nature communications

Article

https://doi.org/10.1038/s41467-025-66240-7

Error-aware probabilistic training for memristive neural networks

Received: 7 May 2025

Accepted: 3 November 2025

Published online: 12 December 2025

Check for updates

Jinchang Liu[1,3], Jian Lu[1,3], Shuangzhu Tang[1], Ruixi Zhou[1], Huiqin Ma[1], Bo Lyu[1], Yang Tian[1], Tuo Shi[1] & Qi Liu[2]

Analog computing-in-memory devices leverage fundamental physical laws for computation, greatly enhancing energy efficiency. However, the stochastic characteristics of analog devices conflict with the deterministic weight update of the backpropagation algorithm (BP), limiting training performance. To overcome the algorithm-device mismatch, we propose an error-aware probabilistic update method (EaPU) that updates the weights based on a specified probability derived from device writing noise. Compared to BP, EaPU reduces the number of weight updates to <1‰ with minimal performance loss. Furthermore, we validate EaPU experimentally on a 180 nm memristor system for image denoising and super-resolution and simulate its performance on ResNet and Vision Transformers. Results confirm that EaPU training yields over 60% accuracy improvement, with ~50.54× and 13.23× lower training energy (and ~35.51× and 11.26× lower inference energy) compared to BP-based memristor training and MADEM, respectively. Moreover, EaPU-based memristor hardware reduces training energy by nearly 6 orders of magnitude compared to graphics processing units. Here, we present a promising approach to precisely and efficiently train analog device-based deep neural networks.

Deep neural networks (DNNs) have brought great success in extensive industrial applications, such as image classification[1,2], object detection[3,4], and natural language processing[5,6]. Large-scale DNNs impose a substantial computing workload, requiring advanced hardware platforms to accelerate the computational tasks[7]. Analog computing-in-memory (ACIM) architecture, which performs vector-matrix multiplication (VMM) based on physical laws, has been regarded as a promising solution to address the limitations of the von Neumann architecture and has achieved significant energy efficiency[8]. Typically, memristive ACIM leverages the tunable non-volatile conductance of memristors as the weight and performs VMMs with Ohm's law and Kirchhoff's law on crossbar arrays, showing significant efficiency improvement[9,10].

Owing to the nonideal characteristics inherent in memristive devices, arrays, and peripheral circuits[11], there are inevitably precision and efficiency limitations in memristor-based DNNs (shown in Fig. 1a), causing degradation in network performance. When memristors are utilized for calculation, the first step is device programming. It loads network weights from software onto hardware. In this process, writing noise (ε_{cell}) is inevitably introduced. ε_{cell} is composed of programming noise and device relaxation. In ACIM hardware, limited resolutions in devices and external circuits always result in discrete conductance states[9,12], thus a tolerance range is typically set to efficiently program the device conductance to a desired continuous state[13,14], leading to programming noise. The relaxation phenomenon after programming significantly contributes to retention failure and increases the error ε_{cell}[9,15,16]. Therefore, ε_{cell} always exists, causing a mismatch between the actual weight and the required weight. Afterward, in the implementation of VMM on memristor arrays, there exists reading noise, IR drop, and external circuit nonidealities. These nonidealities in the VMM calculation result in a residual error ($\varepsilon_{residual}$). Therefore, ε_{cell} and $\varepsilon_{residual}$ lead to precision limitations (quantitative indices for ε_{cell} and $\varepsilon_{residual}$ are introduced in "Methods"). Moreover, in device programming, spatiotemporal variations ε_{var}[17,18] bring about the challenge of

[1]Zhejiang Laboratory, Hangzhou, China. [2]The Frontier Institute of Chip and System, Fudan University, Shanghai, China. [3]These authors contributed equally: Jinchang Liu, Jian Lu. e-mail: shituo@zhejianglab.com; qi_liu@fudan.edu.cn

https://doi.org/10.1038/s41467-025-66240-7

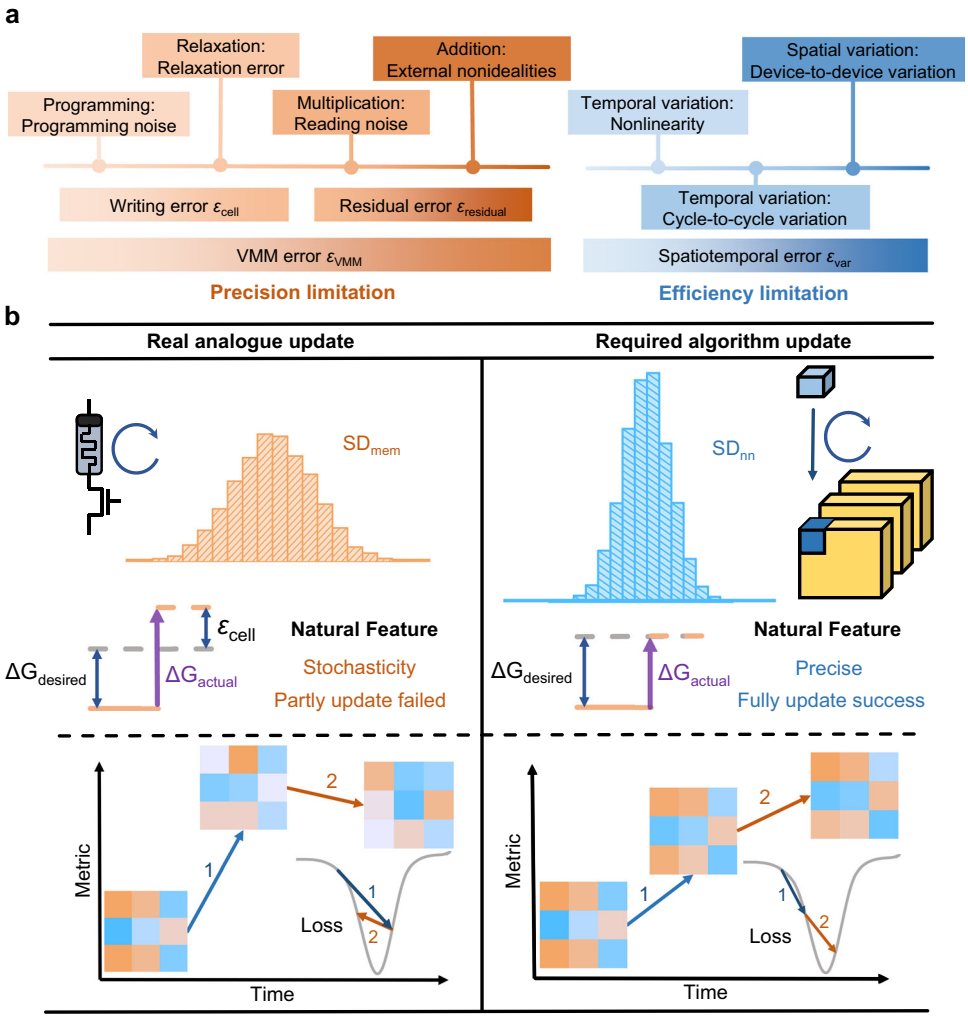

Fig. 1 | Challenge of analog training. a Limitations of memristor-based calculations. Precision limitations of VMM cause the difference between the desired output and actual output, which then worsens the results of ACIM. Efficiency limitations cause more operations, resulting in more time and energy costs during the update process. **b** Distinction between the analog and algorithm-based training processes. The actual update magnitudes ($\Delta G_{desired} + \varepsilon_{cell}$) in real analog devices are much different from the required update magnitudes ($\Delta G_{desired}$) due to the writing error ε_{cell}, making the update process more stochastic and the histogram of actual update magnitudes much flatter (Actually, $|\varepsilon_{cell}| \gg |\Delta G_{desired}|$ and $SD_{mem} \gg SD_{nn}$, analyzed in "Results"). ε_{cell} causes different actual update magnitudes in analog training, shown as the variation of the weight matrix, which leads to deviations from the desired update process and unstable training.

obtaining common programming parameters, reducing the weight update efficiency.

Hardware-aware training[11,19], which involves precise modeling and employs software for error-aware learning and then applies trained parameters to memristor systems, represents an approach to addressing nonidealities. However, training on graphics processing units (GPUs) incurs high training power consumption and latency compared to memristor systems[20], along with substantial modeling costs. Additionally, the separation of training and inference hardware may lead to issues such as modeling biases and the impact of unmodeled parameters. Memristor-based training[18,21], a memristor-based training pathway for error-aware learning, can effectively leverage the high energy efficiency and low latency characteristics of memristor systems to reduce energy consumption and improve training efficiency. Meanwhile, since training is performed directly on the memristor system, it can counteract nonidealities through the learning process and remain unaffected by modeling biases.

It has been confirmed that memristor-based in situ training allows the neural network to tolerate nonideal characteristics of the device[20–23], resulting in a high-performance model. There exist several training strategies, including global learning and local learning. The global training strategy, such as the backpropagation algorithm (BP), has made great progress[18,21]. However, the training strategy based on BP meets the mismatch between analytically calculated training information and the imprecisely programmed device conductance of analog devices[20]. Owing to the programming noise and relaxation effects in memristors, ε_{cell} introduces significant deviations in the actual conductance update magnitudes ($\Delta G_{desired} + \varepsilon_{cell}$) from the desired update magnitudes ($\Delta G_{desired}$) in memristor arrays, resulting in a stochastic conductance update process and thus an unstable training process, as depicted in Fig. 1b. Even though the writing precision in analog devices is around 1 μS[9,19,24,25], there still exists a mismatch between analog devices and algorithm (detailed analysis in "Results"), reducing the training performance. Moreover, the realization of the above high precision relies on complicated circuit design or intensive software processing techniques, which substantially reduce energy and area efficiency. Recently, local learning techniques have been chosen to deal with the imprecision in memristive neural networks[20]. However, owing to the lack of global information, local learning techniques are inferior in the training of DNNs with large datasets yet[26]. Moreover, local learning techniques typically require additional operations (such as different phases or additional branches)[26,27],

increasing energy and time consumption. Therefore, it is necessary to explore an efficient training technique that overcomes the algorithm-device mismatch while preserving global information.

