

# Supplementary Information

## Efficient Nonlinear Function Approximation in Analog Resistive Crossbars for Recurrent Neural Networks

Junyi Yang<sup>1,\*</sup>, Ruibin Mao<sup>2,\*</sup>, Mingrui Jiang<sup>2</sup>, Yichuan Cheng<sup>1</sup>, Pao-Sheng Vincent Sun<sup>1</sup>, Shuai Dong<sup>1</sup>, Giacomo Pedretti<sup>3</sup>, Xia Sheng<sup>3</sup>, Jim Ignowski<sup>3</sup>, Haoliang Li<sup>1</sup>, Can Li<sup>2,+</sup>, and Arindam Basu<sup>1,+</sup>

<sup>1</sup>Department of Electrical Engineering, City University of Hong Kong, Hong Kong SAR, China.

<sup>2</sup>Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong SAR, China.

<sup>3</sup>Hewlett Packard Labs, Hewlett Packard Enterprise, Milpitas, CA, USA.

\* Equal Contribution

+ Correspondence to arinbasu@cityu.edu.hk and canli@hku.hk

### Contents

Supplementary Note S1. Nonlinear function Approximation by ramp ADC

Supplementary Note S2. Spice simulation

Supplementary Note S3. Energy, Area and Latency estimation at Macro level

Supplementary Note S4. Energy, Area and Latency estimation at system level

Supplementary Note S5. Programming of NL-ADC on crossbar arrays

Supplementary Note S6. Latency estimation of MAC operation in In-memory computing and Digital

Non-linear function implementations

Supplementary Note S7. Circuit implementation

Supplementary Note S8. Analysis of Accuracy Drop with Write Noise

Supplementary Note S9. One-point calibration process

Supplementary Note S10. Capacitor-based accumulation method for high-dimensional inputs

Supplementary Note S11. Redundancy for improved NL-ADC programming

Supplementary Note S12. Non-monotonic nonlinear function approximation by ramp ADC

Supplementary Note S13. Effect of long-term drift of the RRAM conductances

Supplementary Note S14. Memristor programming circuits and overhead

## Supplementary Note S1. Nonlinear function approximation by ramp ADC

Six commonly used nonlinear functions in neural networks and their inverse functions are presented in Tab. S1. To illustrate how the nonlinear functions are approximated, we consider the utilization of a 5-bit NL-ADC and sample 33 points in the inverse function of each nonlinear function. The curves of  $(t_k, V_k)$  tuples in Equation (3) Results for six inverse functions are presented in Fig. S1. Following the acquisition of 33 sampling points  $((t_k, V_k))$ , the calculation of  $\Delta V_k$  can be performed using:

$$\Delta V_k = V_k - V_{k-1} \quad k \in [1, 32] \quad (S1)$$

The advantage associated with the tunability of memristor conductance allows for the encoding of each  $\Delta V_k$  using a single memristor. However, in the case of the SRAM model, each cell possesses the capacity to store only two levels. Consequently, representing a  $\Delta V_k$  within the SRAM framework may require the utilization of multiple cells. The determination of the number of SRAM cells required per  $\Delta V_k$  can be computed based on the provided formula ( $\text{round}\left(\frac{\Delta V_k}{\min(\Delta V_k)}\right)$ ). The values of  $\Delta V_k$ , along with the corresponding number of SRAM cells required to store  $\Delta V_k$  for six functions, are presented in Tab. S2. By comparing the number of SRAM cells and the number of memristor required for each  $\Delta V_k$  (each  $\Delta V_k$  only requires one memristor) and considering the inherent smaller bitcell size of memristors in comparison to SRAM, our utilization of memristor for nonlinear function approximation demonstrates a distinct advantage in terms of overhead.

**Tab. S1.** Six commonly used nonlinear functions in neural networks and their inverse functions.

| Name     | Nonlinear function                                                                       | Inverse function                                                                              |
|----------|------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| Sigmoid  | $f(x) = \frac{1}{1+e^{-x}}$                                                              | $f^{-1}(t) = \ln \frac{t}{1-t}$                                                               |
| Softplus | $f(x) = \ln(1 + e^x)$                                                                    | $f^{-1}(t) = \ln(e^t - 1)$                                                                    |
| Tanh     | $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$                                               | $f^{-1}(t) = 0.5 \ln \frac{1+t}{1-t}$                                                         |
| Softsign | $f(x) = \begin{cases} \frac{x}{1+x} & (x \geq 0) \\ \frac{x}{1-x} & (x < 0) \end{cases}$ | $f^{-1}(t) = \begin{cases} \frac{t}{1-t} & (t \geq 0) \\ \frac{t}{1+t} & (t < 0) \end{cases}$ |
| Elu      | $f(x) = \begin{cases} x & (x \geq 0) \\ e^x - 1 & (x < 0) \end{cases}$                   | $f^{-1}(t) = \begin{cases} t & (t \geq 0) \\ \ln(t + 1) & (t < 0) \end{cases}$                |
| Selu     | $f(x) = \begin{cases} 0.5x & (x \geq 0) \\ 2(e^x - 1) & (x < 0) \end{cases}$             | $f^{-1}(t) = \begin{cases} 2t & (t \geq 0) \\ \ln(0.5t + 1) & (t < 0) \end{cases}$            |

