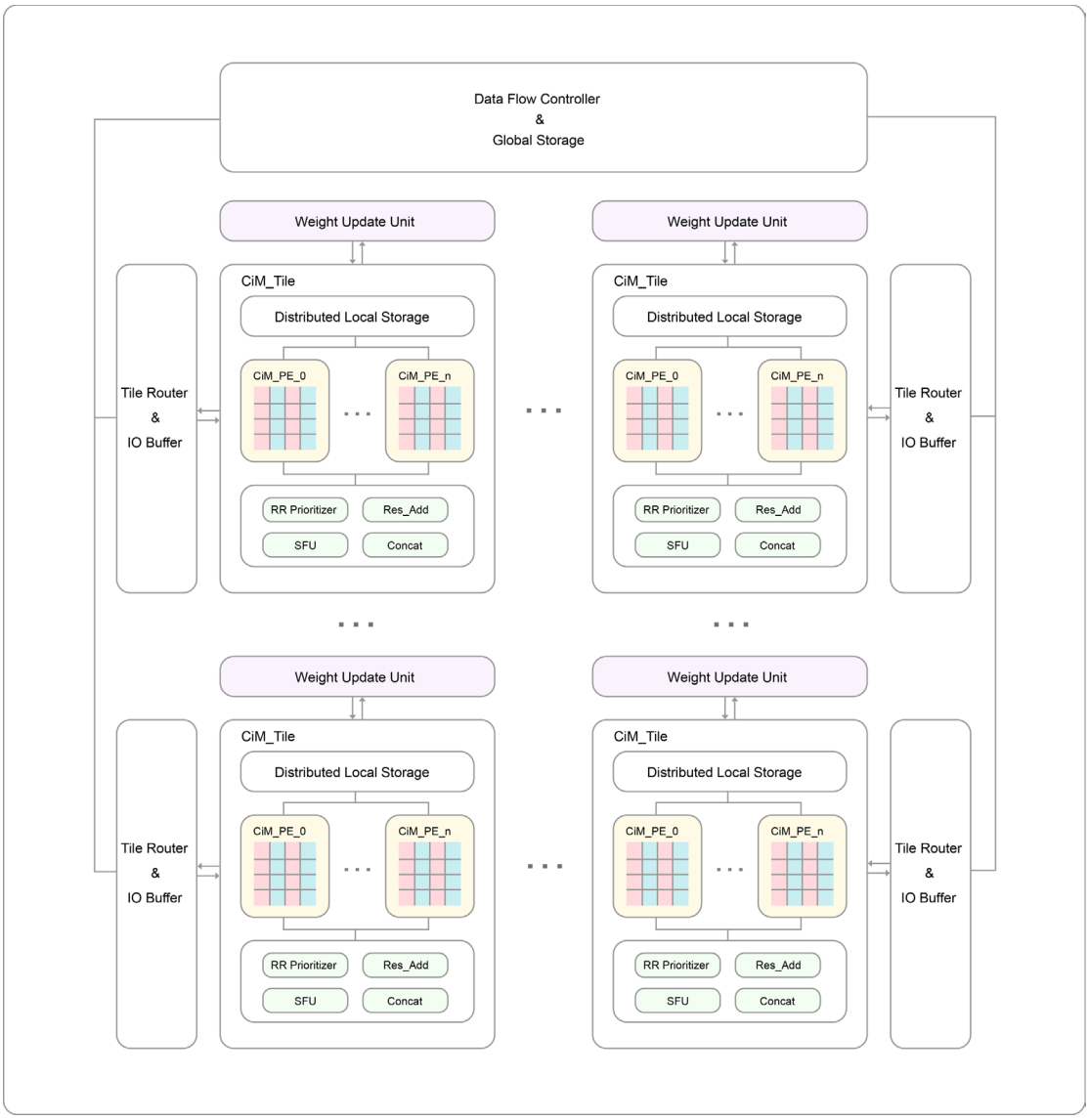

Fig. S11. The architecture of the tiled memristor chip with the STELLAR scheme for efficient improvement learning.

Table S1. Detailed metrics produced during each stage of the learning mode.

	Delay (μs)	Power (mW)	Energy (nJ)
FWD stage	14.85	54.64	811.3
SET stage	85.95	2.49	213.7
RESET stage	55.95	3.01	168.1

Table S2. Parameters for the performance evaluation of conventional hardware.