In this work, to address the algorithm-device discrepancy, we propose an error-aware probabilistic update method (EaPU) that employs probabilistic update magnitudes to align with the stochastic updating process of memristors. With the EaPU, only 0.86‰ of the parameters (for ResNet152) are updated, reducing the number of update operations dramatically. For example, EaPU achieves an average update pulse count N_{up} (see "Methods") of 6×10^{-3} in memristors and over 10^4 times less than that of conventional write-verify methods (average update pulse count of 66) with the same precision. When combined with the non-write-verify methods[18], EaPU further achieves an average update pulse count N_{up} of 8.6×10^{-4}, 10^3 times less than that of the original scheme (average update pulse count of 1). The sparse update characteristic is equivalent to enhancing the device's endurance and extending its service life. Moreover, EaPU is highly compatible with BP, making training with EaPU capture global features and facilitate error-aware training simultaneously. For noiseless models, EaPU achieves negligible performance loss in comparison with the original BP. For noisy analog devices, the EaPU is experimentally validated on memristor hardware in multi-point regression problems, including denoising and super-resolution tasks. Training with EaPU on memristor hardware achieves better experimental results, with >80% improvement in structural similarity index (SSIM) in most cases, compared to related BP-based memristive training approaches (see "Methods"). Furthermore, the feasibility of EaPU in large-scale models, such as ResNet[1,28] with CIFAR10 and CIFAR100[29], SRResNet[30] with ImageNet[31], and Swin Transformer[32] with ImageNet100, is verified via simulation. The simulation results confirm >60% improvements in accuracy when noises are present. Moreover, evaluated at the 180 nm technology node, EaPU demonstrates a training energy cost reduction of 50.54× and 13.23× over classical BP-based memristor training method[18] and advanced MADEM[20], respectively (For inference, the reduction is 35.51× and 11.26×, respectively.). Besides, when scaled down to 16 nm, the memristor hardware with EaPU offers an energy advantage of nearly six orders of magnitude compared with GPUs. At last but not least, we further test the retention characteristics of the network after EaPU training, confirming its long-term stability. Comparisons with training strategies from other distinct approaches demonstrate that EaPU exhibits good and balanced performance in learning efficiency, training stability, training performance, algorithm compatibility, energy-accuracy efficiency, and latency-accuracy efficiency.

Results

Algorithm-device mismatch and EaPU method

To overcome the unstable training processes originating from ε_{cell}, it is necessary to conduct an in-depth analysis of ε_{cell}. The relationship between device conductance and actual weight is typically set to a linear correspondence[8,33,34]. Then the relationship between the conductance update magnitude ΔG and the weight update magnitude ΔW is given by

$$\frac{\triangle G}{G_{max}} = \frac{\triangle W}{W_{max}} \tag{1}$$

where G_{max} is the maximum programmable device conductance range and W_{max} is the maximum absolute synaptic weight value. We define the R_{wg} as the ratio between W_{max} and G_{max}, then

$$\Delta W_{desired} = R_{wg} \times \Delta G_{desired} \tag{2}$$

$$\Delta W_{actual} = R_{wg} \times (\Delta G_{desired} + \varepsilon_{cell}) \tag{3}$$

where Eqs. (2) and (3) show the ideal and actual update magnitudes in memristors, respectively. By analyzing Eqs. (1) and (3), it can be observed that a narrow conductance range (small G_{max}) leads to a small $\Delta G_{desired}$, further exacerbating the negative impact of ε_{cell}. Due to the small learning rate (typically smaller than 1e-4)[35–37], the $|\Delta G_{desired}|$ of DNNs is always smaller than $|\varepsilon_{cell}|$. For example, we gather and analyze the weight update magnitudes $\Delta W_{desired}$ by training ResNet34[1,28] with CIFAR10[29] using a large learning rate of 1e-3 (introduced in Supplementary Fig. 2). It turns out that the weight update magnitudes are extremely small (~10^{-5}). According to Eq. (1), even if G_{max} is in the range of thousands of μS, $|\Delta G_{desired}|$ is always less than 0.1 μS. Recently, researchers have achieved the programming precision of 1 μS[9,19,24,25]. However, $|\varepsilon_{cell}|$ of 1 μS is still much larger than update magnitudes $|\Delta G_{desired}|$(<0.1 μS), causing learning information loss. Some excellent high-precision methods[38,39] may reduce ε_{cell} further, but the reliance on external software processing or the need for precise circuits leads to increased area, energy costs, and reduced flexibility. It would be more advantageous to explore a flexible and low-cost training approach. According to Eq. (3), the large ε_{cell} makes the ΔW_{actual} much larger than $\Delta W_{desired}$, leading to the terrible learning information missing in analog learning (see detailed experiments in the "Simulation results on advanced networks" section). To overcome the contradiction between a large ε_{cell} and a small $\Delta G_{desired}$, expanding $\Delta G_{desired}$ is an efficient choice. Small R_{wg} indicates a wide conductance range, causing large $\Delta G_{desired}$ and more ideal device characteristics. However, a wide conductance range means high conductance, causing high energy costs and complex circuit design. Moreover, most existing memristors are always in a narrow conductance range[13,20,40]. Therefore, we propose a robust and cost-effective roadmap to extend the weight update magnitudes with negligible network performance degradation.

Enlarging the update magnitudes and maintaining the algorithm performance typically requires a probabilistic method to reserve the statistical mean value of updates[41,42]. The variability of ε_{cell} represents the stochasticity of the memristor update. Moreover, the standard deviation of ε_{cell} ($SD_{\varepsilon_{cell}}$) can be thought of as the quantified description of ε_{cell}. We propose the EaPU (shown in Fig. 2a) to adjust the update magnitudes (ΔW, see "Methods") with a threshold ΔW_{th} (typically is $SD_{\varepsilon_{cell}} \times R_{wg}$, analyzed in the "Hyperparameter ΔW_{th}" section), thereby reducing the impact of update noise. The key idea of EaPU is to extend the target update magnitudes with a certain probability. If the absolute value of the original update magnitude $|\Delta W|$ exceeds the threshold ΔW_{th}, the update magnitude ΔW_n remains ΔW. Otherwise, if the $|\Delta W|$ is below the ΔW_{th}, the ΔW_n has a possibility ($|\Delta W|/\Delta W_{th}$) to be set as the $\text{sign}(\Delta W)\cdot\Delta W_{th}$ or it may stay at 0, meaning no update. The parameter update formula of EaPU can be expressed as follows:

$$\triangle W_n = \begin{cases} \triangle W, & |\triangle W| \geq \triangle W_{th} \\ \text{sign}(\triangle W) \cdot \triangle W_{th}, & |\triangle W| < \triangle W_{th} \text{and with probability } p = \frac{|\triangle W|}{\triangle W_{th}} \\ 0, & |\triangle W| < \triangle W_{th} \text{and with probability } 1 - p \end{cases} \tag{4}$$

where ΔW_{th} is the threshold and p is the probability rounded to ΔW_{th}. EaPU offers plug-and-play compatibility with popular optimizers (introduced in "Methods"). For example, we can improve the optimizer Adam[43] to AdamEaPU (summarized in Supplementary Note 1) by transforming original update magnitudes to desired update magnitudes with Formula (4) to adapt the feature of memristors. Experimental results (shown in Supplementary Fig. 3) demonstrate that AdamEaPU and SGDEaPU work well in practice and compare favorably to the original BP (91.83% for AdamEaPU vs 91.76% for Adam and 98.40% for SGDEaPU vs 98.50% for SGD, respectively). Furthermore, when the device is idealized, the ΔW_{th} can be reduced to near zero so that the AdamEaPU degrades to Adam. Even under ideal conditions, EaPU can still substantially reduce the number of update parameters due to the probabilistic update.

