Supplementary Information for Reconfigurable Digital RRAM Logic Enables In-situ Pruning and Learning for Edge AI

Songqi Wang[1,2,3†], Yue Zhang[1,2†], Jia Chen[4,5], Xinyuan Zhang[1,3], Yi Li[1], Ning Lin[1], Yangu He[1], Jichang Yang[1,3], Yingjie Yu[5], Yi Li[5], Zhongrui Wang[1,2,4*], Xiaojuan Qi[1*], and Han Wang[1,3*]

[1] *Department of Electrical and Electronic Engineering, the University of Hong Kong, Hong Kong, China*

[2] *School of Microelectronics, Southern University of Science and Technology, Shenzhen, China.*

[3] *Center for Advanced Semiconductor and Integrated Circuit, The University of Hong Kong, Hong Kong, China*

[4] *ACCESS – AI Chip Center for Emerging Smart Systems, InnoHK Centers, Hong Kong Science Park, Hong Kong, China*

[5] *School of Optical and Electronic Information Huazhong University of Science and Technology, China*

June 14, 2025

S1. Supplementary Figures

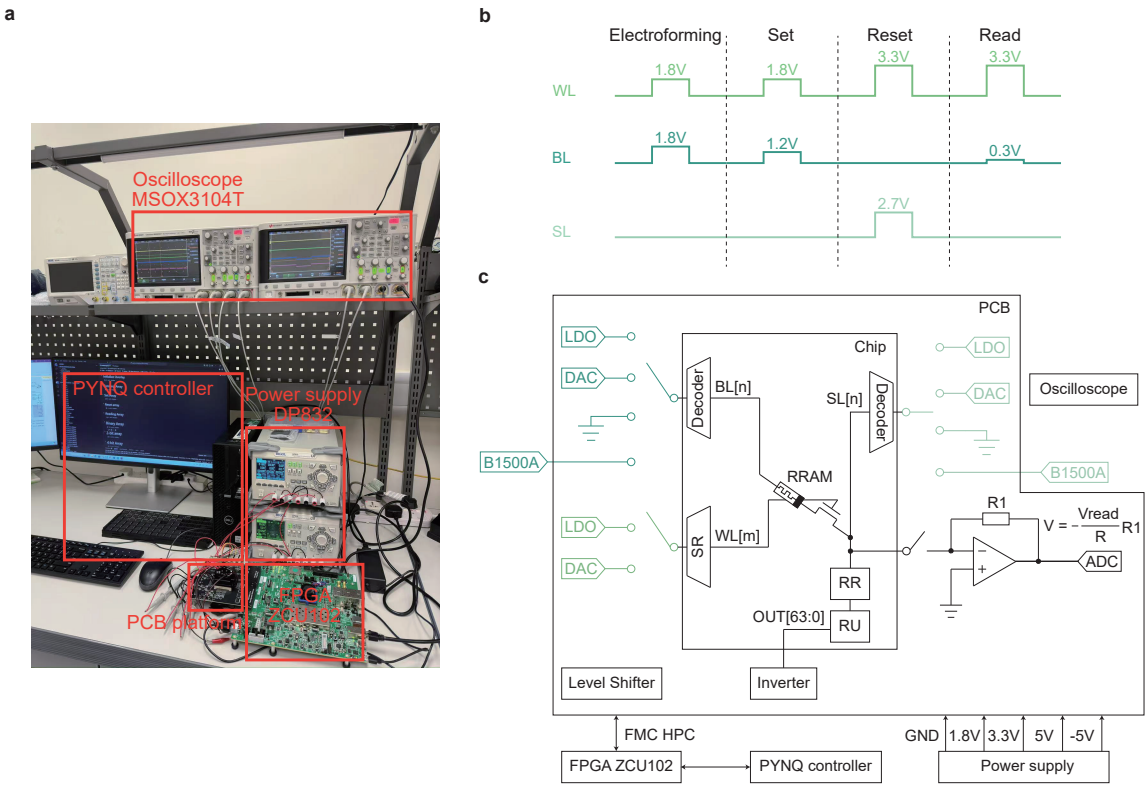

Figure 1: **Experimental setup and RRAM operation scheme. a,** Photograph of the experimental setup, including two oscilloscopes (MSOX3104T) for signal monitoring, a PYNQ controller for digital control, an FPGA (ZCU102) for interfacing with a custom PCB platform, and two power supplies (DP832) for voltage regulation. **b,** Voltage waveform sequence for RRAM electroforming, set, reset, and read operations. The applied voltages on the word line (WL), bit line (BL), and source line (SL) are specified for each phase. **c,** Schematic of the programming system. The PCB integrates DACs, switches, and a low dropout regulator (LDO) to support different operational modes. During neural network training and inference, only the LDO and switches are required to program the RRAM resistance state, significantly reducing power consumption and hardware complexity. The resistance state is read through the Rref Read and Reconfigurable