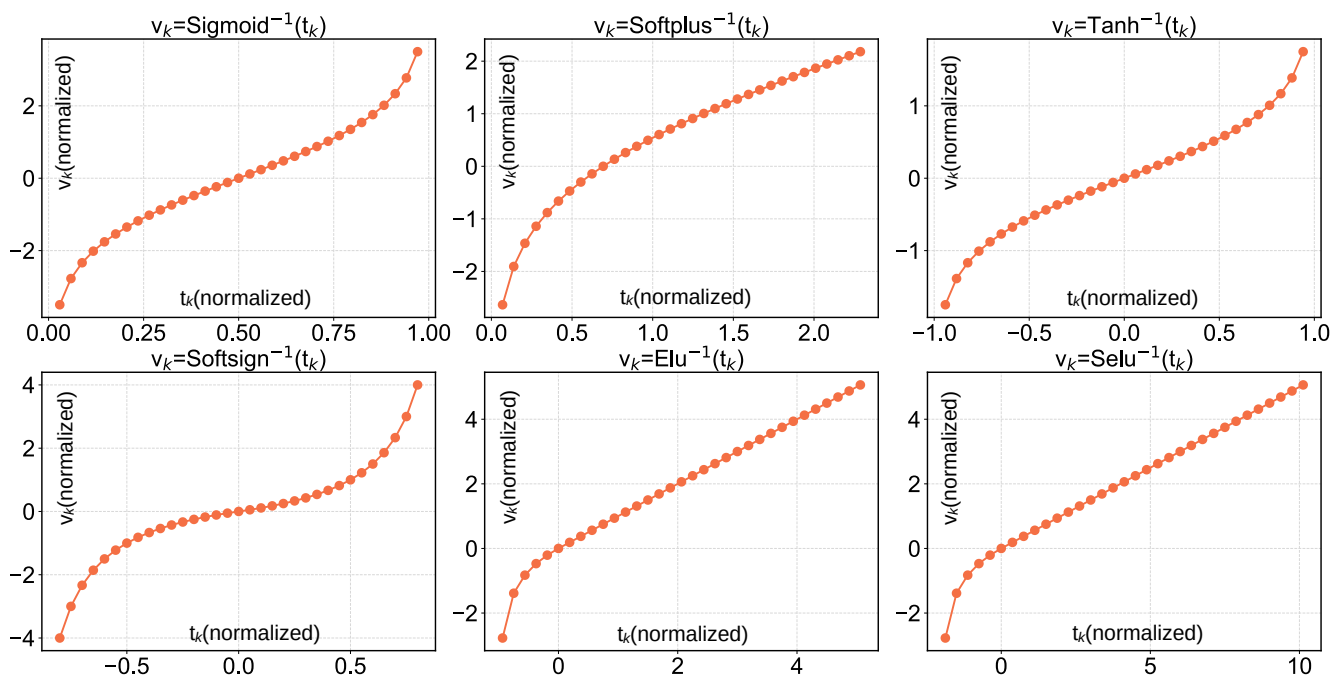


**Fig. S1. Plot of the six inverse functions.**

**Tab. S2.**  $\Delta V_k$  and SRAM cell numbers for six inverse functions.

| k          | Sigmoid      |            | Softplus     |            | Tanh         |            | Softsign     |            | Elu          |            | Selu         |            |
|------------|--------------|------------|--------------|------------|--------------|------------|--------------|------------|--------------|------------|--------------|------------|
|            | $\Delta V_k$ | # SR. cell | $\Delta V_k$ | # SR. cell | $\Delta V_k$ | # SR. cell | $\Delta V_k$ | # SR. cell | $\Delta V_k$ | # SR. cell | $\Delta V_k$ | # SR. cell |
| 1          | 0.724        | 6          | 0.728        | 9          | 0.362        | 6          | 1            | 19         | 1.386        | 7          | 1.386        | 7          |
| 2          | 0.437        | 4          | 0.441        | 6          | 0.219        | 4          | 0.667        | 13         | 0.56         | 3          | 0.56         | 3          |
| 3          | 0.32         | 3          | 0.324        | 4          | 0.16         | 3          | 0.476        | 9          | 0.357        | 2          | 0.357        | 2          |
| 4          | 0.257        | 2          | 0.26         | 3          | 0.129        | 2          | 0.357        | 7          | 0.262        | 1          | 0.262        | 1          |
| 5          | 0.217        | 2          | 0.219        | 3          | 0.109        | 2          | 0.278        | 5          | 0.208        | 1          | 0.208        | 1          |
| 6          | 0.191        | 2          | 0.191        | 2          | 0.095        | 2          | 0.222        | 4          | 0.188        | 1          | 0.188        | 1          |
| 7          | 0.171        | 1          | 0.171        | 2          | 0.086        | 1          | 0.182        | 3          | 0.188        | 1          | 0.188        | 1          |
| 8          | 0.157        | 1          | 0.156        | 2          | 0.079        | 1          | 0.152        | 3          | 0.188        | 1          | 0.188        | 1          |
| 9          | 0.146        | 1          | 0.144        | 2          | 0.073        | 1          | 0.128        | 2          | 0.188        | 1          | 0.188        | 1          |
| 10         | 0.138        | 1          | 0.134        | 2          | 0.069        | 1          | 0.11         | 2          | 0.188        | 1          | 0.188        | 1          |
| 11         | 0.131        | 1          | 0.126        | 2          | 0.066        | 1          | 0.095        | 2          | 0.188        | 1          | 0.188        | 1          |
| 12         | 0.127        | 1          | 0.12         | 2          | 0.063        | 1          | 0.083        | 2          | 0.188        | 1          | 0.188        | 1          |
| 13         | 0.123        | 1          | 0.114        | 1          | 0.061        | 1          | 0.074        | 1          | 0.188        | 1          | 0.188        | 1          |
| 14         | 0.12         | 1          | 0.109        | 1          | 0.06         | 1          | 0.065        | 1          | 0.188        | 1          | 0.188        | 1          |
| 15         | 0.119        | 1          | 0.105        | 1          | 0.059        | 1          | 0.058        | 1          | 0.188        | 1          | 0.188        | 1          |
| 16         | 0.118        | 1          | 0.102        | 1          | 0.059        | 1          | 0.053        | 1          | 0.188        | 1          | 0.188        | 1          |
| 17         | 0.118        | 1          | 0.099        | 1          | 0.059        | 1          | 0.053        | 1          | 0.188        | 1          | 0.188        | 1          |
| 18         | 0.119        | 1          | 0.096        | 1          | 0.059        | 1          | 0.058        | 1          | 0.188        | 1          | 0.188        | 1          |
| 19         | 0.12         | 1          | 0.094        | 1          | 0.06         | 1          | 0.065        | 1          | 0.188        | 1          | 0.188        | 1          |
| 20         | 0.123        | 1          | 0.091        | 1          | 0.061        | 1          | 0.074        | 1          | 0.188        | 1          | 0.188        | 1          |
| 21         | 0.127        | 1          | 0.089        | 1          | 0.063        | 1          | 0.083        | 2          | 0.188        | 1          | 0.188        | 1          |
| 22         | 0.131        | 1          | 0.088        | 1          | 0.066        | 1          | 0.095        | 2          | 0.188        | 1          | 0.188        | 1          |
| 23         | 0.138        | 1          | 0.086        | 1          | 0.069        | 1          | 0.11         | 2          | 0.188        | 1          | 0.188        | 1          |
| 24         | 0.146        | 1          | 0.085        | 1          | 0.073        | 1          | 0.128        | 2          | 0.188        | 1          | 0.188        | 1          |
| 25         | 0.157        | 1          | 0.084        | 1          | 0.079        | 1          | 0.152        | 3          | 0.188        | 1          | 0.188        | 1          |
| 26         | 0.171        | 1          | 0.082        | 1          | 0.086        | 1          | 0.182        | 3          | 0.188        | 1          | 0.188        | 1          |
| 27         | 0.191        | 2          | 0.081        | 1          | 0.095        | 2          | 0.222        | 4          | 0.188        | 1          | 0.188        | 1          |
| 28         | 0.217        | 2          | 0.08         | 1          | 0.109        | 2          | 0.278        | 5          | 0.188        | 1          | 0.188        | 1          |
| 29         | 0.257        | 2          | 0.08         | 1          | 0.129        | 2          | 0.357        | 7          | 0.188        | 1          | 0.188        | 1          |
| 30         | 0.32         | 3          | 0.079        | 1          | 0.16         | 3          | 0.476        | 9          | 0.188        | 1          | 0.188        | 1          |
| 31         | 0.437        | 4          | 0.078        | 1          | 0.219        | 4          | 0.667        | 13         | 0.188        | 1          | 0.188        | 1          |
| 32         | 0.724        | 6          | 0.077        | 1          | 0.362        | 6          | 1            | 19         | 0.188        | 1          | 0.188        | 1          |
| <b>Sum</b> | <b>6.992</b> | <b>58</b>  | <b>4.813</b> | <b>59</b>  | <b>3.498</b> | <b>58</b>  | <b>8.0</b>   | <b>150</b> | <b>7.849</b> | <b>41</b>  | <b>7.849</b> | <b>41</b>  |

**Supplementary Note S2. Spice simulation**

In order to evaluate the effect of circuit non-idealities in the charge integrator, we performed spice simulations (Fig. S2) using a TSMC 65 nm CMOS process models. Finite gain and bandwidth of the integrator stage were varied to find acceptable values where the error in integration is  $< 1\%$ . A finite DC gain of  $\approx 1000$  and gain bandwidth product of  $\approx 200$  MHz was found to be sufficient in this case. A sigmoidal nonlinear activation was considered here. We also performed more SPICE simulations with more realistic memristor models to make sure our energy estimation based on first order model of a memristor as a resistor are sufficiently accurate. First, we estimate the capacitance of the Ta/TaOx/Pt vertically stacked memristor device based on the parallel plate capacitor model. The lateral dimension of the device is  $50 \text{ nm} \times 50 \text{ nm}$ . The thickness of the switching layer between two metal electrodes is  $8 \text{ nm}$ . The dielectric constant of the TaOx is assumed to be 30 and the overall estimated capacitance of the devices is  $0.083 \text{ fF}$ . The SPICE simulation was redone using a more conservative estimate of  $C_p = 1 \text{ fF}$ , resulting in  $\approx 0.18\%$  increase in energy.

Further, we also chose the popularly used detailed HSPICE model of memristors<sup>1</sup> and redid the simulation. The HSPICE model simulation was redone using the more conservative estimate of  $C_p = 1 \text{ fF}$ , resulting in  $0.19\%$  increase in energy.

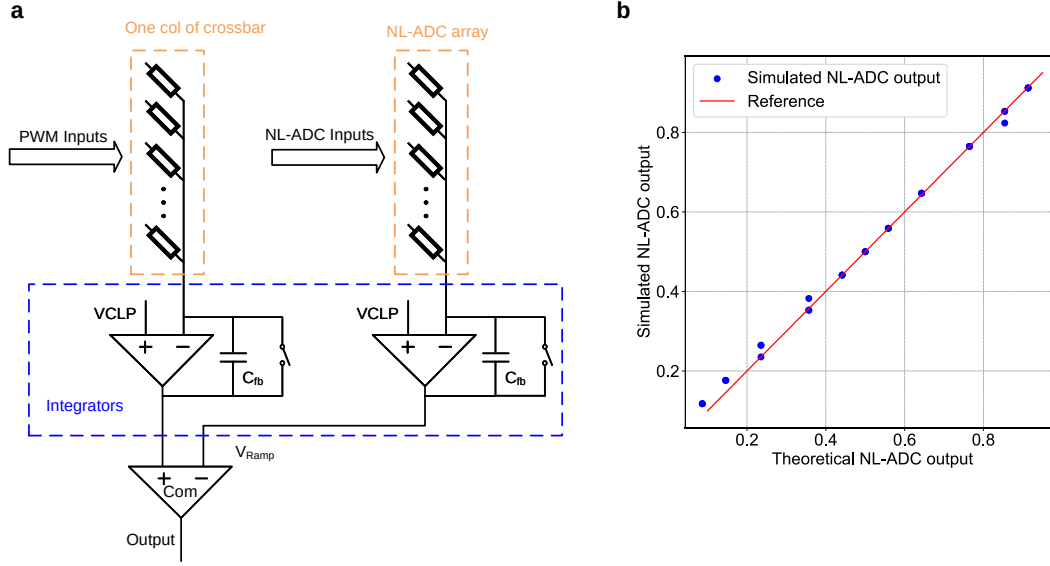


**Fig. S2. Spice simulation.** **a** Architecture of circuit simulation. **b** simulation result. In our simulation setup, we utilize a column of Resistive Random Access Memory (RRAM) to simulate the Multiply-Accumulate (MAC) operation. Additionally, we employ 33 RRAMs to simulate the Nonlinear Analog-to-Digital Converter (NL-ADC), with one RRAM dedicated to NL-ADC compensation. The integrator in our system has a Gain-Bandwidth Product (GBW) of 200MHz and a direct current (DC) gain of 1000. Throughout our simulations, we analyze a total of 30 cases, where the theoretical output is  $\text{sigmoid}^{-1}(V_{MAC})$ .