https://doi.org/10.1038/s41467-025-66240-7

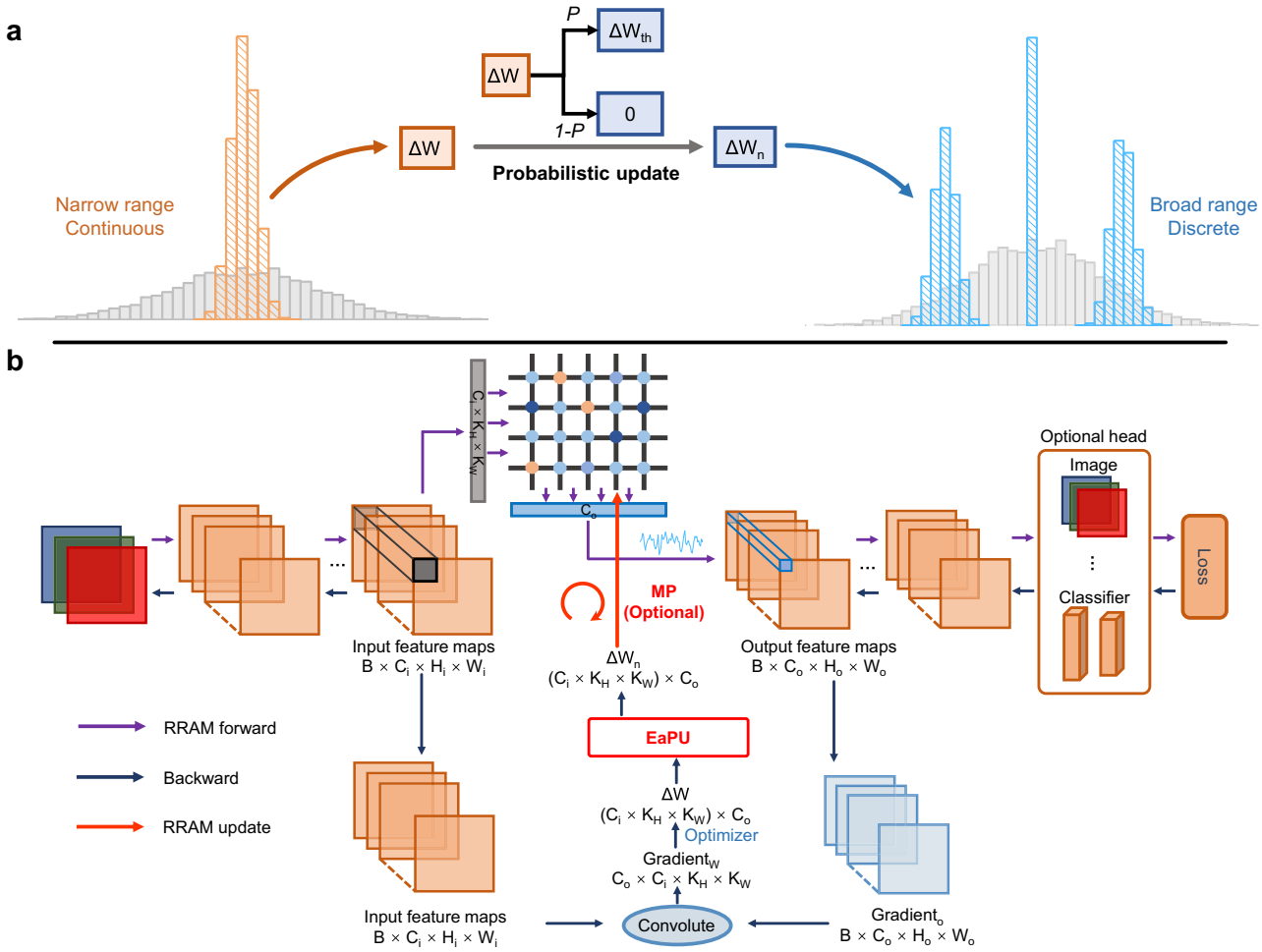

Fig. 2 | The training architecture with EaPU. a Schematic diagram of EaPU. EaPU is introduced to increase the algorithmic update magnitudes to address the discrepancy between larger analog update noise and smaller algorithmic update magnitudes. The updated histogram using EaPU (blue histogram) has become larger and more discrete compared with the original updated histogram (orange histogram), making the desired update magnitudes comparable to the update noise ε_{cell} (brown histogram). **b** Error-aware probabilistic training for learning tasks. The EaPU is used to enlarge the desired update magnitudes ΔW and MP is used to implement efficient updates. During the training process, each layer follows the same method illustrated in the figure.

Error-aware probabilistic training

With the suggested EaPU methods, the training procedure on memristors is shown in Fig. 2b. The memristor performs the most expensive VMM, while digital computing handles the rest. The EaPU is utilized to compute the weight update magnitudes during error back-propagation. The EaPU method can be implemented by introducing a Bernoulli distribution (as shown in Supplementary Fig. 4). Furthermore, since the Bernoulli distribution can be realized using advanced random number generators[44,45], this validates the simplicity and efficiency of hardware implementation for EaPU. Additionally, to achieve an efficient update process, we propose a universal memorial programming method (MP, details in "Methods") to process the spatio-temporal variations. MP maintains the updated simplicity of the two-pulse scheme[18] and the programming precision of the write-verify method[46]. It should be noted that the hardware-level weight update on memristors is not limited to MP, but other update methods can be applied. With error-aware probabilistic training, we can achieve in situ learning on memristors.

Hyperparameter ΔW_{th}

To conduct the hardware experiments, we fabricate the one-transistor-one-resistor (1T1R) TiN/TaO_x/HfO_y/TiN memristor array and develop a corresponding computing system (introduced in "Methods" and Supplementary Fig. 11) with the National University of Defense Technology. We program the device to different target conductance ranges to obtain the ε_{cell}. With a programming tolerance value of 1 μS, the programming noise of the device (programmed conductance point in Supplementary Fig. 12a) is less than 1 μS, while the ε_{cell} (relaxed conductance point in Supplementary Fig. 12a) turns out to be much larger than the programming noise owing to the relaxation. It is confirmed that the relaxation error in the device is the main contributor to ε_{cell}. With an increase in programming tolerance value, the programming noise becomes more significant (shown in Supplementary Fig. 13). By these measurements, we can see that $SD_{\varepsilon_{cell}}$ ranges from 2 to 3 μS, and the standard deviation of relaxation error is around 2 μS in our device. We simulate the ResNet34 with varying ε_{cell} and ΔW_{th} values (details in Supplementary Fig. 14) and find that the ΔW_{th} that yields the best results is consistently around the value of $SD_{\varepsilon_{cell}}$. Therefore, we choose $SD_{\varepsilon_{cell}}$ of devices as ΔW_{th} ($\Delta W_{th} = SD_{\varepsilon_{cell}} \times R_{wg}$, $SD_{\varepsilon_{cell}} = 2$ μS, $R_{wg} = 1/80$ μS^{-1} of our device) for experiments to enhance the training performance. Moreover, there exists a wide range of ΔW_{th} values that can achieve satisfactory performance, allowing us to select an empirical ΔW_{th} for network training. Furthermore, the simulated results of high $SD_{\varepsilon_{cell}}$ (up to 16 μS, shown in Supplementary Fig. 14d) confirm that even though $SD_{\varepsilon_{cell}}$ is 4 orders of magnitude larger than the original update magnitudes used in BP, EaPU still achieves an accuracy of approximately 80%. The robustness of EaPU is evidenced by the impressive results.

https://doi.org/10.1038/s41467-025-66240-7

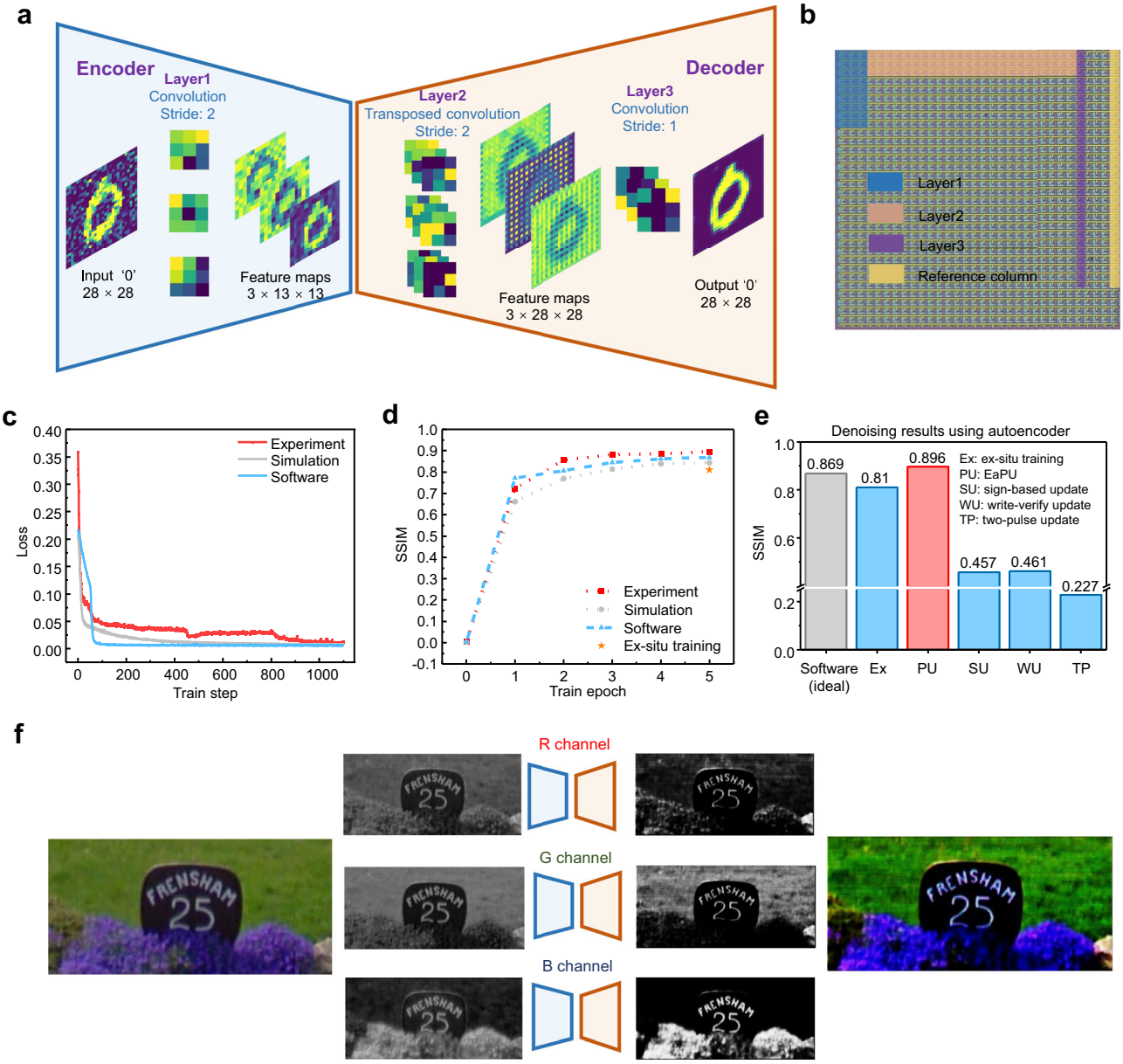

Fig. 3 | Denoising autoencoder tasks on memristor hardware. a Feature map structure of the autoencoder. The first convolutional layer is an encoder, while the last two layers are the decoder model. The activation of ReLU in the interlayer is neglected in the figure. **b** Optical micrographs of the 32 × 32 1T1R array with color blocks to show the different layers of the memristor-based autoencoder. **c** Loss curves during the in situ training process. At the end of the training, the loss of the experiment, simulation, and software is nearly indistinguishable. **d** Variation of SSIM over 5 epochs. Ex situ training point is the inference result after programming trained weights to memristors, while the weights are trained from a GPU using the memristive model. **e** Comparison of denoising results among different methods. The abbreviations mean the different training strategies rather than solely the update method. Training with EaPU achieves better results. **f** Denoising results for colorful images. The color image was split into three channels (R, G, B) and separately transmitted to the trained autoencoder to get a denoised image.