Parameters of the HNPU-based system	Value
Computing power	3.07 TOPS
Energy efficiency	2.64 TOPS/W
Memory bandwidth	19.2 GB/s
Energy consumption of memory operation	55 pJ/bit

Table S3. Key settings of the simulated chip model.

Key configurations	Value
Clock frequency	200 MHz
Data bus width	64 bit
SRAM buffer size	628.5 KB
PE shape	1,152×512
# ADC per PE	64
# Row buffer per PE	1,152
DAC precision	4 bit
ADC precision	8 bit
Latency per VMM calculation	30 ns
Latency per parallel update	50 ns
ADC energy of VMM calculation per PE	0.97 nJ
Row buffers and DAC energy of VMM calculation per PE	2.54 nJ
Memristor array energy of VMM calculation per PE	0.20 nJ

Captions for Movie S1.

The movie shows the improvement learning of new samples in a light-chasing car task. Before the improvement learning, the car was adept at pursuing the target in dark scenes; however, it occasionally missed the target in bright ones, even deviating from the target or proceeding forward in the absence of target. After the improvement learning of bright scene samples, the car demonstrated adeptness in bright scenes and preserved its performance in the dark scenes.

Captions for Movie S2.

The movie shows the improvement learning of a new class in a light-chasing car task. Before improvement learning, the car understood only when to move forward or stop. In unreachable situations where the target was located at the corner of the visual field, the car would not reach the target even though it turned to the extreme right (or left) and moved forward until it stopped. After the improvement learning of the unreachable situation images labeled with “moving backward”, the car mastered moving backward to align itself with the target and ultimately succeeded in reaching it.

Captions for Movie S3.

The movie shows the process of handwritten digit recognition using a memristor chip. We conducted real-time handwritten digit recognition with the memristor chip.

References and Notes

1. W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge Computing: Vision and Challenges. *IEEE Internet Things J.* **3**, 637–646 (2016). doi:10.1109/JIOT.2016.2579198

2. D. E. Rumelhart, G. Hinton, R. J. Williams, Learning representations by back-propagating errors. *Nature* **323**, 533–536 (1986). doi:10.1038/323533a0

3. Y. LeCun, Y. Bengio, G. Hinton, Deep learning. *Nature* **521**, 436–444 (2015). doi:10.1038/nature14539 Medline

4. A. Coates, B. Huval, T. Wang, D. J. Wu, A. Y. Ng, B. Catanzaro, "Deep learning with COTS HPC systems" in *Proceedings of the 30th International Conference on Machine Learning* (PMLR, 2013), pp. 1337–1345.

5. M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)" in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)* (IEEE, 2014), pp. 10–14.

6. H.-S. P. Wong, S. Salahuddin, Memory leads the way to better computing. *Nat. Nanotechnol.* **10**, 191–194 (2015). doi:10.1038/nnano.2015.29 Medline

7. E. Strubell, A. Ganesh, A. McCallum, "Energy and policy considerations for modern deep learning research" in *Proceedings of the AAAI Conference on Artificial Intelligence* (AAAI, 2020), pp. 13693–13696.

8. D. Ielmini, H.-S. P. Wong, In-memory computing with resistive switching devices. *Nat. Electron.* **1**, 333–343 (2018). doi:10.1038/s41928-018-0092-2

9. M. A. Zidan, J. P. Strachan, W. D. Lu, The future of electronics based on memristive systems. *Nat. Electron.* **1**, 22–29 (2018). doi:10.1038/s41928-017-0006-8

10. M. Lanza, A. Sebastian, W. D. Lu, M. Le Gallo, M.-F. Chang, D. Akinwande, F. M. Puglisi, H. N. Alshareef, M. Liu, J. B. Roldan, Memristive technologies for data storage, computation, encryption, and radio-frequency communication. *Science* **376**, eabj9979 (2022). doi:10.1126/science.abj9979 Medline

11. Q. Liu, B. Gao, P. Yao, D. Wu, J. Chen, Y. Pang, W. Zhang, Y. Liao, C.-X. Xue, W.-H. Chen, J. Tang, Y. Wang, M.-F. Chang, H. Qian, H. Wu, "33.2 A Fully Integrated Analog ReRAM Based 78.4TOPS/W Compute-In-Memory Chip with Fully Parallel MAC Computing" in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)* (IEEE, 2020), pp. 500–502.