### Supplementary Note S3. Energy, Area and Latency estimation at Macro level

We estimate our work from two levels, Macro level and full system level. The Macro level includes most of the LSTM layer operations and full system level consists of all LSTM layer operations and FC layer operations.

In order to have a fair comparison between our proposed method combining ADC and non-linear function computation in IMC with the traditional method of IMC for the MAC operation followed by a conventional ramp ADC and digital processor, we created two models as shown in Fig. S3 where other parameters such as input and output bit-width, clock frequency etc. were kept the same. These two models consist of crossbar arrays, integrators, sample and hold (S&H) blocks, comparators, conventional ramp ADCs, ripple counters and processors. We acquired data pertaining to crossbar arrays, drivers and S&H from reference<sup>2</sup>. The data regarding integrators and comparators were obtained from reference<sup>3</sup> and reference<sup>4</sup> respectively. In the case of the conventional ramp ADC, we get the data from reference<sup>5</sup>. Ripple counter efficiently converts the thermometer code output from comparators into binary code, thus simplifying subsequent processing by the digital domain processor. The energy consumption and area information of the ripple counter are determined through Spice simulation and reference paper<sup>6</sup> respectively. As for processors, we get the data from reference<sup>6</sup>. We consider a programming 8-bit ADC for every sub-array of size 512x512 for RRAM writing. This ADC is only used to program RRAM and it is inactive during the inference process. Therefore, we only need to pay attention to its area information and do not need to consider its energy when we calculate efficiency for inference. The area of ADC is 280  $\mu\text{m}^2$  from reference<sup>7</sup>. We followed the methods in recent papers<sup>2,3</sup> to estimate the energy, latency and area of different sub-circuits in the whole system. For fair comparison, all of the numbers are scaled to the same process node of 16 nm and frequency of 1 GHz.

We train two models, and evaluate the two models separately. In Keyword Spotting (KWS) model, the LSTM parameter size ( $72 \times 128$ ) is small enough such that it can fit in one crossbar of size  $128 \times 128$ . But for Natural language processing (NLP) model, the LSTM parameter size ( $633 \times 8064$ ) is too big to fit in one crossbar. We assume a crossbar size of  $633 \times 512$  similar to recent work<sup>8</sup> and use 16 such crossbars. The number of rows is kept at 633 to ensure that it can fit the input dimension of the LSTM layer. However, to avoid large IR drop penalties, we assume only 256 inputs are enabled at one phase. Hence, it requires  $\lceil \frac{633}{256} \rceil = 3$  phases to provide the inputs where each phase requires 32 ns for a 5-bit input with 1 GHz clock. As an alternative, the number of rows could be limited to 256 and 3 separate columns could be used to store these weights with a 3-phase multiplexing of these columns to the same integrator being implemented at the periphery. In that case, the same input lines have to be reused to provide different inputs for the three phases. This would increase the number of required crossbars by 3X to 48 assuming the maximum number of columns is kept fixed at 512. If the crossbar size is reduced to  $256 \times 256$  similar to other works<sup>7</sup>, the number of cores would increase to 96 which can be split across multiple chips if needed. For both cases, the

advantages of our proposed IMC architecture is retained over the conventional IMC one. We provide detailed estimates of our work (Tab S6) and the conventional model for the two cases of  $k = 1$  (Tab S7) and  $k = 8$  (Tab S8) digital processors. Table S9 presents a summary comparing the two methods showing 16X, 42X and 1.1X improvement in throughput, area efficiency and energy efficiency respectively for the 5-bit ADC case with  $k = 8$  digital processors in the conventional model.

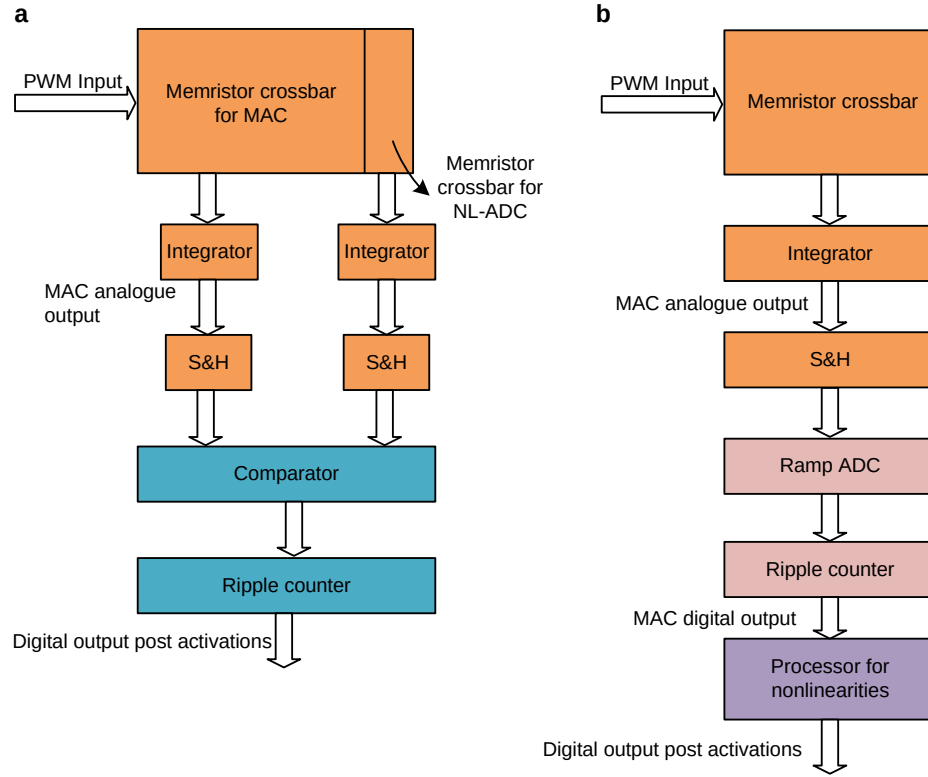


**Fig. S3. Architectures of this work and conventional ADC model.** **a** Architecture of this work. **b** Architecture of conventional ADC model. The massive matrix multiplications are distributed in  $72 \times 128$  crossbar arrays for KWS model. For our work, crossbar arrays comprises a 72 bit line (BL) drivers and 9216 1T1R devices. The peripheral circuits of crossbar include 128 integrators, 128 S&H blocks and 128 ripple counters. Additionally, an extra column of crossbar arrays, one integrator, one S&H block and 128 comparators are utilized for NL-ADC. The 5-bit ripple counter efficiently converts the thermometer code output from Macro into binary code. In order to process the computations, each real-valued input is divided into 5 binary pulse-width modulation (PWM) inputs. Compared to our work, the conventional model does not include NL-ADC circuits, but instead incorporates 128 ramp analog-to-digital converters (ADCs) and a processor for nonlinear operations.

#### A. KWS model (Macro level)

**Tab. S3.** Energy, area and latency estimation for **this work (5-bit NL-ADC)** at Macro level for KWS task.

| Module            | Number | On time (ns) | Area ( $\mu\text{m}^2$ ) | Energy (pJ) | Delay (ns) |
|-------------------|--------|--------------|--------------------------|-------------|------------|
| MAC array         | 72×128 | 32           | 126.45                   | 188.74      | <0.3       |
| NL-ADC array      | 32     | 32           | 0.44                     | 0.12        | <0.3       |
| Drivers           | 72     | 32           | 198.40                   | 3.92        | 0.1        |
| Integrator        | 129    | 32           | 1253.88                  | 324.42      | 0.2        |
| S&H               | 129    | 32           | 4.08                     | 0.41        | –          |
| Comparator        | 128    | 32           | 547.84                   | 33.10       | 0.1        |
| Ripple counter    | 128    | 32           | 36.48                    | 7.09        | –          |
| ADC (for writing) | 1      | –            | 280                      | –           | –          |
| Sum               | 9835   | –            | 2447.57                  | 557.79      | <1         |