Hardware experiment for memristive autoencoder

To verify the feasibility of the EaPU method on memristors, we first conduct hardware experiments using an autoencoder[47,48], which is a typical multi-point regression problem. In the experiment, the Modified National Institute of Standards and Technology (MNIST) dataset[49] is used to train a denoising autoencoder, following the probabilistic training procedure. Figure 3a illustrates the feature map structure of the autoencoder. The trainable parameters of the autoencoder network are mapped using the dense mapping scheme, as illustrated in Fig. 3b with different color blocks. Existing mapping methods always use two or more cells to implement a signed weight[8,22,38], reducing the utilization rate and increasing the energy costs. To optimize the utilization of memristor cells, the signed weight is encoded as a conductance difference between a reference cell with a fixed conductance in the reference column and a trainable cell with variable conductance, called the reference column mapping method (introduced in "Methods"). By employing this approach, only 162 cells (135 + 27) are required to represent the 135 parameters of the autoencoder network, enhancing the energy efficiency of the training and inference process.

The training flow of the autoencoder follows the error-aware probabilistic training, and training configurations are introduced in "Methods." During the training process, the loss over the training steps is presented in Fig. 3c, and the SSIM of the training epochs is shown in Fig. 3d. To obtain the simulation results, the autoencoder with memristor features is simulated, taking into account the nonidealities of

memristors. The hardware experiment (SSIM: 0.896) and noise-free software simulation (SSIM: 0.869) achieve similar performance, confirming the ability of the memristor autoencoder to learn the various nonidealities during the training procedure. The memristor noise might improve the training robustness and performance of experimental results[33], causing better results. Supplementary Fig. 15 shows the traditional peak signal-to-noise ratio (PSNR, another metric) index of the autoencoder, where the experimental result (PSNR: 19.99) achieves nearly the same PSNR as the simulation (PSNR: 19.84). To verify the effectiveness of EaPU, we implement the error-aware probabilistic training, sign-based update method with BP, write-verify update method with BP, and two-pulse scheme with BP separately on the memristor system to train an autoencoder and compare them, as shown in Fig. 3e and Supplementary Fig. 16. The experimental results show that better results (SSIM:0.896, >80% improvements of SSIM to other training methods) are achieved with EaPU, due to the matching of the update process.

Supplementary Fig. 17a, b confirms the denoising ability of the trained memristor-based autoencoder by displaying the denoising result. We apply uniform noise and bicubic interpolation noise to the images, as shown in Supplementary Fig. 17c, further confirming the denoising capability of the trained autoencoder on other types of noise. Subsequently, we apply the trained autoencoder to denoise an image sourced from the SVHN (Street View House Numbers) dataset[50], mitigating various forms of noise such as compression artifacts and image editing distortions, and achieve the denoising image in Fig. 3f. Moreover, Supplementary Fig. 18 presents the data compression result achieved by a trained memristor-based autoencoder (compression rate is 64.67%). Furthermore, an important application of autoencoders is anomaly detection, introduced in Supplementary Fig. 19, achieving a classified accuracy of 96.56% to detect MNIST and Fashion MNIST images.

Hardware experiment for memristive super-resolution networks

In this part, a larger noise-sensitive task, super-resolution (SR)[51,52], is chosen to validate the effectiveness of EaPU on memristor hardware. Figure 4a illustrates the feature-map-based architecture of the super-resolution network with an upscaling factor of 2× (SRNet ×2). Similar to the autoencoder, the dense mapping scheme is employed in the super-resolution network, as shown in Fig. 4b. As the transposed convolutional layer of SRNet ×2 results in a large weight size after dense mapping, we divide its weight into two parts, namely layer2-1 and layer2-2, for separate mapping. Besides, through the reference column mapping method, 396 cells (369 + 27) are required to represent the 369 parameters rather than 738 (369 × 2) cells, reducing nearly half the number of cells.

The training process of the SRNet ×2 follows the error-aware probabilistic training, and the training configurations are introduced in "Methods." Figure 4c illustrates the loss curve of the training process during 400 training steps, and Fig. 4d shows the variation of SSIM with epochs, confirming that the experimental results (SSIM:0.933) closely align with the noise-free software-trained results (SSIM: 0.958). We further compare the error-aware probabilistic training with alternative training methods for SRNet ×2, as illustrated in Fig. 4e and Supplementary Fig. 20, achieving better results with EaPU (SSIM: 0.933, >0.6 improvements of SSIM to the classical two-pulse scheme with BP). This result confirms the robustness and effectiveness of EaPU in the noise networks. Besides, Fig. 4f, g displays the results using the trained memristor-based SRNet ×2 for two-times image super-resolution, and Supplementary Fig. 21 shows the further tasks on SRNet ×4.

To discuss the efficiency of error-aware probabilistic training, we gather the update pulse number in different programming strategies during the aforementioned training process (shown in Fig. 4h and Supplementary Fig. 22). Due to the probabilistic updates of cells, by using the EaPU, only around 6% parameters are updated during the training process and the rest parameters are unchanged, shown in Supplementary Fig. 22b, c, reducing the programming complexity. Thus, the combination of EaPU and MP significantly reduces the number of cells required for updating and results in an N_{up} of less than 1 (~0.7) per device, reducing the N_{up} by 10 times compared to MP alone and 90 times compared to the write-verify method (WU). When the fast-programming scheme (FP), including the sign-based update (SU) and two-pulse scheme (TP), is integrated with EaPU, it is observed that the N_{up} (0.1) decreases by a factor of ten compared to the fast-programming scheme alone. When training with deep networks, N_{up} decreases further, thereby yielding great energy consumption advantages, which are analyzed in detail below.

Simulation results on advanced networks

Given that previous experiments have confirmed EaPU's good training performance and efficiency on memristor hardware, we further conduct simulations on advanced networks to verify the feasibility and effectiveness of EaPU in deep networks.

We simulate the ResNet34 (34 layers) with the CIFAR10 dataset and SRResNet ×4 network[30] (37 layers, details in Supplementary Fig. 23) with the ImageNet dataset to verify the enhancements and robustness of EaPU. To analyze the impact of ε_{cell}, we implement simulated experiments with the original BP and EaPU, shown in Fig.5a. To be precise, the two-pulse scheme, write-verify update, and sign-based update can all be equivalent to BP learning algorithms with update noise. The dashed lines in Fig. 5a depict the training results of BP for ResNet34 and SRResNet, highlighting the impact of large ε_{cell} on the original BP algorithm. The arrow in Fig.5a further confirms the substantial improvement offered by the proposed EaPU methods (For example, EaPU achieves an improvement of 73% in accuracy and 0.84 in SSIM for ResNet34 and SRResNet, respectively. The $SD_{\varepsilon_{cell}}$ is 2 μS and R_{wg} is 1/80 μS^{-1}). The robustness of EaPU to large ε_{cell} relaxes the requirements for device performance. Meanwhile, the simulation with different R_{wg} (shown in Fig. 5b) further confirms the robustness and insensitivity of EaPU to conductance range, achieving an improvement of 70% in accuracy and 0.79 in SSIM for ResNet34 and SRResNet corresponding to the original algorithm, respectively, when R_{wg} is 1/20 μS^{-1}. Due to the great robustness to ε_{cell} and the insensitivity to R_{wg}, EaPU can be effectively applied to analog devices with various update noise and conductance range, reducing the hardware constraints of training. Furthermore, we supplement simulations of ResNet34 with respect to unresponsive devices (shown in Supplementary Fig. 24), confirming that the network can still achieve a training accuracy of 85% in the presence of a 10% stuck-off ratio and a 2% stuck-on ratio.

As to very deep networks or complex datasets, we simulate ResNet152 with the CIFAR100 dataset and Swin Transformer[32] networks with the ImageNet100 dataset and achieve the experimental outcomes presented in Fig. 5c and Supplementary Fig. 25. These results align with those in Fig. 5a, showing that the original BP algorithm struggles to train effectively in the presence of significant weight update noise, while EaPU effectively addresses this issue and the accuracies are improved by 60.23% and 88.78% for ResNet152 and Swin Transformer, respectively. Moreover, as shown in Fig. 5a, c, the training results of the noiseless model with EaPU can achieve negligible performance loss (<1%) in comparison with the original BP. The aforementioned hardware tests have confirmed the sparse update characteristic of the EaPU strategy. Given that the sparse update characteristic is of greater practical significance for deep networks, we statistically analyze the parameter update ratios during the training of ResNet152 and Swin Transformer. During the training process of ResNet152, we observe that the ratio of parameter updates (shown in Fig. 5d) follows a pattern of first increasing and then decreasing, with such updates generally remaining below 1.4‰ (the number of updated parameters is reduced by 99.86% at least), while the training accuracy