Unit modules integrated within the chip, enabling efficient and low-overhead resistance state retrieval. For experimental studies of RRAM array performance, programming and characterization can be performed using on-board DACs and switches for fast and coarse tuning. During read operations, a voltage is applied to the BL, and the resulting current is converted into a voltage via a transimpedance amplifier (TIA) before being digitized by the ADC to estimate the RRAM resistance state. For precise programming and high-accuracy resistance measurement, an external B1500A unit provides fine-tuned voltage control.

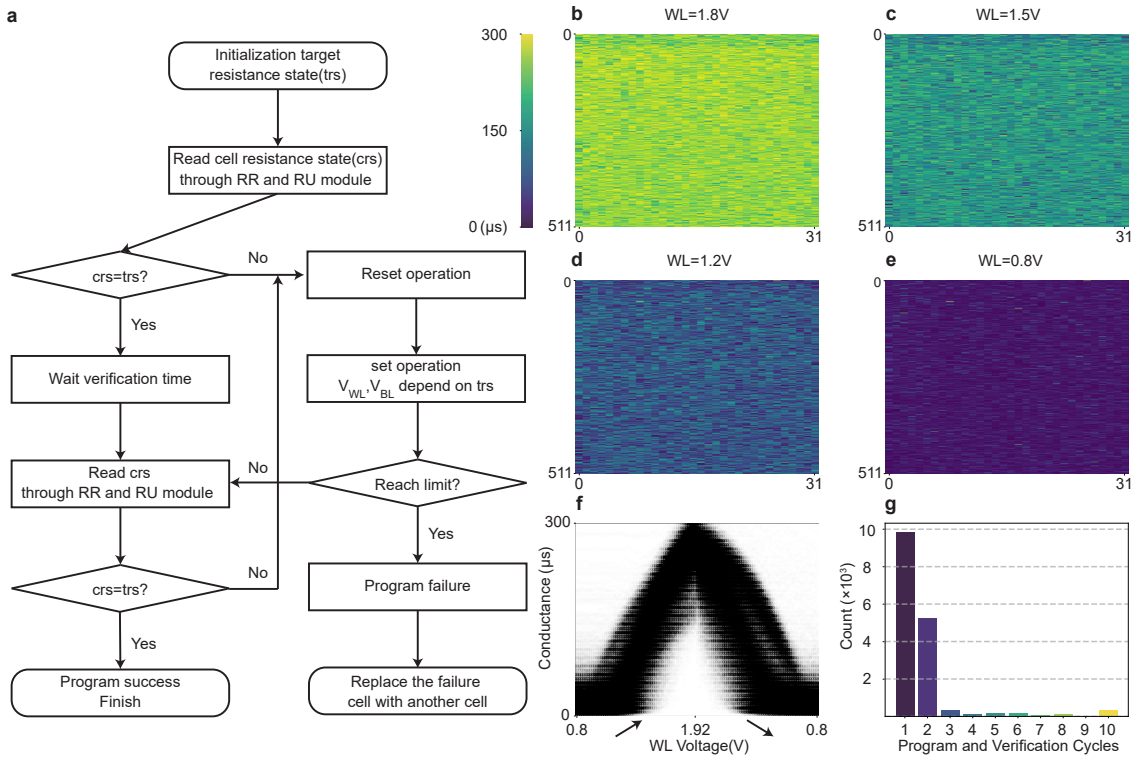

Figure 2: **RRAM programming flowchart and characteristics.** **a**, Programming and verification flowchart for tuning each RRAM cell to a target resistance state (trs). The process begins by reading the current resistance state (crs) through the Read Rref (RR) module and the Reconfigurable Unit (RU) module and comparing it with the trs. If the states do not match, a reset or set operation is applied depending on the deviation. This loop of read and verification continues until the target state is achieved or the maximum allowed number of cycles is reached, in which case the cell is marked as a failure and replaced. **b**–**e**, Conductance mapping of the 512 × 32 RRAM array under four different word-line (WL) voltages during programming to the INT2 state: **b**, 1.8 V; **c**, 1.5 V; **d**, 1.2 V; **e**, 0.8 V. **f**, Global conductance distribution of the array as a function of WL voltage. The WL voltage is swept from 0.8 V to 1.92 V in 0.02 V increments and then decreased back to 0.8 V with the same step size. **g**, Histogram showing the number of program-and-verify cycles required for all cells in the array to reach the target resistance state. The majority of cells reach the target within 1–2 cycles.