Tab S3 shows energy, area and latency estimation for this work (5-bit NL-ADC) based on process node of 16 nm and frequency of 1 GHz. For MAC array, we use formula ( $E_{\text{MAC}} = N_{\text{row}} N_{\text{col}} (G_{\text{on}} + G_{\text{off}}) V_{\text{read}}^2 \bar{T}_{\text{on}}$ ) to calculate energy consumption.  $N_{\text{row}}$  and  $N_{\text{col}}$  is 72 and 128 respectively, we assume that  $G_{\text{off}}$  is 5  $\mu\text{S}$  and  $G_{\text{on}}$  is based on the value corresponding to the actual weight.  $V_{\text{read}}$  is 0.2 V. The input is 5-bit PWM waves, which means that for MAC operation, the maximum on-time of MAC array is 32 ns, and the minimum on-time is 0 ns. To calculate the energy, we take the average of these two values as  $\bar{T}_{\text{on}}$ . For NL-ADC, we also use aforementioned equation but  $G_{\text{on}}$  is from Fig. 2d and  $\bar{T}_{\text{on}}$  is 1 ns. We acquire data pertaining to drivers and S&H from reference<sup>2</sup>. The energy consumption and area information of the ripple counter are determined through Spice simulation and reference paper<sup>6</sup> respectively. The data regarding integrators and comparators are obtained from reference<sup>3</sup> and reference<sup>4</sup> and scaled respectively. According to timing of in-memory computing circuits in Fig. 2b, MAC and NL-ADC operation requires a total of 64 clock cycles. Taking into account the delay of the circuit, we take 65 ns as an evaluation period to evaluate the energy consumption. The power dissipation of the system is estimated by dividing the energy consumption for total latency ( 557.79 pJ) by the time needed for total latency (65 ns). The result of this calculation is 8.58 mW.

**Tab. S4.** Energy, area and latency estimation for **conventional ADC model (5-bit ADC)** at Macro level for KWS task.

| Module         | Number | On time (ns) | Area ( $\mu\text{m}^2$ ) | Energy (pJ) | Delay (ns) |
|----------------|--------|--------------|--------------------------|-------------|------------|
| MAC array      | 72×128 | 32           | 126.45                   | 188.74      | <0.3       |
| Drivers        | 72     | 32           | 198.40                   | 3.92        | 0.1        |
| Integrator     | 128    | 32           | 1244.16                  | 321.90      | 0.2        |
| S&H            | 128    | 32           | 4.04                     | 0.41        | –          |
| 5-bit Ramp-ADC | 128    | 32           | 4546.30                  | 256         | 0.2        |
| Ripple counter | 128    | 32           | 36.48                    | 7.09        | –          |
| Processor      | 1      | 256          | 119.17                   | 256         | 0.2        |
| Sum            | 9801   | –            | 6275.01                  | 829.26      | <1         |

Tab S4 shows energy, area and latency estimation for conventional ADC model(5-bit ADC) based on process node of 16 nm and frequency of 1 GHz. In the case of the conventional ramp ADC and processor, we get the data from reference<sup>5</sup> and reference<sup>6</sup> and scaled respectively. The evaluation method for other modules is the same as the previous one. The total latency of one period is 321 ns, which includes four components: 1 ns for circuit delay time, 32 ns for PWM input time, 32 ns for ADC conversion time, and 256 ns for the processor to calculate the nonlinear functions (One nonlinear function needs 2 clock cycles). The power dissipation of the system is estimated by dividing the energy consumption for total latency ( 829.26 pJ) by the time needed for total latency (321 ns). The result of this calculation is 2.58 mW.

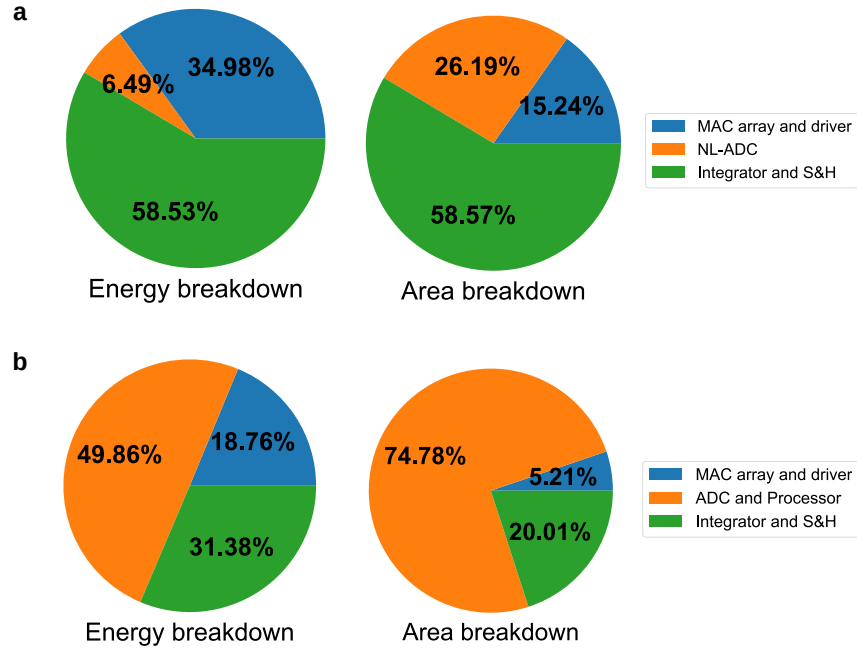


**Fig. S4. Energy and area breakdown based on Tab. S3 and Tab. S4 at Macro level for KWS task. a** Energy and area breakdown of this work. **b** Energy and area breakdown of conventional ADC model. The NL-ADC part in the figure includes the NL-ADC array, an integrator, a S&H, and 128 comparators. Our ADC has two functions, nonlinear calculation and conversion of analog signals to digital signals. Therefore, this is a reasonable comparison to a conventional ADC coupled with a processor that primarily computes nonlinearities.

**Tab. S5.** Comparison of the performance of our work for different NL-ADC resolution and the performance of conventional ADC work **at Macro level for KWS task.**

| Benchmark metric                        | This work(5-bit) | This work(4-bit) | This work(3-bit) | Conventional ADC model(5-bit) |
|-----------------------------------------|------------------|------------------|------------------|-------------------------------|
| Throughput (TOPS)                       | 0.28             | 0.56             | 1.08             | 0.06                          |
| Power (mW)                              | 8.58             | 8.43             | 8.12             | 2.58                          |
| Energy-efficiency (TOPS/W)              | 33.04            | 66.24            | 133.77           | 23.26                         |
| Area-efficiency (TOPS/mm <sup>2</sup> ) | 115.86           | 228.87           | 445.64           | 9.56                          |

### B. NLP model (Macro level)

**Tab. S6.** Energy, area and latency estimation for **this work (5-bit NL-ADC) at Macro level for NLP task.**

| Module            | Number   | On time (ns) | Area ( $\mu\text{m}^2$ ) | Energy (pJ) | Delay (ns) |
|-------------------|----------|--------------|--------------------------|-------------|------------|
| MAC array         | 633×8064 | 32           | 70039.01                 | 104540.41   | <0.3       |
| NL-ADC array      | 512      | 32           | 7.02                     | 1.86        | <0.3       |
| Drivers           | 10128    | 32           | 27908.7                  | 551         | 0.1        |
| Integrator        | 8065     | 96           | 78391.80                 | 60847.52    | 0.2        |
| S&H               | 8065     | 32           | 254.85                   | 25.81       | –          |
| Comparator        | 8064     | 32           | 34513.92                 | 2085.03     | 0.1        |
| Ripple counter    | 8064     | 32           | 2298.24                  | 446.42      | –          |
| ADC (for writing) | 16       | –            | 4480                     | –           | –          |
| Sum               | 5147426  | –            | 217893.57                | 168498.01   | <1         |

Due to the relatively large input data dimension in this model, we split the input data into three parts, each with a duration of 32 ns. Consequently, the operation time for the MAC array, integrator, and driver is 96 ns, while the comparator time remains unchanged at 32 ns. Additionally, considering a circuit delay of 1ns, the total latency is 129 ns. The energy evaluation method is the same as in Tab. S3.

**Tab. S7.** Energy, area and latency estimation for **conventional ADC model (5-bit ADC) at Macro level for NLP task (k=1).**

| Module         | Number   | On time (ns) | Area ( $\mu\text{m}^2$ ) | Energy (pJ) | Delay (ns) |
|----------------|----------|--------------|--------------------------|-------------|------------|
| MAC array      | 633×8064 | 32           | 70039.01                 | 104540.41   | <0.3       |
| Drivers        | 10128    | 32           | 27908.7                  | 551         | 0.1        |
| Integrator     | 8064     | 96           | 78382.08                 | 60839.98    | 0.1        |
| S&H            | 8064     | 32           | 254.82                   | 25.80       | –          |
| 5-bit Ramp-ADC | 8064     | 32           | 286417.15                | 16128       | 0.2        |
| Ripple counter | 8064     | 32           | 2298.24                  | 446.42      | –          |
| Processor (k)  | 1        | 16128        | 119.17                   | 16128       | 0.2        |
| Sum            | 5146897  | –            | 465419.19                | 185757.17   | <1         |

The variable "k" represents the number of processors in the system, which signifies the degree of parallelism in processing nonlinear functions. A higher value of "k" indicates a greater degree of parallelism, meaning that more processors are employed simultaneously for processing the nonlinear functions and the nonlinear processing time will be reduced. For this case, k=1. The input method is the same as the previous one. The total latency is 16257 ns. The energy evaluation method is the same as in Tab. S4.