https://doi.org/10.1038/s41467-025-66240-7

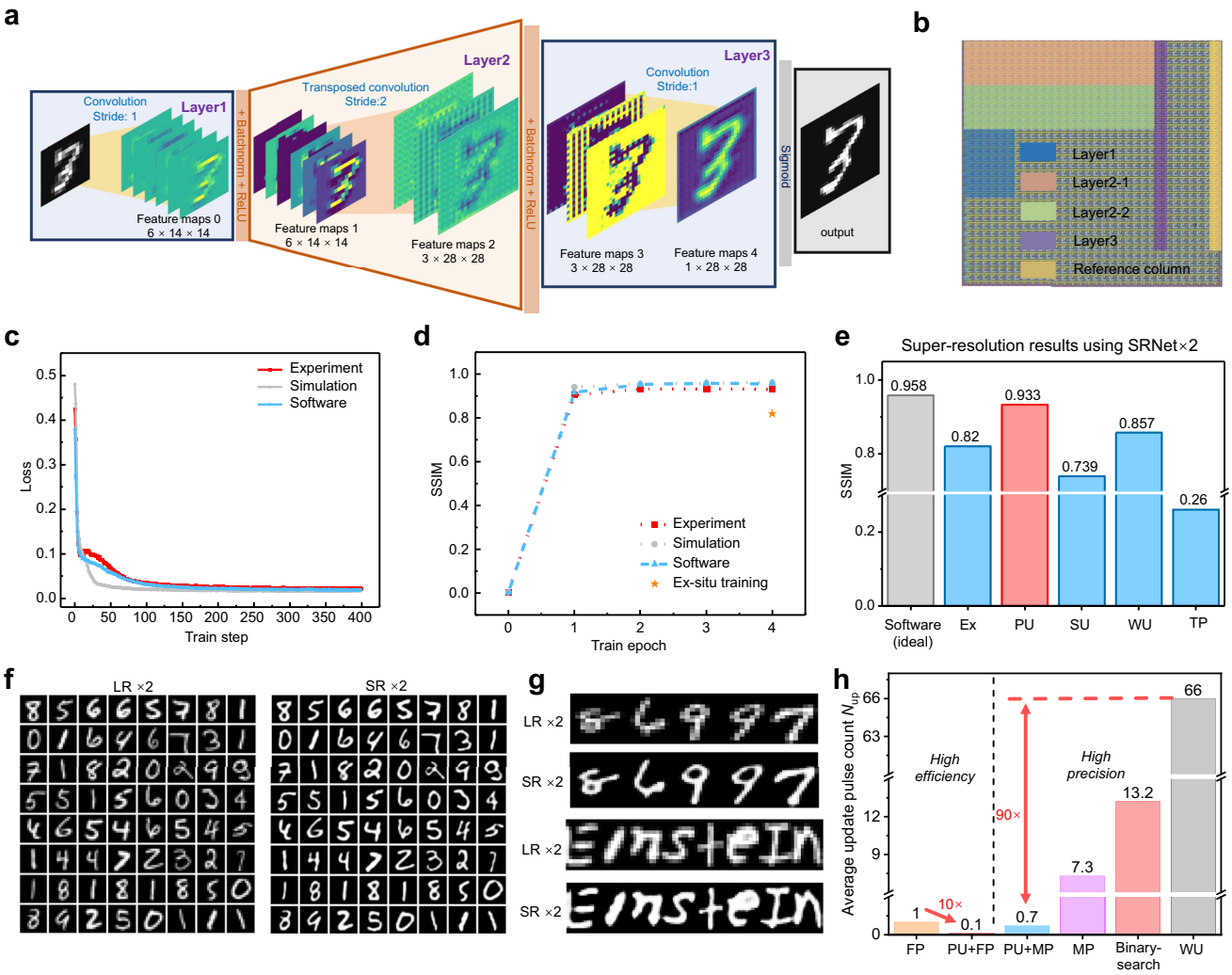

Fig. 4 | The implementation of SRNet ×2 on memristor hardware. a Feature-map-based structure of SRNet ×2. **b** Optical micrographs of the 1T1R array with colored blocks to illustrate the partitions to implement the trainable parameters of the two convolutional layers and a transposed convolutional layer. **c** Loss curve of the network training process. **d** SSIM results of software, simulation, and experiment over 4 epochs. Ex situ training point is the inference result after programming trained weights to memristors. **e** Comparison of training results among different methods in memristors, where training with EaPU achieves the best result, expects an ideal result. The abbreviations in this figure are the same as in Fig. 3e (Ex ex situ training, PU EaPU, SU sign-based update, WU write-verify update, TP two-pulse update). **f** The two-times image super-resolution results using the trained memristor-based SRNet ×2. **g** The two-times image super-resolution results using memristor-based SRNet ×2 for concatenating image "886997" and "Einstein." **h** Average update pulse count N_{up} of various programming methods. EaPU + MP achieves the precision of the write-verify method and a 90-time reduction of the average pulse (0.7 vs 66). Moreover, EaPU + FP achieves a 10-time reduction of the average pulse than the fast-programming scheme (0.1 vs 1).

continues to rise throughout the process. Moreover, the average update ratio R_{up} reaches 0.86‰ during the training process (shown in Fig. 5e). This is equivalent to increasing the number of maximum updates of the memristor system by three orders of magnitude, further extending the service life of memristors and alleviating the constraints imposed by endurance. The Supplementary Fig. 26 further validates the effectiveness of the sparse update characteristic of EaPU. During the transfer learning process of Swin Transformer, the parameter update ratio and R_{up} decrease to less than 0.15‰ and 0.04‰ (shown in Supplementary Fig. 27), respectively. The statistic of Swin Transformer confirms that the use of pre-trained weights can reduce the proportions of parameter updates to a great extent. The ultra-low R_{up} results in an order-of-magnitude reduction in N_{up} (shown in Fig. 5f), thereby significantly lowering the energy consumption of the update process (see "Discussion").

We further test and simulate the long-term retention characteristics of the network after training. Given that the long-term retention characteristics of the trained network are closely related to device properties, we discuss this from two aspects, including the RRAM system and the PCM system. Firstly, we measure the conductance changes of 4000 programmed RRAM devices over time on our hardware, as shown in Supplementary Fig. 28. It shows that the conductance of RRAM devices does exhibit a drift over time. The conductance distribution of the RRAM only undergoes a mean shift, which can be restored to the original distribution by adding a mean offset, as shown in Supplementary Fig. 28f. We further test the accuracy over time of a fully connected network (64-32-24-10) under different methods on RRAM hardware, as shown in Supplementary Fig. 29, confirming that the EaPU-trained model without any compensation can address the drift issue in RRAM and maintain accuracy for more than 60 h. Then, we conduct the relevant simulation in the PCM system using AIHWKit[53,54]. Due to the complex drift characteristics of PCM, GDC plays an important and indispensable role[11,19]. By introducing GDC to the EaPU-trained ResNet32 model, the inference accuracy of the network can be effectively maintained, as shown in Supplementary Fig. 30.

https://doi.org/10.1038/s41467-025-66240-7

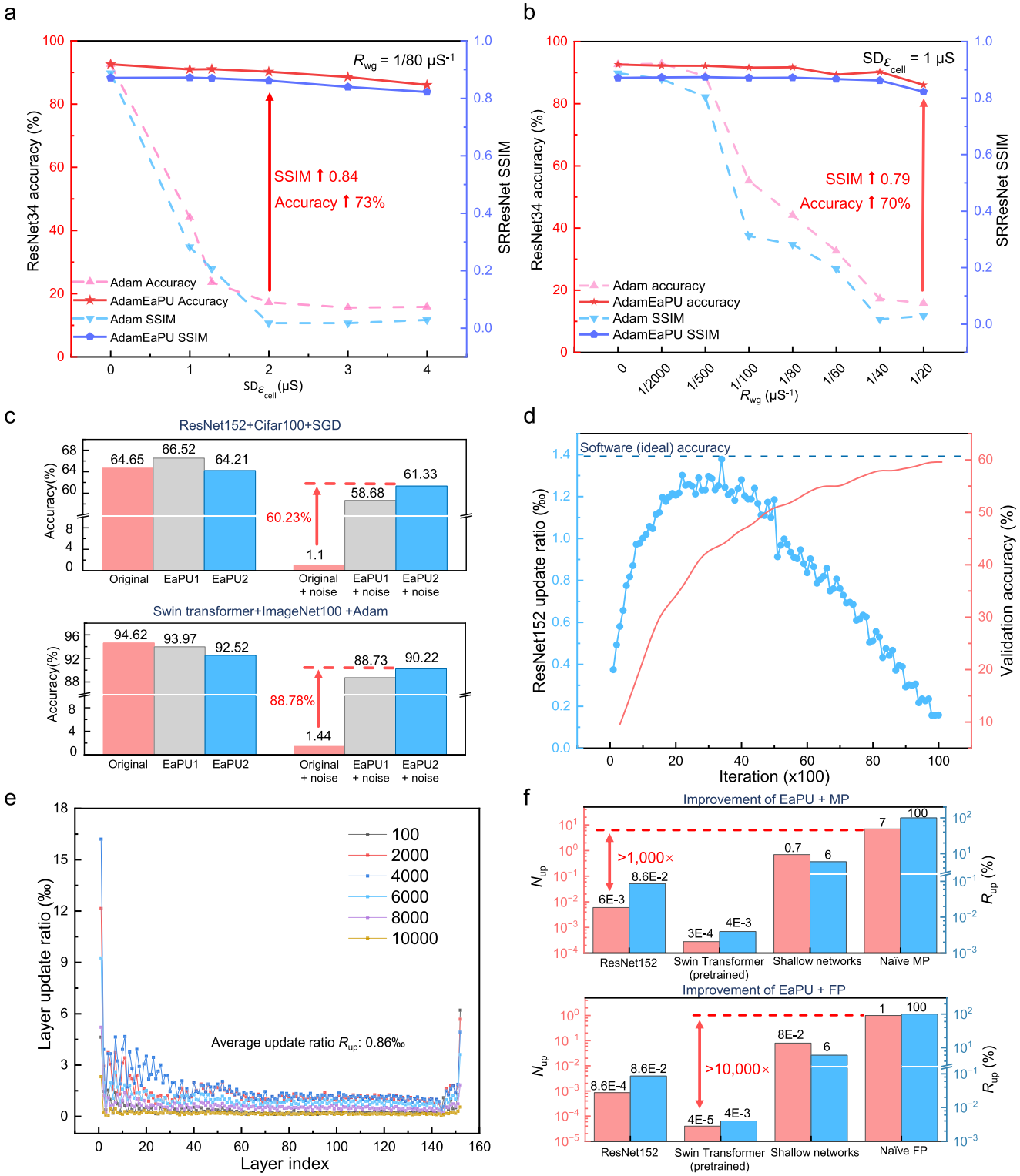

Fig. 5 | Simulation on deep networks. a Training results of ResNet34 and SRResNet with different ε_{cell} and improvement of the suggested EaPU. As the increase of ε_{cell}, the performance of the original algorithm declines dramatically, while our suggested EaPU achieves near-invariable performance. **b** Training process of ResNet34 and SRResNet with different R_{wg}. A larger R_{wg} indicates a narrower conductance range, and $R_{wg} = 0$ means the noiseless training results. With $SD_{\varepsilon_{cell}}$ kept constant, the performance of the original algorithm degrades rapidly as R_{wg} increases, whereas training with EaPU maintains stable performance. **c** Training results of simulated ResNet152 and Swin Transformer. Although the training results of the nonideal model ($SD_{\varepsilon_{cell}}$ is 2.4 μS and R_{wg} is 1/80 μS^{-1}) decline by around 3% or 4% when using EaPU, they are much better than the original algorithm with the nonideal model (61.33% vs 1.1% and 90.22% vs 1.44%). **d** Update the ratio of ResNet152 during the training process. During the training process, the update ratio is smaller than 1.4‰. **e** Update the ratio of all the layers in ResNet152. The legend represents the number of training iterations. **f** The reduction in R_{up} and N_{up} achieved by EaPU. Adopting EaPU reduces N_{up} by over three orders of magnitude (with pre-trained weights, a further reduction of four orders of magnitude can be achieved), which can contribute to lowering training energy consumption.