12. W. Wan, R. Kubendran, S. B. Eryilmaz, W. Zhang, Y. Liao, D. Wu, S. Deiss, B. Gao, P. Raina, S. Joshi, H. Wu, G. Cauwenberghs, H.-S. P. Wong, "33.1 A 74 TMACS/W CMOS-RRAM Neurosynaptic Core with Dynamically Reconfigurable Dataflow and In-situ Transposable Weights for Probabilistic Graphical Models" in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)* (IEEE, 2020), pp. 498–500.

13. C.-X. Xue, Y.-C. Chiu, T.-W. Liu, T.-Y. Huang, J.-S. Liu, T.-W. Chang, H.-Y. Kao, J.-H. Wang, S.-Y. Wei, C.-Y. Lee, S.-P. Huang, J.-M. Hung, S.-H. Teng, W.-C. Wei, Y.-R. Chen, T.-H. Hsu, Y.-K. Chen, Y.-C. Lo, T.-H. Wen, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, M.-S. Ho, C.-Y. Su, C.-C. Chou, Y.-D. Chih, M.-F. Chang, A CMOS-integrated

compute-in-memory macro based on resistive random-access memory for AI edge devices. *Nat. Electron.* **4**, 81–90 (2021). doi:10.1038/s41928-020-00505-5

14. W.-H. Chen, C. Dou, K.-X. Li, W.-Y. Lin, P.-Y. Li, J.-H. Huang, J.-H. Wang, W.-C. Wei, C.-X. Xue, Y.-C. Chiu, Y.-C. King, C.-J. Lin, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, J. J. Yang, M.-S. Ho, M.-F. Chang, CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors. *Nat. Electron.* **2**, 420–428 (2019). doi:10.1038/s41928-019-0288-0

15. S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. di Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. P. Farinha, B. Killeen, C. Cheng, Y. Jaoudi, G. W. Burr, Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature* **558**, 60–67 (2018). doi:10.1038/s41586-018-0180-5 Medline

16. A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, E. Eleftheriou, Memory devices and applications for in-memory computing. *Nat. Nanotechnol.* **15**, 529–544 (2020). doi:10.1038/s41565-020-0655-z Medline

17. W. Zhang, B. Gao, J. Tang, P. Yao, S. Yu, M.-F. Chang, H.-J. Yoo, H. Qian, H. Wu, Neuro-inspired computing chips. *Nat. Electron.* **3**, 371–382 (2020). doi:10.1038/s41928-020-0435-7

18. M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, D. B. Strukov, Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* **521**, 61–64 (2015). doi:10.1038/nature14441 Medline

19. P. Yao, H. Wu, B. Gao, S. B. Eryilmaz, X. Huang, W. Zhang, Q. Zhang, N. Deng, L. Shi, H. P. Wong, H. Qian, Face classification using electronic synapses. *Nat. Commun.* **8**, 15199 (2017). doi:10.1038/ncomms15199 Medline

20. C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang, W. Song, J. P. Strachan, M. Barnell, Q. Wu, R. S. Williams, J. J. Yang, Q. Xia, Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nat. Commun.* **9**, 2385 (2018). doi:10.1038/s41467-018-04484-2 Medline

21. Z. Wang, C. Li, W. Song, M. Rao, D. Belkin, Y. Li, P. Yan, H. Jiang, P. Lin, M. Hu, J. P. Strachan, N. Ge, M. Barnell, Q. Wu, A. G. Barto, Q. Qiu, R. S. Williams, Q. Xia, J. J. Yang, Reinforcement learning with analogue memristor arrays. *Nat. Electron.* **2**, 115–124 (2019). doi:10.1038/s41928-019-0221-6

22. C. Li, Z. Wang, M. Rao, D. Belkin, W. Song, H. Jiang, P. Yan, Y. Li, P. Lin, M. Hu, N. Ge, J. P. Strachan, M. Barnell, Q. Wu, R. S. Williams, J. J. Yang, Q. Xia, Long short-term memory networks in memristor crossbar arrays. *Nat. Mach. Intell.* **1**, 49–57 (2019). doi:10.1038/s42256-018-0001-4

23. P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, H. Qian, Fully hardware-implemented memristor convolutional neural network. *Nature* **577**, 641–646 (2020). doi:10.1038/s41586-020-1942-4 Medline

24. F. Cai, J. M. Correll, S. H. Lee, Y. Lim, V. Bothra, Z. Zhang, M. P. Flynn, W. D. Lu, A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations. *Nat. Electron.* **2**, 290–299 (2019). doi:10.1038/s41928-019-0270-x