**Tab. S8.** Energy, area and latency estimation for **conventional ADC model (5-bit ADC) at Macro level for NLP task(k=8).**

| Module         | Number   | On time (ns) | Area ( $\mu\text{m}^2$ ) | Energy (pJ) | Delay (ns) |
|----------------|----------|--------------|--------------------------|-------------|------------|
| MAC array      | 633×8064 | 32           | 70039.01                 | 104540.41   | <0.3       |
| Drivers        | 10128    | 32           | 27908.7                  | 551         | 0.1        |
| Integrator     | 8064     | 96           | 78382.08                 | 60839.98    | 0.2        |
| S&H            | 8064     | 32           | 254.82                   | 25.80       | –          |
| 5-bit Ramp-ADC | 8064     | 32           | 286417.15                | 16128       | 0.2        |
| Ripple counter | 8064     | 32           | 2298.24                  | 446.42      | –          |
| Processor (k)  | 8        | 2016         | 953.36                   | 16128       | 0.2        |
| Sum            | 5146904  | –            | 466253.38                | 185757.17   | <1         |

The evaluation conditions for the current scenario are identical to the previous one, with the exception that the value of "k" is now set to 8. The total latency of one period is 2145 ns. The energy evaluation method is the same as in Tab. S4.

**Tab. S9.** Comparison of the performance of our work for different NL-ADC resolution and the performance of conventional ADC work at Macro level for NLP task.

| Benchmark metric                        | This work (5-bit ) | This work (4-bit) | This work (3-bit) | Conv ADC model (5-bit, k=1) | Conv ADC model (5-bit, k=8) |
|-----------------------------------------|--------------------|-------------------|-------------------|-----------------------------|-----------------------------|
| Throughput (TOPS)                       | 79.14              | 157.06            | 309.36            | 0.62                        | 4.8                         |
| Power (mW)                              | 1306.2             | 1295.5            | 1275.2            | 11.4                        | 86.35                       |
| Energy-efficiency (TOPS/W)              | 60.77              | 121.62            | 243.36            | 55.11                       | 55.11                       |
| Area-efficiency (TOPS/mm <sup>2</sup> ) | 363.2              | 722.34            | 1425.81           | 1.35                        | 10.21                       |

**Supplementary Note S4. Energy, Area and Latency estimation at system level**

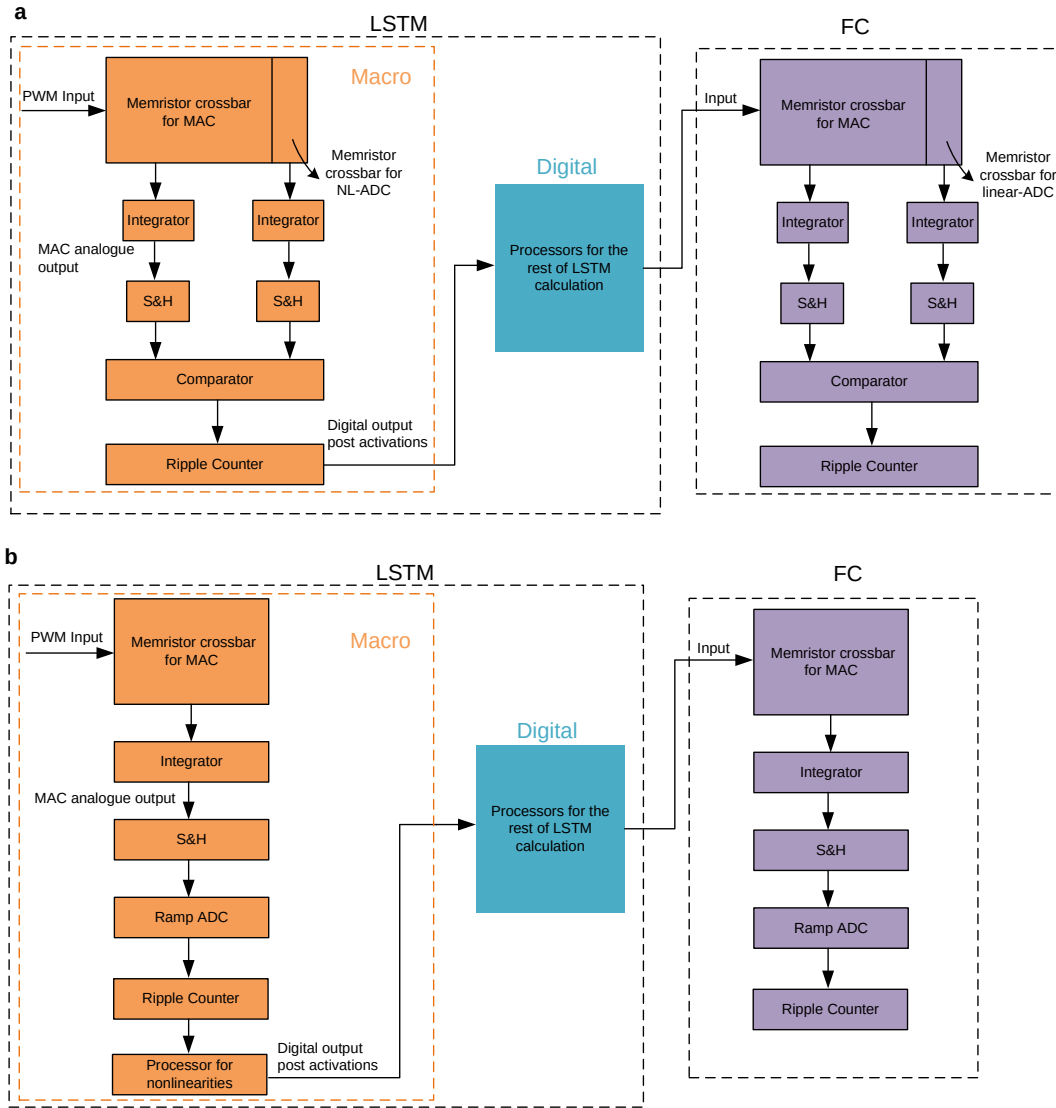


**Fig. S5.** Full system architectures of this work and conventional ADC model. **a** Full system architecture of this work. **b** Full system architecture of conventional ADC model.

We also consider the performance of full system comprising of two layers, LSTM layer and FC layer in Fig. S5a . The LSTM layer in Fig. S5a includes two parts, Macro part and digital part for calculating Equation (S2) and Equation (S3) respectively.

For the KWS model, the output from Macro consists of 128 5-bit binary numbers. This output data size is relatively small, eliminating the need for additional buffers or caches. The ripple counter module incorporates registers that serve as temporary

storage for the output data. This allows for direct access and reading of the data by the subsequent processor for further processing and analysis. But for the NLP model with a much bigger network, a simulator (NeuroSim<sup>9,10</sup>) is used to estimate the latency and energy requirements of the buffer and peripheral circuits.

The digital part of LSTM layer in Fig. S5 is computed in digital domain. Pipeline method is used to estimate the energy and latency of computing Equation (S3) in digital part in LSTM layer as depicted in Fig. S6. Pipeline 1 implements two elementwise multiplication simultaneously in Equation (S3) (left equation), which needs one clock cycle. Pipeline 2 finishes adding using one clock cycle and then  $h'_c$  in Equation (S3) (left equation) is obtained. For one  $\tanh(h'_c)$  in Equation (S3) (right equation), two clocks are needed at least and this can be done in Pipeline 3. Then Pipeline 4 can calculate the last elementwise multiplication in Equation (S3) (right equation) and then get the final LSTM output ( $h^t$ ). Therefore, total latency of four Pipelines is  $(2 * N_{tanh} + 3) * T_{clk}$ , where  $N_{tanh}$  is the number of Tanh function in Equation (S3) in one time step of LSTM layer and  $T_{clk}$  is clock period (1 ns) of digital processor<sup>6</sup>.