https://doi.org/10.1038/s41467-025-66240-7

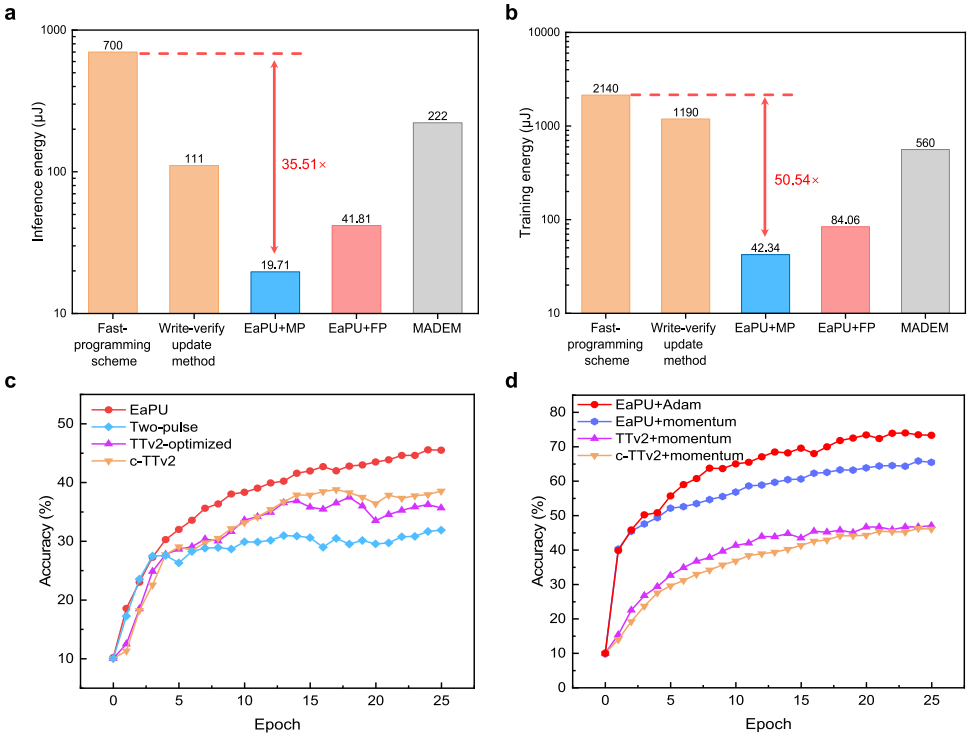

Fig. 6 | Energy consumption comparison with related training methods and performance comparison with different memristor training approaches. **a** Inference energy consumption comparison. **b** Training energy consumption comparison. Whether in inference or training, EaPU + MP achieves the lowest energy consumption while attaining a several dozen-fold improvement in energy efficiency compared to others. **c** Training process of ResNet34 with naïve SGD. The CIFAR10 dataset is used for the ResNet34 training. The hyperparameters are optimized in TTv2 and extend to c-TTv2. **d** Training process of ResNet34 with an advanced optimizer. Here, TTv2+momentum and c-TTv2+momentum apply momentum characteristics to layers such as batch normalization (batchnorm) layers. By comparison, EaPU enables a stable learning process while allowing advanced training techniques to further enhance training performance (achieving ~30% improvement) than naïve SGD with EaPU.

Discussion

Due to the insensitivity to R_{wg}, EaPU can achieve comparable accuracy for analog devices with much narrower conductance ranges, reducing the hardware constraints for in situ training. As shown in Supplementary Table 2 of Supplementary Note 2, EaPU with R_{wg} of 1/40 μS^{-1} achieves better accuracy than the original BP algorithm with R_{wg} of 1/4000 μS^{-1}, reducing the energy consumption (shown in Supplementary Note 2.3). Owing to its ability to learn efficiently within the narrow training conductance range, the inference energy consumption of the EaPU-trained model is reduced, achieving 35.51× lower than the fast-programming scheme (700 μJ vs. 19.71 μJ) and 11.26× lower than advanced MADEM (222 μJ vs. 19.71 μJ), as shown in Fig. 6a. Furthermore, EaPU yields an ultra-low update characteristic (with R_{up} being 0.86‰ in ResNet152), which can significantly reduce the update frequency (with N_{up} being 6×10^{-3} in the configuration of EaPU + MP) and thereby lower the energy consumption of the update process (more than 100× than related training method, shown in Supplementary Table 3 of Supplementary Note 2). Owing to low inference and update energy consumption, EaPU-based training exhibits great training advantages, with an energy consumption reduction of 50.54× compared to the fast-programming scheme (and 13.23× compared to the advanced MADEM), as shown in Fig. 6b. Furthermore, a comparison between EaPU-based systems and GPU is performed, introduced in Supplementary Note 3. In Flash memory systems where write energy consumption and latency are substantial, EaPU can achieve over three orders of magnitude reduction in update energy consumption and an 18.69× decrease in update latency. This ultimately results in a two-orders-of-magnitude reduction in training energy consumption and an 18.19× decrease in training latency, confirming the positive significance of extending EaPU to other nonvolatile devices. Meanwhile, training with high-energy-efficiency memristor systems yields better performance, enabling nearly a 6-order-of-magnitude reduction in training energy consumption and nearly a 2-order-of-magnitude reduction in training latency compared with GPUs.

We further compare EaPU with other distinct memristor training approaches using AIHWKit, including Tika-Takav2 (TTv2)[55], chopped-TTv2 (c-TTv2)[56], and hardware-aware training (Hwa)[11]. EaPU, TTv2, and c-TTv2 are memristor-based in situ training, and they all exhibit favorable learning capabilities on the LeNet5 task; however, EaPU achieves faster convergence efficiency, as shown in Supplementary Fig. 31. The two-pulse scheme demonstrates the fastest convergence efficiency in LeNet5. Nevertheless, its sensitivity to update noise limits training accuracy. For deeper networks such as ResNet34, inappropriate hyperparameters can cause TTv2 and c-TTv2 to suffer from learning instability (as shown in Supplementary Fig. 32). It is necessary to perform fine-grained hyperparameter tuning for TTv2 and c-TTv2, which incurs optimization costs. By adjusting the threshold of the H matrix, the unstable learning problem can be effectively addressed (as shown in Supplementary Fig. 33). However, TTv2 and c-TTv2 require a certain degree of accumulation of gradient magnitudes and can only perform weight updates after reaching the threshold. This results in no weight updates within a certain number of iterations, which reduces the learning efficiency of the algorithm (as shown in Fig. 6c). TTv2 and c-TTv2 employ digital modules for the cumulative computation of H arrays, which endow them with momentum-like properties and enhance their learning capability (see Supplementary Fig. 34 for further optimization). By additional optimizations and fine parameter tuning, they may achieve training performance comparable to that of naive SGD. However, they fail to match the accuracy of momentum-based

training methods (as shown in Fig. 6d) and struggle to effectively accommodate more advanced optimizer strategies (e.g., Adam). In contrast, EaPU is compatible with such advanced optimizer strategies, thereby achieving better training performance. Furthermore, in simulations related to EaPU, we don't employ any optimization methods other than the one detailed in the "Hyperparameter ΔW_{th}" section. Hwa is an approach that trains the noise model in software (GPUs or CPUs) to improve noise robustness and programs the trained weights onto analog hardware[56]; thus, it is mainly used for the inference application and cannot leverage the energy and latency advantages offered by memristor in situ training. Meanwhile, even in the presence of update noise, EaPU in memristor system can achieve learning capability comparable to that of Hwa in digital system, as shown in Supplementary Fig. 35a. Compared with Hwa in digital systems, EaPU in memristor training systems exhibit substantial advantages in energy consumption and latency (analyzed in Supplementary Note 3), with order-of-magnitude accuracy gains per unit of energy and per unit of latency (as shown in Supplementary Fig. 35b, c). This confirms the ultra-high energy-accuracy and latency-accuracy efficiency of EaPU-based training. Thus, under energy and latency constraints, EaPU can achieve training iterations that are multiple orders of magnitude greater than those of Hwa, thereby enabling higher learning performance.

In summary, EaPU training achieves robustness to update noise and insensitivity to conductance ranges, compensating for the limitations of in situ memristor training. Thus, it can handle various nonideal characteristics, supporting the precise and efficient training of DNNs in analog devices (such as ResNet152 and Swin Transformer). Meanwhile, due to the probabilistic update characteristic of EaPU, the number of updated parameters during training is significantly reduced (down to 0.86‰ for ResNet152 and 0.04‰ for Swin Transformer), which effectively lowers the energy consumption of the update process (a three-order-of-magnitude reduction in Flash memory) and increases the number of maximum iterations of the memristor system. These reasons make the EaPU promising to be extended to various nonvolatile device systems. Due to the ultra-low operating conductance range and N_{up}, EaPU can achieve a several dozen-fold reduction in training and inference energy consumption compared with related memristor training strategies. Coupled with the ultra-high energy efficiency of memristors, EaPU is expected to achieve a nearly six-order-of-magnitude reduction in energy consumption compared with GPUs. Furthermore, EaPU-trained models can exhibit good retention characteristics, and in particular, no additional compensation strategies are required for RRAM systems. Comparisons with training strategies from other distinct approaches have further confirmed EaPU's good and balanced performance in learning efficiency, training stability, training performance, algorithm compatibility, energy-accuracy efficiency, and latency-accuracy efficiency, further enhancing the performance of analog training. The EaPU method holds promise for combination with other distinct learning approaches to achieve further improvements, thereby advancing progress in the field of memristor training.