25. E. J. Fuller, S. T. Keene, A. Melianas, Z. Wang, S. Agarwal, Y. Li, Y. Tuchman, C. D. James, M. J. Marinella, J. J. Yang, A. Salleo, A. A. Talin, Parallel programming of an ionic floating-gate memory array for scalable neuromorphic computing. *Science* **364**, 570–574 (2019). doi:10.1126/science.aaw5581 Medline

26. G. W. Burr, R. M. Shelby, S. Sidler, C. Di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, B. N. Kurdi, H. Hwang, 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). doi:10.1109/TED.2015.2439635

27. H. Wu, P. Yao, B. Gao, W. Wu, Q. Zhang, W. Zhang, N. Deng, D. Wu, H.-S. P. Wong, S. Yu, H. Qian, "Device and circuit optimization of RRAM for neuromorphic computing" in *2017 IEEE International Electron Devices Meeting (IEDM)* (IEEE, 2017), pp. 11.5.1–11.5.4.

28. P. Y. Chen, X. Peng, S. Yu, NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning. *IEEE Trans. Comput. Aided Des. Integrated Circ. Syst.* **37**, 3067–3080 (2018). doi:10.1109/TCAD.2018.2789723

29. Y. Xi, B. Gao, J. Tang, A. Chen, M.-F. Chang, X. S. Hu, J. Van Der Spiegel, H. Qian, H. Wu, In-memory Learning with Analog Resistive Switching Memory: A Review and Perspective. *Proc. IEEE* **109**, 14–42 (2021). doi:10.1109/JPROC.2020.3004543

30. Q. Zhang, H. Wu, P. Yao, W. Zhang, B. Gao, N. Deng, H. Qian, Sign backpropagation: An on-chip learning algorithm for analog RRAM neuromorphic computing systems. *Neural Netw.* **108**, 217–223 (2018). doi:10.1016/j.neunet.2018.08.012 Medline

31. A. Krizhevsky, G. Hinton, "Learning multiple layers of features from tiny images"; (Univ. of Toronto, 2009) https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

32. W. Wu, H. Wu, B. Gao, P. Yao, X. Zhang, X. Peng, S. Yu, H. Qian, "A Methodology to Improve Linearity of Analog RRAM for Neuromorphic Computing" in *2018 IEEE Symposium on VLSI Technology* (IEEE, 2018), pp. 103–104.

33. Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition. *Proc. IEEE* **86**, 2278–2324 (1998). doi:10.1109/5.726791

34. P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, Y. Xie, "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory" in *Proceedings of the 43rd International Symposium on Computer Architecture* (ACM/IEEE, 2016), pp. 27-39.

35. L. Song, X. Qian, H. Li, Y. Chen, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning" in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (IEEE, 2017), pp. 541–552.

36. D. Han, D. Im, G. Park, Y. Kim, S. Song, J. Lee, H.-J. Yoo, HNPU: An Adaptive DNN Training Processor Utilizing Stochastic Dynamic Fixed-Point and Active Bit-Precision Searching. *IEEE J. Solid-State Circuits* **56**, 2858–2869 (2021). doi:10.1109/JSSC.2021.3066400

37. R. Khaddam-Aljameh, M. Stanisavljevic, J. Fornt Mas, G. Karunaratne, M. Braendli, F. Liu, A. Singh, S. M. Muller, U. Egger, A. Petropoulos, T. Antonakopoulos, K. Brew, S. Choi,

I. Ok, F. L. Lie, N. Saulnier, V. Chan, I. Ahsan, V. Narayanan, S. R. Nandakumar, M. Le Gallo, P. A. Francese, A. Sebastian, E. Eleftheriou, "HERMES Core – A 14nm CMOS and PCM-based In-Memory Compute Core using an array of 300ps/LSB Linearized CCO-based ADCs and local digital processing" in *2021 Symposium on VLSI Circuits* (IEEE, 2021), pp. 1–2.

38. W. Wan, R. Kubendran, B. Gao, S. Joshi, P. Raina, H. Wu, G. Cauwenberghs, H.-S. P. Wong, "A Voltage-Mode Sensing Scheme with Differential-Row Weight Mapping for Energy-Efficient RRAM-Based In-Memory Computing" in *2020 IEEE Symposium on VLSI Technology* (IEEE, 2020), pp. 1–2.