Equation (S2) and Equation (S3) are computed within Macro and digital domain, respectively. To minimize system latency and energy consumption, the FC layer is also computed within the RRAM array and its implementation method is the same as Macro in LSTM layer. The following evaluates the system performance of the KWS model and the NLP model respectively.

$$\begin{bmatrix} h'_f \\ h'_a \\ h'_i \\ h'_o \end{bmatrix} = \begin{bmatrix} \sigma \\ \tanh \\ \sigma \\ \sigma \end{bmatrix} \begin{bmatrix} x^t & h^{t-1} \end{bmatrix} \begin{bmatrix} W \\ U \end{bmatrix} \quad (S2)$$

$$h'_c = h'_f \odot h_c^{t-1} + h'_i \odot h_a^t \quad h^t = h'_o \odot \tanh(h'_c) \quad (S3)$$

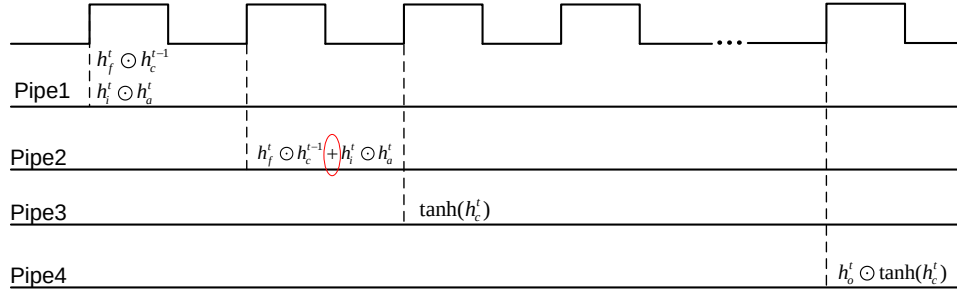


**Fig. S6. Pipeline for Equation (S3) calculation of LSTM layer.** We use digital processor<sup>6</sup> to calculate Equation (S3).

#### A. KWS model (full system level)

For KWS model, the dimensions of  $h'_f, h'_a, h'_i, h'_o$  in Equation (S3) are all 32. We leverage two digital processors to estimate the performance of digital part in LSTM layer. One processor calculates 16 Tanh operations. So according to this equation  $(2 * N_{tanh} + 3) * T_{clk}$ , the total latency for digital part in LSTM layer is  $(2 * 16 + 3) * 1 \text{ ns} = 35 \text{ ns}$  ( $N_{tanh} = 16, T_{clk} = 1 \text{ ns}$ ). We get power and area information of processor from reference paper<sup>6</sup>. Therefore energy consumption of digital part in LSTM layer can be obtained (power times latency).

The energy consumption and area information of the ripple counter are determined through Spice simulation and reference paper<sup>7</sup> respectively. The evaluation method of other modules in the FC layer is the same as that in the Macro level in the section A. KWS model (Macro level).

The performance of each module in digital part in LSTM layer and FC layer is shown in Tab. S10 below. The total area, latency, and energy consumption of digital part in LSTM layer and FC layer is  $513.78 \mu\text{m}^2, 98.3 \text{ ns}$  and  $46.24 \text{ pJ}$ .

Combining Tab. S3 and estimate of digital part, we can get full system data as shown in Tab. S10. The power dissipation of the full system is estimated by dividing the energy consumption ( $618.01 \text{ pJ}$ ) for total latency by the time needed for total latency ( $165.6 \text{ ns}$ ) and the result of this calculation is  $3.73 \text{ mW}$ .

**Tab. S10.** Energy, area and latency estimation for **this work (5-bit NL-ADC)** at system level for KWS task.

| Layer | Module                          | Number | On time (ns) | Area ( $\mu\text{m}^2$ ) | Energy (pJ) | Latency (ns) |
|-------|---------------------------------|--------|--------------|--------------------------|-------------|--------------|
| LSTM  | MAC array                       | 9216   | 32           | 126.45                   | 188.74      | 65.3         |
|       | NLADC array                     | 32     | 32           | 0.4391                   | 0.1165      |              |
|       | Drivers                         | 72     | 32           | 198.4                    | 3.9168      |              |
|       | Integrator                      | 129    | 32           | 1253.9                   | 324.42      |              |
|       | Comparator                      | 128    | 32           | 547.84                   | 33.096      |              |
|       | S&H                             | 128    | 32           | 4.0448                   | 0.4096      |              |
|       | Ripple counter                  | 128    | 32           | 36.48                    | 7.0861      |              |
| LSTM  | Processors for the rest of LSTM | 2      | 35           | 238.34                   | 14          | 35           |
| FC    | MAC array                       | 384    | 32           | 5.2689                   | 7.8643      | 65.3         |
|       | ADC array                       | 32     | 32           | 0.4391                   | 0.1165      |              |
|       | Drivers                         | 32     | 32           | 88.179                   | 1.7408      |              |
|       | Integrator                      | 13     | 32           | 126.36                   | 32.693      |              |
|       | Comparator                      | 12     | 32           | 51.36                    | 3.1027      |              |
|       | S&H                             | 13     | 32           | 0.4108                   | 0.0416      |              |
|       | Ripple counter                  | 12     | 32           | 3.42                     | 0.6643      |              |
|       | ADC (for writing)               | 1      | –            | 280                      | –           | –            |
|       | Sum                             | 10334  |              | 2961.32                  | 618.01      | 165.6        |

For conventional ADC model, we utilize conventional ADC to estimate both digital part in LSTM layer and FC layer as depicted in Tab. S11 and the method is same as Tab. S4. The power dissipation of the full system is estimated by dividing the energy consumption (910.27 pJ) for total latency by the time needed for total latency (421.6 ns) and the result of this calculation is 2.16 mW.

**Tab. S11.** Energy, area and latency estimation for **conventional ADC model (5-bit ADC)** at system level for KWS task.

| Layer | Module                          | Number | On time (ns) | Area ( $\mu\text{m}^2$ ) | Energy (pJ) | Latency (ns) |
|-------|---------------------------------|--------|--------------|--------------------------|-------------|--------------|
| LSTM  | MAC array                       | 9216   | 32           | 126.452736               | 188.74368   | 321.3        |
|       | 5 bit RA- ADC                   | 128    | 32           | 4546.304                 | 256         |              |
|       | Drivers                         | 72     | 32           | 198.4032                 | 3.9168      |              |
|       | Integrator                      | 128    | 32           | 1244.16                  | 321.90464   |              |
|       | S&H                             | 128    | 32           | 4.0448                   | 0.4096      |              |
|       | Ripple counter                  | 128    | 32           | 36.48                    | 7.08608     |              |
|       | Processor(NL)                   | 1      | 256          | 119.17                   | 51.2        |              |
| LSTM  | Processors for the rest of LSTM | 2      | 35           | 238.34                   | 14          | 35           |
| FC    | MAC array                       | 384    | 32           | 5.268864                 | 7.86432     | 65.3         |
|       | 5 bit RA- ADC                   | 12     | 32           | 426.216                  | 24          |              |
|       | Drivers                         | 32     | 32           | 88.1792                  | 1.7408      |              |
|       | Integrator                      | 13     | 32           | 126.36                   | 32.69344    |              |
|       | S&H                             | 13     | 32           | 0.4108                   | 0.0416      |              |
|       | Ripple counter                  | 12     | 32           | 3.42                     | 0.66432     |              |
|       | Sum                             | 10409  |              | 7163.21                  | 910.27      | 421.6        |

**Tab. S12.** Comparison of the performance of full system for different NL-ADC resolution and the performance of conventional 5-bit ADC work **at system level for KWS task**. In terms of throughput, energy efficiency and area efficiency, this work is 2 times, 1.5 times and 6.8 times that of traditional conventional architectures at the system level.

| Benchmark metric                        | This work(5-bit) | This work(4-bit) | This work(3-bit) | Conventional ADC model(5-bit) |
|-----------------------------------------|------------------|------------------|------------------|-------------------------------|
| Throughput (TOPS)                       | 0.12             | 0.19             | 0.28             | 0.06                          |
| Power (mW)                              | 3.73             | 3.15             | 2.41             | 2.16                          |
| Energy-efficiency (TOPS/W)              | 31.33            | 60.09            | 114.40           | 21.27                         |
| Area-efficiency (TOPS/mm <sup>2</sup> ) | 39.48            | 63.36            | 92.78            | 6.41                          |

**Tab. S13.** Energy efficiency comparison **at various levels for KWS task**: MAC array, NL-processing, full system. For this work, the energy-efficiency calculation of NL-processing module takes into account the NL-ADC array, an integrator, a S&H, and 128 comparators.

| Energy-efficiency (TOPS/W) | This work(5-bit NL-ADC) | Conventional ADC model(5-bit ADC) | Nature'23 <sup>8</sup> |
|----------------------------|-------------------------|-----------------------------------|------------------------|
| MAC array                  | 97.6                    | 97.6                              | 20.0                   |
| NL-processing              | 3.6                     | 0.3                               | 0.9                    |
| Macro level                | 33.0                    | 23.3                              | 7.09                   |
| Full system                | 31.33                   | 21.27                             | 6.9                    |

### B. NLP model (full system level)

The buffer and interconnect results are simulated in NeuroSim. The trained model is mapped onto the RRAM-based IMC architecture, where each tile consists of  $2 \times 2$  PEs and each PE is composed of  $4 \times 4$  synaptic arrays. The array size is set to  $128 \times 128$  1T1R array at 16 nm technology node. Thus, the PE and tile sizes are  $512 \times 512$  and  $1024 \times 1024$ , respectively. When mapping LSTM layer weight onto arrays, the weight matrices ( $LSTM_w$  size:  $128 \times (2016 \times 4)$  and  $LSTM_u$  size:  $504 \times (2016 \times 4)$ ) are partitioned and assigned to different synaptic arrays due to the limited array size and then sum up the partial results from arrays to get final results. In the column dimension, we need 63 synaptic arrays (i.e.,  $2016 \times 4/128$ ), which is equivalent to 16 PEs, or 8 tiles. In the row dimension, 5 synaptic arrays are needed (i.e.,  $(128+504)/128$ ), which is equal to 2 PEs or 1 tile. So,  $8 \times 1$  tiles are used to implement LSTM layer. Similarly, 1 tile is enough to implement a FC layer weight with small size ( $504 \times 50$ ). In terms of the device characteristics, the transistor has a 1.1 V gate voltage and 15 k $\Omega$  access resistance. The memristor has 128 conductance states, on/off ratio of 30, on-state resistance of 7 k $\Omega$ , and 0.2 V read voltage. Then the evaluated buffer and interconnect results are given as shown in Tab. S14.