Methods

Quantitative index for memristor error

Memristor precision limitation can be described as the calculation deviation of VMM[8]. VMM error ε_{VMM} is utilized to quantify the variance between the actual output y_{actual} and desired output y_{fp}, and can be described as follows:

$$\varepsilon_{VMM} = y_{actual} - y_{fp}. \tag{5}$$

In this formula, $y_{fp} = W_{desired}X$, where $W_{desired}$ is the desired target conductance for memristors and X describes the calculated input. ε_{VMM} can be decomposed as the linear error ε_{linear} from incorrect programming, and residual error $\varepsilon_{residual}$ from calculation errors[8], including reading noise, IR drop, and external circuit nonidealities. ε_{linear} and $\varepsilon_{residual}$ are defined as:

$$\varepsilon_{VMM} = \varepsilon_{linear} + \varepsilon_{residual}, \tag{6}$$

$$\varepsilon_{linear} = W_{actual}X - y_{fp} = (W_{acutal} - W_{desired})X, \tag{7}$$

$$\varepsilon_{residual} = y_{actual} - W_{acutal}X, \tag{8}$$

where W_{actual} is the actual conductance on memristors. Herein, we use writing noise ε_{cell} ($\varepsilon_{cell} = W_{acutal} - W_{desired}$) to represent the ε_{linear} for obtaining measured metrics. Following the Eqs. (5), (7), and (8), Supplementary Fig. 1 displays the visual representation of the differences among ε_{VMM}, $\varepsilon_{residual}$, and ε_{cell}. Here, we mainly illustrate the constraint factors of devices during the training process, while factors influencing the long-term retention process (such as long-term drift, discussed in the "Simulation results on advanced networks" section) are not included.

Average update pulse count N_{up}

N_{up} refers to the average number of applied update pulse counts per device per iteration during the training process, which facilitates energy consumption calculation and evaluation. Assuming that the total number of update pulses applied in each iteration is N_{all} and the total number of devices is n, then

$$N_{up} = \frac{N_{all}}{n}. \tag{9}$$

Meanwhile, N_{up} can be derived from the average number of update pulses per device per iteration (N_{cell}, counting only the devices that undergo updates) and the average update ratio (R_{up}), as follows:

$$N_{up} = N_{cell} \times R_{up}. \tag{10}$$

Related training approaches and their limitations

Typical training approaches are a combination of learning algorithms such as backpropagation and update schemes. To implement the conductance update in analog devices, various conductance update schemes for memristive training have been proposed yet, such as the two-pulse scheme[18,34], sign-based scheme[10,57], and write-verify scheme[21]. The two-pulse scheme employs a pair of set and reset pulses to achieve a linear conductance response. However, the requirement for highly linear and symmetric devices to decrease ε_{cell} limits its applicability[18,33]. Sign-based update methods apply the voltage whose polarity relies on the sign of update magnitudes to the memristors. Sign-based update methods are sensitive to operation voltage, necessitating careful operations to prevent the large conductance change[57]. These two methods bring about high efficiency but cause more or less loss of programming precision. Moreover, traditional write-verify methods combine a few pulses to adjust the conductance and estimate whether the current conductance is required. Though traditional write-verify strategies are efficient methods to decrease ε_{cell}[21,24], the low-parallel operation and spatiotemporal variations ε_{var} make existing write-verify strategies highly time- and energy-cost. Therefore, there exists an unbalance between precision and efficiency in existing update methods, then limiting the training performance. The proposed EaPU can overcome the limitations of unbalance, achieving robust training and better performance.

Definition for $\triangle W$, $\triangle W_{th}$, and $\triangle W_n$

During the training process, the naïve stochastic gradient descent (SGD) update step can be described as follows:

$$w_{ij}^{t+1} = w_{ij}^{t} - \eta \sum_{b=1}^{B} x_i^b \delta_j^b \tag{11}$$

Where t is the iteration number, η is the learning rate, and b is the index of batch size. x_i^b is the activation at the input layer, and δ_j^b is the error computed by the output layer. Thus, $\triangle$W denotes the update magnitudes $-\eta \sum_{b=1}^{B} x_i^b \delta_j^b$. Meanwhile, in more complex update strategies, the update magnitudes may include components such as momentum and second-order gradients. $\triangle W_{th}$ is defined as the threshold in the EaPU training process, which can be simply set as $SD_{\varepsilon_{cell}}$ ($SD_{\varepsilon_{cell}} \times R_{wg}$, specifically). $\triangle W_n$, the target update magnitudes processed by EaPU can be directly used in the memristor update process.

Transformation of the original optimizer to the suggested optimizer

Composing the original optimizer with EaPU requires only a few steps:

Step 0: define required learning hyperparameters, such as learning rate α, threshold ΔW_{th};

Step 1: initialize the trainable parameters θ;

Step 2: calculate the update magnitudes ΔW through backpropagation;

Step 3: calculate the target update magnitudes ΔW_n through Formula (4);

Step 4: update parameters.

By repeating Steps 2–4, we finally obtain converged trainable parameters.

Memorial programming method

The memorial programming method (MP) is modified from the two-pulse scheme[18] that maintains the original scheme's update simplicity and the write-verify[21] method's programming precision. As shown in Supplementary Fig. 5, MP is composed of binary-search and step-to-step update procedures, where the binary-search is employed to load pre-trained weights or initial weights from existing initialization methods to achieve enhanced training outcomes. The binary-search procedure, detailed in Supplementary Fig. 6, addresses the issues caused by device-to-device variation (limiting the uniform programming parameters, shown in Supplementary Fig. 7) and nonlinearity (posing a challenge for the linearity dependence programming methods, shown in Supplementary Fig. 8) by only requiring a positive correlation between the conductance and the gate control voltage. Since the update magnitudes of parameters during network training are generally small, we can use the voltage position updated in the previous training step as the initial voltage position for the current training step and update the parameters by the step-to-step update procedure (The flow diagram is illustrated in Supplementary Fig. 9). Cycle-to-cycle variation (shown in Supplementary Fig. 10) allows the same control voltage to correspond to a conductance range, which improves the efficiency of the step-by-step update procedure, as only a few pulses are needed to reach the target conductance. As the position search is only performed in the first training step, the time cost associated with the binary search becomes negligible when conducting hundreds, thousands, or even more training steps, greatly reducing the complexity of using the MP. By comparing the programming complexity in the Supplementary Table 1, it can be observed that the MP has the same complexity as the two-pulse scheme during network training.

Fabrication and integration of 1T1R array

The transistor array was fabricated in a commercial foundry using the 180 nm technology node (SMIC 1P6M). The metal layers M1 to M5 and the vias V1 to V5 were manufactured in the foundry. Subsequently, the resistive layer comprising TiN/TaO_x/HfO_y/TiN and the metal layer M6 were integrated in a laboratory cleanroom. The process involved the deposition of a 30-nm TiN bottom electrode using physical vapor deposition. Following this, 8-nm HfO_y and 45-nm TaO_x were grown with the atomic layer deposition method. The top TiN electrode was then stacked to 30 nm using physical vapor deposition. Finally, the metal layer was fabricated via sputtering under a suitable vacuum environment. In the 1T1R cell configuration, transistors were employed to mitigate the sneak path problem and implement precise conductance tuning, while memristors implemented nonvolatile storage. The computing system was designed to control the 1T1R array and conduct experiments on neural networks. Further details about the measuring system can be found in Supplementary Fig. 11.

Reference column mapping method

In memristor-based neural networks, the relationship between the real domain training parameter W and the memristor training parameter $\mathbf{G}$ is

$$W = R_{wg} \times \mathbf{G}, \tag{12}$$

where R_{wg} is a positive number and $\mathbf{G}$ is a signed matrix. When conductance is used to represent weights, since the conductance of memristors is always positive, we need to represent memristor weights by taking the difference. In our experiment, we represent $\mathbf{G}$ as follows:

$$\mathbf{G} = \mathbf{G}_{trainable} - \mathbf{G}_{zero}, \tag{13}$$

where $\mathbf{G}_{zero}$ is the bias to keep $\mathbf{G}_{trainable}$ positive. $\mathbf{G}_{zero}$ is a fixed matrix in our experiments and is represented by a memristor column (called the reference column). After obtaining the $\Delta\mathbf{G}$, we only need to update the memristor parameters corresponding to $\mathbf{G}_{trainable}$, reducing the number of parameter updates by half. Using a single memristor cell to represent signed weights means that G_{max} is only half of a single memristor conductance range, making the impact of the ε_{cell} more pronounced.

Training configurations of the autoencoder

During the training process, the Gaussian noise with a standard deviation of 0.3 is added to the original image as the input image, while the original image is set as the target image. The mean square error is used to calculate the loss and activate the backward pass. The training consists of 1100 mini-batches, and we choose 5 points with an interval of 200 mini-batches to validate the trained results. Moreover, the mini-batch size used in the training dataset is 64.

Training configurations of SRNet ×2

During the training process, we employ the bicubic method to downscale the original images to half of their size as the input data, while retaining the original images as the target images for training. The L1 loss calculates the output loss value used in backward propagation. Moreover, the training involves 400 mini-batches across 4 epochs, with each epoch containing 100 mini-batches, and the mini-batch size is 256.

Data availability

The datasets used for the experiments and simulations in this study are publicly available[29,49,50]. The MNIST dataset is available at http://yann.lecun.com/exdb/mnist/. The SVHN dataset is available at http://ufldl.stanford.edu/housenumbers/. The CIFAR datasets are available at

https://www.cs.toronto.edu/~kriz/cifar.html. Source data are provided with this paper.

Code availability

The code used to simulate the model with EaPU in this study is available at https://github.com/LJinchang/Experiment_EaPU[58]. The code that supports the experiments on memristors relies on the custom-built measurement system and is available from the corresponding authors upon request.