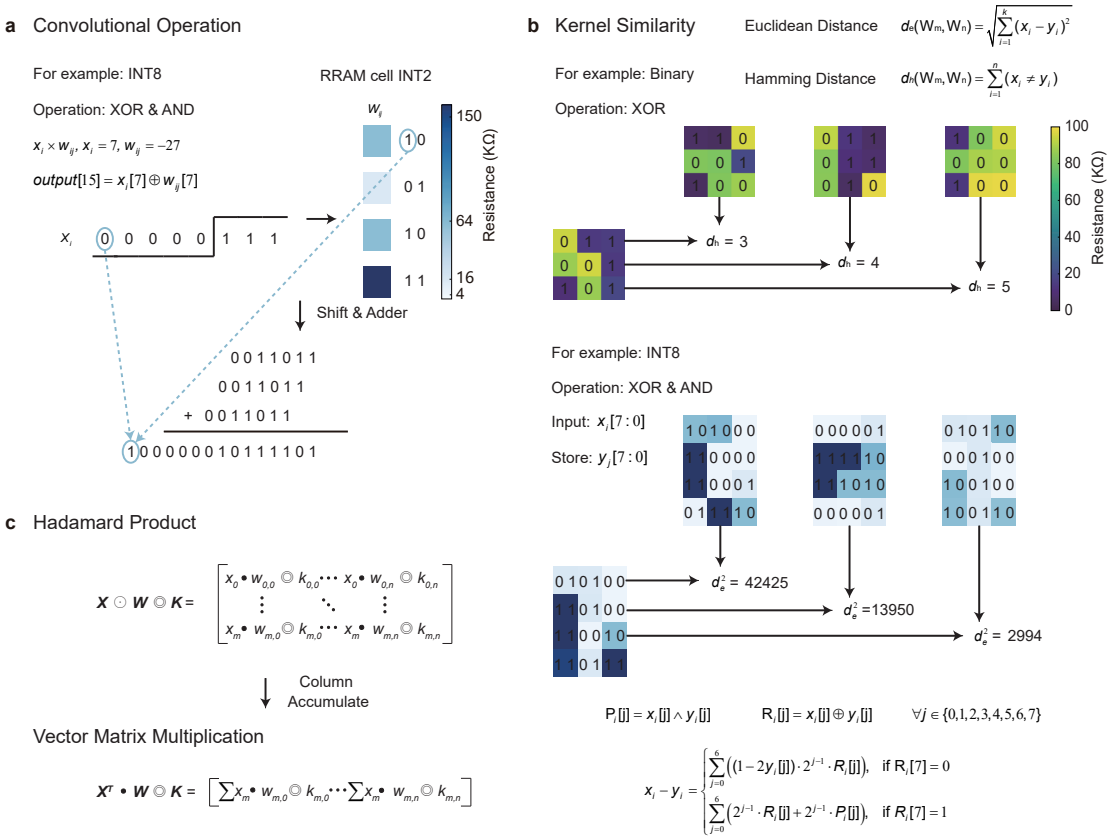

Figure 3: **RRAM-based convolution operation and weight similarity.** **a,** Convolution operation using signed 8-bit integer (INT8) inputs and weights. Each 2-bit segment of the weight is stored in a single RRAM cell, with resistance values representing the encoded bits. Computation is performed using bitwise XOR and AND operations. For example, given $x_i = 7$ (00000111) and $w_{ij} = -27$ (10011011), the output sign bit is computed as output$[15] = x_i[7] \oplus w_{ij}[7]$. The partial products are accumulated through a shift-and-adder group to generate the final output. **b,** Weight similarity evaluation based on either Hamming or Euclidean distance. For binary weights, the Hamming distance d_h, computed via XOR, can approximate the Euclidean distance d_e. For INT8 weights, XOR and AND operations are applied bitwise between two weight vectors x_i and y_i to compute $x_i - y_i$. The squared Euclidean distance d_e^2 is then derived from the bitwise logic values $P_j = x_j \wedge y_j$ and $R_j = x_j \oplus y_j$ using the provided formula. **c,** Ternary computation based on the Hadamard product and accumulation. The element-wise operation $X \odot W \circledcirc K$ is computed, followed by column-wise accumulation to yield the result of ternary vector-matrix multiplication $X^\top \cdot W \circledcirc K$, where $\circledcirc$ represents logical operations such as NAND, AND, XOR, or OR.