39. J.-M. Hung, C.-X. Xue, H.-Y. Kao, Y.-H. Huang, F.-C. Chang, S.-P. Huang, T.-W. Liu, C.-J. Jhang, C.-I. Su, W.-S. Khwa, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, M.-S. Ho, C.-C. Chou, Y.-D. Chih, T.-Y. Chang, M.-F. Chang, A four-megabit compute-in-memory macro with eight-bit precision based on CMOS and resistive random-access memory for AI edge devices. *Nat. Electron.* **4**, 921–930 (2021). doi:10.1038/s41928-021-00676-9

40. K. Weiss, T. M. Khoshgoftaar, D. Wang, A survey of transfer learning. *J. Big Data* **3**, 9 (2016). doi:10.1186/s40537-016-0043-6

41. W.-H. Huang, T.-H. Wen, J.-M. Hung, W.-S. Khwa, Y.-C. Lo, C.-J. Jhang, H.-H. Hsu, Y.-H. Chin, Y.-C. Chen, C.-C. Lo, R.-S. Liu, K.-T. Tang, C.-C. Hsieh, Y.-D. Chih, T.-Y. Chang, M.-F. Chang, "A Nonvolatile AI-Edge Processor with 4MB SLC-MLC Hybrid-Mode ReRAM Compute-in-Memory Macro and 51.4-251TOPS/W" in *2023 IEEE International Solid- State Circuits Conference (ISSCC)* (IEEE, 2023), pp. 15–17.

42. W. Zhang, P. Yao, B. Gao, H. Wu, Data for Edge Learning Using a Fully Integrated Neuro-Inspired Memristor Chip, Zenodo, (2023); https://doi.org/10.5281/zenodo.8145521.

43. W. Zhang, P. Yao, B. Gao, H. Wu, Code for Edge Learning Using a Fully Integrated Neuro-Inspired Memristor Chip, Zenodo (2023); https://doi.org/10.5281/zenodo.8151757.

44. N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, D. H. Yoon, "In-Datacenter Performance Analysis of a Tensor Processing Unit" in *Proceedings of the 44th Annual International Symposium on Computer Architecture* (ACM/IEEE, 2017), pp. 1–12.

45. T. Oh, N. Maghari, U.-K. Moon, "A 5MHz BW 70.7dB SNDR noise-shaped two-step quantizer based ΔΣ ADC" in *2012 Symposium on VLSI Circuits (VLSIC)* (IEEE, 2012), pp. 162–163.

46. V. Joshi, M. Le Gallo, S. Haefeli, I. Boybat, S. R. Nandakumar, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian, E. Eleftheriou, Accurate deep neural network inference using computational phase-change memory. *Nat. Commun.* **11**, 2473 (2020). doi:10.1038/s41467-020-16108-9 Medline

47. Z. Wang, H. Huang, J. Zhang, G. Alonso, “Shuhai: Benchmarking High Bandwidth Memory On FPGAs” in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)* (IEEE, 2020), pp. 111–119.

48. M. M. Sabry Aly, T. F. Wu, A. Bartolo, Y. H. Malviya, W. Hwang, G. Hills, I. Markov, M. Wootters, M. M. Shulaker, H.-S. Philip Wong, S. Mitra, The N3XT Approach to Energy-Efficient Abundant-Data Computing. *Proc. IEEE* **107**, 19–48 (2019). doi:10.1109/JPROC.2018.2882603

49. A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, D. Nellans, “MCM-GPU: Multi-Chip-Module GPUs for Continued Performance Scalability” in *Proceedings of the 44th Annual International Symposium on Computer Architecture* (ACM/IEEE, 2017), pp. 320–332.

50. M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, K. Zieba, End to End Learning for Self-Driving Cars. arXiv:1604.07316 [cs.CV] (2016).

51. S. Becker, M. Ackermann, S. Lapuschkin, K.-R. Müller, W. Samek, Interpreting and Explaining Deep Neural Networks for Classification of Audio Signals. arXiv:1807.03418 [cs.SD] (2018).

52. A. Mallya, S. Lazebnik, “PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE/CVF, 2018), pp. 7765–7773.

53. S. C. Y. Hung, C.-H. Tu, C.-E. Wu, C.-H. Chen, Y.-M. Chan, C.-S. Chen, Compacting, Picking and Growing for Unforgetting Continual Learning. *Adv. Neural Inf. Process. Syst.* **32**, 13669–13679 (2019).