References

1. 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).
2. Huang, G., Liu, Z., Van Der Maaten, L. & Weinberger, K. Q. Densely connected convolutional networks. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition* 4700–4708 (IEEE,2017).
3. Ren, S., He, K., Girshick, R. & Sun, J. Faster R-CNN: towards real-time object detection with region proposal networks. In *Proc. of the 29th International Conference on Neural Information Processing Systems* Vol. 1, 91–99 (MIT Press, 2015).
4. Redmon, J. & Farhadi, A. Yolov3: an incremental improvement. Preprint at *arXiv* https://doi.org/10.48550/arXiv.1804.02767 (2018).
5. Brown, T.B. et al. Language models are few-shot learners. In *Proc. of the 34th International Conference on Neural Information Processing Systems* Vol. 33, 1877–1901 (Curran Associates Inc., 2020).
6. Achiam, J. et al. GPT-4 technical report. Preprint at *arXiv* https://doi.org/10.48550/arXiv.2303.08774 (2023).
7. Jouppi, N. et al. TPU v4: an optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *Proc. of the 50th Annual International Symposium on Computer Architecture* 82 (Association for Computing Machinery, 2023).
8. Le Gallo, M. et al. A 64-core mixed-signal in-memory compute chip based on phase-change memory for deep neural network inference. *Nat. Electron.* **6**, 680–693 (2023).
9. Wan, W. et al. A compute-in-memory chip based on resistive random-access memory. *Nature* **608**, 504–512 (2022).
10. Zhang, W. et al. Edge learning using a fully integrated neuro-inspired memristor chip. *Science* **381**, 1205–1211 (2023).
11. Rasch, M. J. et al. Hardware-aware training for large-scale and diverse deep learning inference workloads using in-memory computing-based accelerators. *Nat. Commun.* **14**, 5282 (2023).
12. Feng, Y. et al. Memristor-based storage system with convolutional autoencoder-based image compression network. *Nat. Commun.* **15**, https://doi.org/10.1038/s41467-024-45312-0 (2024).
13. Yao, P. et al. Fully hardware-implemented memristor convolutional neural network. *Nature* **577**, 641–646 (2020).
14. Yang, J. et al. Resistive memory-based neural differential equation solver for score-based diffusion model. Preprint at *arXiv* https://doi.org/10.48550/arXiv.2404.05648 (2024).
15. Chen, Y. Y. et al. Understanding of the endurance failure in scaled HfO 2-based 1T1R RRAM through vacancy mobility degradation. In *Proc. 2012 International Electron Devices Meeting*. 20.23. 21-20.23. 24 (IEEE, 2012).
16. Xu, X. et al. 40× Retention Improvement by Eliminating Resistance Relaxation with High Temperature Forming in 28 nm RRAM Chip. In *Proc. 2018 IEEE International Electron Devices Meeting*. 20.21.21-20.21.24 (IEEE, 2018).
17. Laube, S. M. & TaheriNejad, N. Device variability analysis for memristive material implication. Preprint at *arXiv* https://doi.org/10.48550/arXiv.2101.07231 (2021).
18. Li, C. et al. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nat. Commun.* **9**, 2385 (2018).
19. Joshi, V. et al. Accurate deep neural network inference using computational phase-change memory. *Nat. Commun.* **11**, 2473 (2020).
20. Yi, S.-i, Kendall, J. D., Williams, R. S. & Kumar, S. Activity-difference training of deep neural networks using memristor crossbars. *Nat. Electron.* **6**, 45–51 (2022).
21. Wang, R. et al. Implementing in-situ self-organizing maps with memristor crossbar arrays for data mining and optimization. *Nat. Commun.* **13**, 2289 (2022).
22. Kiani, F., Yin, J., Wang, Z., Yang, J. J. & Xia, Q. A fully hardware-based memristive multilayer neural network. *Sci. Adv.* **7**, eabj4801 (2021).
23. Shi, T. et al. Stochastic neuro-fuzzy system implemented in memristor crossbar arrays. *Sci. Adv.* **10**, eadl3135 (2024).
24. Rao, M. et al. Thousands of conductance levels in memristors integrated on CMOS. *Nature* **615**, 823–829 (2023).
25. Feng, Y. et al. Improvement of state stability in multi-level resistive random-access memory (RRAM) array for neuromorphic computing. *IEEE Electron Device Lett.* **42**, 1168–1171 (2021).
26. Scellier, B., Ernoult, M., Kendall, J. & Kumar, S. Energy-based learning algorithms for analog computing: a comparative study. In *Proc. of the 37th International Conference on Neural Information Processing Systems* 2295 (Curran Associates Inc., 2023).
27. Nøkland, A. & Eidnes, L. H. Training neural networks with local error signals. In *Proc. International conference on machine learning* 4839–4850 (PMLR, 2019).
28. He, K., Zhang, X., Ren, S. & Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*. 630–645 (Springer, 2016).
29. Krizhevsky, A. Learning multiple layers of features from tiny images. (University of Toronto, 2012).
30. Ledig, C. et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition* 4681–4690 (IEEE, 2017).
31. Russakovsky, O. et al. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.* **115**, 211–252 (2015).
32. Liu, Z. et al. Swin transformer: hierarchical vision transformer using shifted windows. In *Proc. IEEE/CVF International Conference on Computer Vision* 10012–10022 (IEEE, 2021).
33. Wang, Z. et al. In situ training of feed-forward and recurrent convolutional memristor networks. *Nat. Mach. Intell.* **1**, 434–442 (2019).
34. Wang, Z. et al. Reinforcement learning with analogue memristor arrays. *Nat. Electron.* **2**, 115–124 (2019).
35. Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proc. 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* 4171–4186. https://doi.org/10.18653/v1/N19-1423 (Association for Computational Linguistics, 2019).
36. Liu, Z. et al. A ConvNet for the 2020s. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* 11976–11986 (IEEE, 2022).
37. Rombach, R., Blattmann, A., Lorenz, D., Esser, P. & Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* 10684–10695 (IEEE, 2022).
38. Song, W. et al. Programming memristor arrays with arbitrarily high precision for analog computing. *Science* **383**, 903–910 (2024).
39. Sharma, D. et al. Linear symmetric self-selecting 14-bit kinetic molecular memristors. *Nature* **633**, 560–566 (2024).
40. Cai, F. et al. A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations. *Nat. Electron.* **2**, 290–299 (2019).

41. Bernstein, J., Wang, Y.-X., Azizzadenesheli, K. & Anandkumar, A. signSGD: Compressed optimisation for non-convex problems. In *Proc. International Conference on Machine Learning* 560–569 (PMLR, 2018).
42. Wen, W. et al. TernGrad: ternary gradients to reduce communication in distributed deep learning. In *Proc. of the 31st International Conference on Neural Information Processing Systems.* Vol. 30 1508–1518 (Curran Associates Inc., 2017).
43. Kingma, D. P. & Ba, J. Adam: a method for stochastic optimization. Preprint at *arXiv* https://doi.org/10.48550/arXiv.1412.6980 (2014).
44. Jiang, H. et al. A novel true random number generator based on a stochastic diffusive memristor. *Nat. Commun.* **8**, 882 (2017).
45. Kim, G. et al. Self-clocking fast and variation tolerant true random number generator based on a stochastic mott memristor. *Nat. Commun.* **12**, 2906 (2021).
46. Shi, T. et al. Memristor-based feature learning for pattern classification. *Nat. Commun.* **16**, https://doi.org/10.1038/s41467-025-56286-y (2025).
47. Michelucci, U. An introduction to autoencoders. Preprint at *arXiv* https://doi.org/10.48550/arXiv.2201.03898 (2022).
48. Choi, C. et al. Reconfigurable heterogeneous integration using stackable chips with embedded artificial intelligence. *Nat. Electron.* **5**, 386–393 (2022).
49. Deng, L. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Process. Mag.* **29**, 141–142 (2012).
50. Netzer, Y. et al. Reading digits in natural images with unsupervised feature learning. In *Proc. NIPS workshop on deep learning and unsupervised feature learning* Vol. 7 (Granada, 2011).
51. Wang, X., Xie, L., Dong, C. & Shan, Y. Real-ESRGAN: training real-world blind super-resolution with pure synthetic data. In *Proc. IEEE/CVF International Conference on Computer Vision* 1905–1914 (IEEE, 2021).
52. Shi, W. et al. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition* 1874–1883 (IEEE, 2016).
53. Rasch, M. J. et al. A flexible and fast PyTorch toolkit for simulating training and inference on analog crossbar arrays. In *Proc. 2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)* 1–4 (IEEE, 2021).
54. Le Gallo, M. et al. Using the IBM analog in-memory hardware acceleration kit for neural network training and inference. *APL Mach. Learn.* **1**, https://doi.org/10.1063/5.0168089 (2023).
55. Gokmen, T. Enabling training of neural networks on noisy hardware. *Front. Artif. Intell.* **4**, 699148 (2021).
56. Rasch, M. J., Carta, F., Fagbohungbe, O. & Gokmen, T. Fast and robust analog in-memory deep neural network training. *Nat. Commun.* **15**, 7133 (2024).
57. Yao, P. et al. Face classification using electronic synapses. *Nat. Commun.* **8**, 15199 (2017).
58. Liu, J. et al. Error-aware probabilistic training for memristive neural networks. EaPU, https://doi.org/10.5281/zenodo.17338135 (2025).

Acknowledgements
This work was supported by the National Key R&D Program of China under Grant No. 2021YFB3601200, the National Natural Science Foundation of China under Grant Nos. U20A20220, U22A6001, 61821091, 61888102, and 61825404, the Key R&D Program of Zhejiang (No. 2022C01048), the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant XDB44000000.

Author contributions
T.S. and J.C.L. conceived the concept and designed the experiments. J.C.L. S.T., R.Z., and H.M. performed the electrical measurements. J.C.L. contributed to the neural network simulation. J.C.L., J.L., T.S., and Q.L. analyzed the experimental data, and J.C.L., J.L., B.L., Y.T., T.S., and Q.L. wrote the manuscript. All authors discussed the results and commented on the manuscript at all stages. Q.L. and T.S. supervised the research.

Competing interests
The authors declare no competing interests.

Additional information
Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s41467-025-66240-7.

Correspondence and requests for materials should be addressed to Tuo Shi or Qi Liu.

Peer review information *Nature Communications* thanks Corey Lammie and the other, anonymous, reviewer for their contribution to the peer review of this work. A peer review file is available.

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

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

Open Access This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by-nc-nd/4.0/.

© The Author(s) 2025