Combining Tab. S6 with estimate of the digital processor, FC layer, interconnect and buffer, we can get full system data as shown in Tab. S14. The power dissipation of the full system is estimated by dividing the energy consumption (196,367 pJ) for total latency by the time needed for total latency (526.9 ns) and the result of this calculation is 372.683 mW.

**Tab. S14.** Energy, area and latency estimation for **this work (5-bit NL-ADC) at system level for NLP task.**

| Layer | Module                          | Number  | On time (ns) | Area ( $\mu\text{m}^2$ ) | Energy (pJ) | Latency (ns) |
|-------|---------------------------------|---------|--------------|--------------------------|-------------|--------------|
| LSTM  | MAC array                       | 5104512 | 32           | 70039.0092               | 104540.4058 | 129          |
|       | NLADC array                     | 512     | 32           | 7.03                     | 1.86        |              |
|       | Drivers                         | 10128   | 32           | 27908.7                  | 550.96      |              |
|       | Integrator                      | 8065    | 96           | 78391.8                  | 60847       |              |
|       | Comparator                      | 8064    | 32           | 34513.92                 | 2085.02784  |              |
|       | S&H                             | 8065    | 32           | 254.854                  | 25.808      |              |
|       | Ripple counter                  | 8064    | 32           | 2298.24                  | 446.42304   |              |
| LSTM  | Processors for the rest of LSTM | 30      | 137.4        | 3575.1                   | 824.4       | 137.4        |
| FC    | MAC array                       | 25200   | 32           | 345.7692                 | 516.096     | 65.3         |
|       | ADC array                       | 32      | 32           | 0.44                     | 0.12        |              |
|       | Drivers                         | 504     | 32           | 1388.8224                | 27.4176     |              |
|       | Integrator                      | 51      | 32           | 495.72                   | 128.25888   |              |
|       | Comparator                      | 50      | 32           | 214                      | 12.928      |              |
|       | S&H                             | 51      | 32           | 1.6116                   | 0.1632      |              |
|       | Ripple counter                  | 50      | 32           | 14.25                    | 2.768       |              |
| All   | Buffer                          | –       | 71.7         | 50916                    | 36751.8     | 71.7         |
|       | Interconnect                    | –       | 123.5        | 433261                   | 7890.42     | 123.5        |
|       | ADC (for writing)               | 16      | –            | 4480                     | –           | –            |
|       | Sum                             | 5173394 |              | 705793.78                | 214203.19   | 526.9        |

**Tab. S15.** Energy, area and latency estimation for **conventional ADC model (5-bit ADC) at system level for NLP task**. The variable "k" represents the number of processors in the system, which signifies the degree of parallelism in processing nonlinear functions. A higher value of "k" indicates a greater degree of parallelism, meaning that more processors are employed simultaneously for processing the nonlinear functions and the nonlinear processing time will be reduced. **For this case, k=1.**

| Layer | Module                          | Number   | On time (ns) | Area ( $\mu\text{m}^2$ ) | Energy (pJ) | Latency (ns) |
|-------|---------------------------------|----------|--------------|--------------------------|-------------|--------------|
| LSTM  | MAC array                       | 5104512  | 32           | 70039.0092               | 104540.4058 | 129          |
|       | 5 bit RA- ADC                   | 8064     | 32           | 286417.152               | 16128       |              |
|       | Drivers                         | 10128    | 32           | 27908.7                  | 550.96      |              |
|       | Integrator                      | 8064     | 96           | 78382.08                 | 60839       |              |
|       | S&H                             | 8064     | 32           | 254.8224                 | 25.8048     |              |
|       | Ripple Counter                  | 8064     | 32           | 2298.24                  | 446.42304   |              |
|       | Processor(k)                    | <b>1</b> | 16128        | 119.17                   | 3225.6      | 16128        |
| LSTM  | Processors for the rest of LSTM | 30       | 137.4        | 3575.1                   | 824.4       | 137.4        |
| FC    | MAC array                       | 100800   | 32           | 1383.0768                | 2064.384    | 65.3         |
|       | 5 bit RA- ADC                   | 50       | 32           | 1775.9                   | 100         |              |
|       | Drivers                         | 2016     | 32           | 5555.2896                | 109.6704    |              |
|       | Integrator                      | 51       | 32           | 495.72                   | 128.25888   |              |
|       | S&H                             | 51       | 32           | 1.6116                   | 0.1632      |              |
|       | Ripple counter                  | 50       | 32           | 14.25                    | 2.768       |              |
| All   | Buffer                          | –        | 71.7         | 50916                    | 36751.8     | 71.7         |
|       | Interconnect                    | –        | 123.5        | 433261                   | 7890.42     | 123.5        |
|       | Sum                             | 5174352  | –            | 961359.83                | 232080.75   | 16655.2      |

**Tab. S16.** Energy, area and latency estimation for **conventional ADC model (5-bit ADC) at system level for NLP task**. The variable "k" represents the number of processors in the system, which signifies the degree of parallelism in processing nonlinear functions. A higher value of "k" indicates a greater degree of parallelism, meaning that more processors are employed simultaneously for processing the nonlinear functions and the nonlinear processing time will be reduced. **For this case, k=8.**

| Layer | Module                          | Number   | On time (ns) | Area ( $\mu\text{m}^2$ ) | Energy (pJ) | Latency (ns) |
|-------|---------------------------------|----------|--------------|--------------------------|-------------|--------------|
| LSTM  | MAC array                       | 5104512  | 32           | 70039.0092               | 104540.4058 | 129          |
|       | 5 bit RA- ADC                   | 8064     | 32           | 286417.152               | 16128       |              |
|       | Drivers                         | 10128    | 32           | 27908.7                  | 550.96      |              |
|       | Integrator                      | 8064     | 96           | 78382.08                 | 60839       |              |
|       | S&H                             | 8064     | 32           | 254.8224                 | 25.8048     |              |
|       | Ripple Counter                  | 8064     | 32           | 2298.24                  | 446.42304   |              |
|       | Processor(k)                    | <b>8</b> | 2016         | 119.17                   | 3225.6      | 2016         |
| LSTM  | Processors for the rest of LSTM | 30       | 137.4        | 3575.1                   | 824.4       | 137.4        |
| FC    | MAC array                       | 100800   | 32           | 1383.0768                | 2064.384    | 65.3         |
|       | 5 bit RA- ADC                   | 50       | 32           | 1775.9                   | 100         |              |
|       | Drivers                         | 2016     | 32           | 5555.2896                | 109.6704    |              |
|       | Integrator                      | 51       | 32           | 495.72                   | 128.25888   |              |
|       | S&H                             | 51       | 32           | 1.6116                   | 0.1632      |              |
|       | Ripple counter                  | 50       | 32           | 14.25                    | 2.768       |              |
| All   | Buffer                          | –        | 71.7         | 50916                    | 36751.8     | 71.7         |
|       | Interconnect                    | –        | 123.5        | 433261                   | 7890.42     | 123.5        |
|       | Sum                             | 5174352  | –            | 962194.02                | 232080.75   | 2543.2       |

**Tab. S17.** Comparison of the performance for different NL-ADC resolution and the performance of conventional ADC work **at system level for NLP task**. In terms of throughput, energy efficiency and area efficiency, this work is 4.9 times, 1.1 times and 7.9 times that of traditional conventional architectures at the system level.