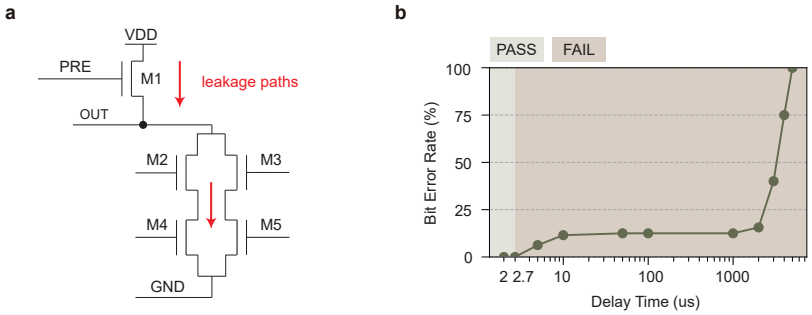

Figure 4: **Impact of leakage on compute accuracy. a,** Schematic illustration of leakage paths [1] in the precharge-compute structure. During the precharge phase, the signal line OUT is pulled up to 1.8V through transistor M1. However, during the subsequent compute phase, if the operation time is prolonged, charge on OUT can dissipate through leakage paths formed by transistors M2-M5, leading to a voltage drop and resulting in incorrect computation outcomes. **b,** Measured relationship between compute delay time and bit error rate (BER). When the delay exceeds a critical threshold ($\sim$2.7 μs), leakage-induced charge loss leads to a rapid increase in BER.

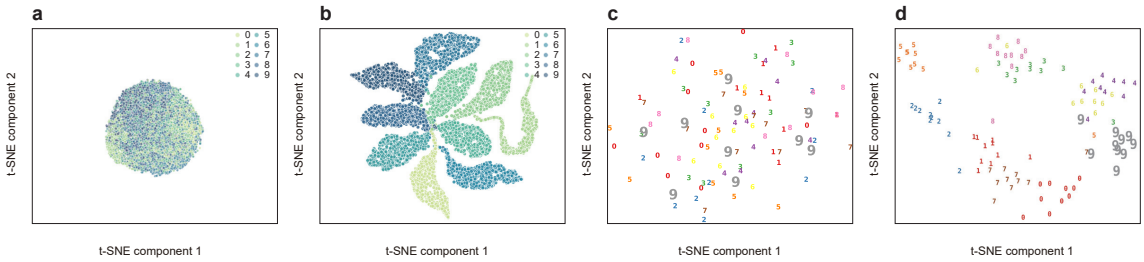

Figure 5: **t-SNE visualization of unpruned networks. a,** Feature distribution before training in a CNN-based network, showing high inter-class overlap. **b,** Feature distribution after training in the same CNN, with improved clustering and separability. **c,** Feature distribution before training in a PointNet++ network for point cloud classification. **d,** Feature distribution after training in PointNet++, exhibiting clear inter-class separation.

S2. Supplementary Tables

Table 1: Area and energy consumption breakdown by module.

	WRC	**BSIC**	**RRAM**	**RR**	**RU**	**S & A**	**ACC**
Area (mm^2)	0.106	0.028	0.534	0.005	0.001	0.037	0.155
Energy (pJ)	2690.400	68.682	0.576	34.001	22.056	269.12	907.136

Abbreviations: WRC – WL Driver & RU Controller; BSIC – BL/SL Driver Circuits & Input Controller; RRAM – Resistive Random-Access Memory Array; RR – Rref Read module; RU – Reconfigurable Unit; S & A – Shift and Adder Group; ACC – Accumulator.

Note. The reported energy consumption values correspond to the operation involving parallel 64-bit AND computations across the system, including the subsequent shift and accumulation processes required for final result generation.

Table 2: Architecture of the VGG16-based model (Task 1) and PointNet++ model (Task 2).

Layer	**VGG16-based model**	**PointNet++ model**
1	`BinaryConv2d(1, 32, kernel_size=3, padding=1)` → `ReLU` → `MaxPool2d(2)`	`SA1: in_channels=`C_{in}`, mlp=[64, 64, 128], group_all=False`
2	`BinaryConv2d(32, 64, kernel_size=3, padding=1)` → `ReLU` → `MaxPool2d(2)`	`SA2: in_channels=131, mlp=[128, 128, 256], group_all=False`
3	`BinaryConv2d(64, 32, kernel_size=3, padding=1)` → `ReLU`	`SA3: in_channels=259, mlp=[256, 512, 1024], group_all=True`
4	`Flatten to vector of size 32` × 7 × 7	`Flatten to vector of size 1024`
5	`Linear(1568, 10) for classification`	`Linear(1024, 512, bias=False)` → `BN(512)` → `ReLU` → `Dropout(0.5)`
6	–	`Linear(512, 256, bias=False)` → `BN(256)` → `ReLU` → `Dropout(0.5)`
7	–	`Linear(256, num_classes) for classification`

Note. The complete implementation of both models is available at:
`https://github.com/wangsongq/Dynamic-Kernel-Pruning.git`

S3. Supplementary Notes

Note1. Computational Cost Estimation

The term "Ops." refers to the number of multiply-accumulate operations (MACs) required during a single forward inference. For different layer types, the computations are estimated as follows:

Convolutional Layer

$$\text{Ops.} = 2 \times C_{\text{in}} \times C_{\text{out}} \times k_h \times k_w \times H_{\text{out}} \times W_{\text{out}}, \tag{1}$$

where C_{in} and C_{out} denote the number of input and output channels, k_h and k_w denote the kernel height and width, and $H_{\text{out}}, W_{\text{out}}$ denote the output feature map dimensions. The factor 2 accounts for both multiplication and accumulation.

Fully Connected Layer

$$\text{Ops.} = 2 \times w_h \times w_w, \tag{2}$$

where w_h and w_w denote the height and width of the weight matrix in the linear layer.

NVIDIA GeForce RTX 4090

To establish a quantitative energy-efficiency baseline for conventional digital accelerators, we estimate the per-operation and per-bit energy consumption of the NVIDIA GeForce RTX 4090 when executing INT8 operations. According to publicly available data from LLM Tracker and TechPowerUp, the peak INT8 tensor compute performance of the RTX 4090 reaches 660.6 TOPS (dense), with a power draw of 450 W under full load. The resulting energy cost per INT8 operation is given by:

$$\text{Energy per Operation} = \frac{\text{Power Draw}}{\text{Peak INT8 Performance}} = \frac{450\ \text{W}}{660.6 \times 10^{12}\ \text{Ops/s}} = 0.6812\ \text{pJ/op} \tag{3}$$

Our system

To evaluate the energy efficiency of our proposed system in a realistic serial multiplication architecture, we model the full execution of a multiplication and accumulation across 32 parallel column units.

The overall power consumption of the system, accounting for WL/SR driver (134.52 mW), bit-line decoder (3.4341 mW), RRAM array (0.0288 mW), read units (1.6992 mW), reconfigurable units (1.104 mW), shift-and-adder units (13.456 mW), and accumulators (45.3568 mW), amounts to 199.60 mW. The average energy consumption per operation is thus:

$$\text{Energy per operation} = \frac{199.60\ \text{mW} \times 22.5\ \text{ns}}{64} = 70.17\ \text{pJ/OP} \tag{4}$$

To provide a normalized comparison with modern digital platforms, we scale the energy consumption based on voltage and frequency using standard dynamic power scaling laws. Specifically, assuming a 0.8 V supply and a clock frequency of 1.8 GHz (typical for modern GPUs), we apply:

$$\text{Scaled Energy} = 70.17\ \text{pJ} \cdot \left(\frac{0.8}{3.3}\right)^2 \cdot \left(\frac{100\ \text{MHz}}{1.8\ \text{GHz}}\right) \approx 0.229\ \text{pJ/OP} = 0.229\ \text{pJ/OP} \tag{5}$$

Note that this estimation excludes process technology scaling (e.g., from 180 nm to 7 nm) and is thus conservative.

References

[1] Bonan Yan, Jeng-Long Hsu, Pang-Cheng Yu, Chia-Chi Lee, Yaojun Zhang, Wenshuo Yue, Guoqiang Mei, Yuchao Yang, Yue Yang, Hai Li, et al. A 1.041-mb/mm 2 27.38-tops/w signed-int8 dynamic-logic-based adc-less sram compute-in-memory macro in 28nm with reconfigurable bitwise operation for ai and embedded applications. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 188–190. IEEE, 2022.