| Benchmark metric                        | This work<br>(5-bit ) | This work<br>(4-bit) | This work<br>(3-bit) | Conv ADC model<br>(5-bit, k=1) | Conv ADC model<br>(5-bit, k=8) |
|-----------------------------------------|-----------------------|----------------------|----------------------|--------------------------------|--------------------------------|
| Throughput (TOPS)                       | 19.49                 | 25.14                | 30.23                | 0.62                           | 4.03                           |
| Power (mW)                              | 406.5                 | 263.87               | 181.9                | 13.9                           | 91.2                           |
| Energy-efficiency (TOPS/W)              | 47.9                  | 95.3                 | 166.5                | 44.2                           | 44.2                           |
| Area-efficiency (TOPS/mm <sup>2</sup> ) | 27.6                  | 59.51                | 72.35                | 0.64                           | 4.2                            |

## Supplementary Note S5. Programming of NL-ADC on crossbar arrays

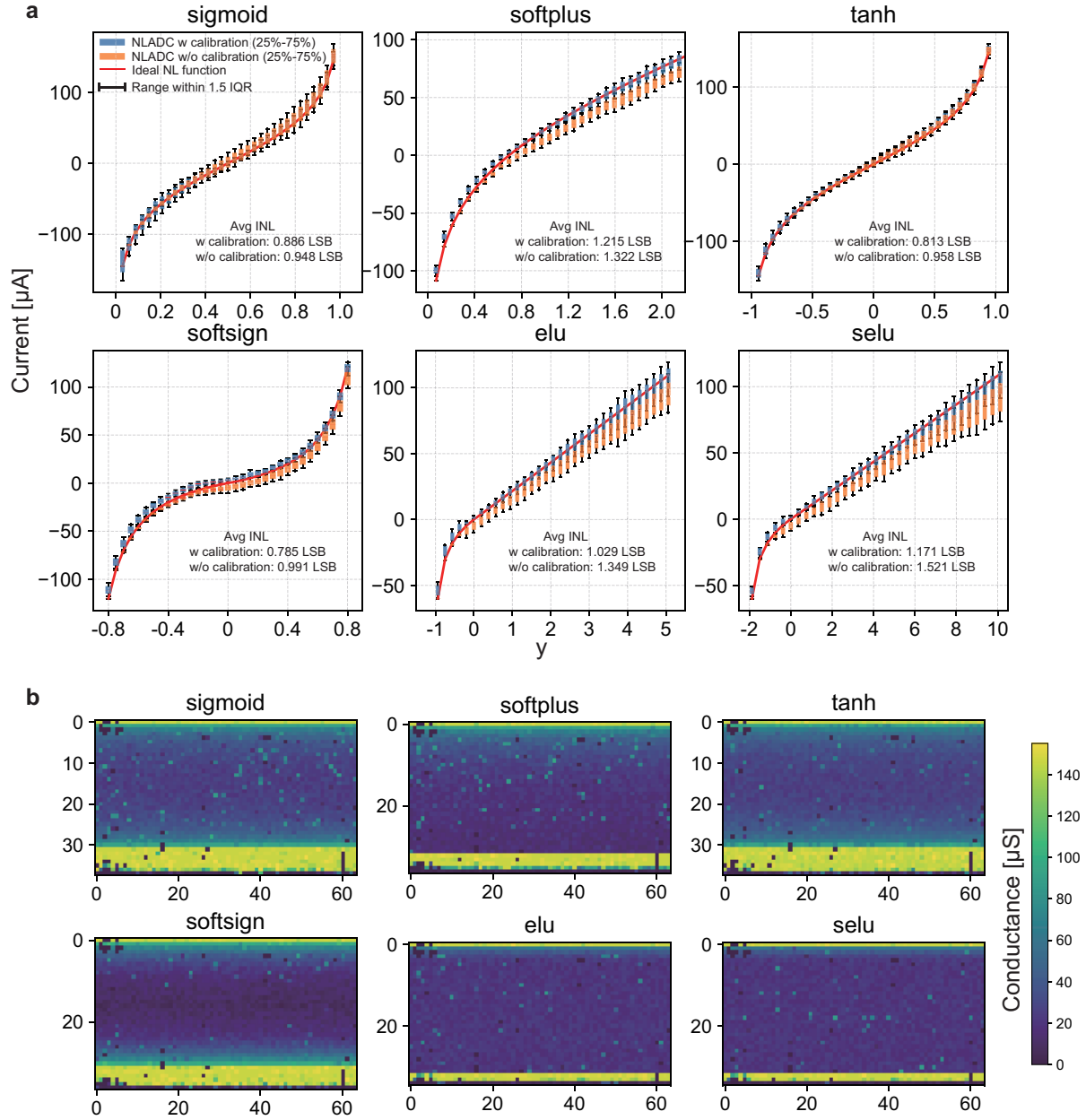


**Fig. S7. Programming of NL-ADC on crossbar arrays.** **a** Transfer function of different NL-functions after mapping on real crossbar array with bias term and without bias term. **b** Actual conductance map of 6 different NL-ADC weights and bias mapped to 64 crossbar columns.

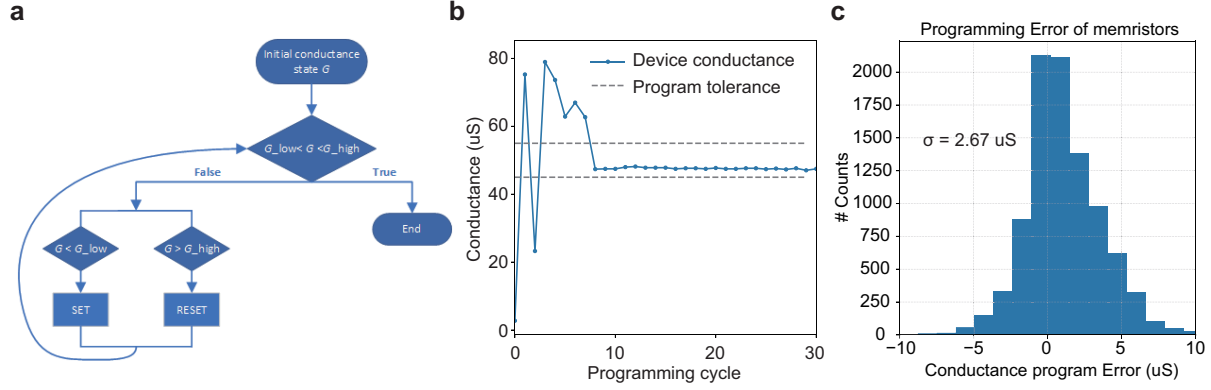


**Fig. S8. Iterative programming on memristor crossbar arrays** **a** Flow char showing the process of programming the entire array. **b** Conductance updating plot of a single device during iterative programming. It shows that under several programming cycles, the conductance finally lies in the tolerated range. **c** Programming error distribution shows that with our iterative programming method, we can achieve programming standard deviation about 2.67  $\mu S$ .

Typically 30-40 iterations are needed for accurate programming, even including repeated programming to counter drift<sup>11</sup>. As a conservative estimate, we consider 100 iterations to program each device. Using a 8-bit version of the current controlled oscillator (CCO) based ADC from<sup>7</sup>, we can estimate 256 ns for each conversion cycle for a read operation and a few ns for any control operations. Do note that this is also a conservative estimate and ADCs with conversion time of 5 ns are easily available<sup>12</sup> Hence, the ADC operation time required for programming a  $256 \times 256$  crossbar of memristors, sufficient for KWS, is  $\approx 1.68$  sec. For the large NLP task with  $\approx 6$  millions parameters, the estimated ADC operation time is 157 sec or less than 3 minutes. Hence, we can see that this is not a bottleneck in terms of programming time.

### Supplementary Note S6. Latency estimation of MAC operation in In-memory computing and Digital Non-linear function implementations

The latency for a MAC operation ( $T_{MAC}$ ) using In-memory computing (IMC) comprises two parts—input generation ( $T_{in}$ ) and output generation ( $T_{out}$ ) by analog to digital conversion. Assuming PWM inputs with bit-width  $b_{in}$  and ramp/CCO ADC for output generation with bit-width  $b_{out}$ , the latency is given by the following equation:

$$T_{MAC} = T_{in} + T_{out} = 2^{b_{in}}T_{clk} + (2^{b_{out}} - 1)T_{clk} \quad (S4)$$

where  $T_{clk}$  denotes the clock period. We choose ramp/CCO ADC since these have areas small enough to be pitch-matched with memory and have been used in recent IMC research<sup>7,8,13,14</sup>. The same equation can be generalized to other ADCs by modifying the equation of  $T_{out}$  (e.g.  $T_{out} = b_{out}$  for a Successive Approximation ADC).

The latency of nonlinear function approximation ( $T_{NL}$ ) using digital processors depends on the method used and desired accuracy. For commonly used methods of cordic<sup>6</sup>, look-up table (LUT) and piece wise linear functions<sup>15,16</sup>, we estimate  $N_{cyc} = 2 - 5$  cycles being needed with a minimum of 2 cycles needed in the case of LUT to read the input and fetch the corresponding data. A second factor affecting the latency is the number of hidden neurons  $N_h$  for which this computation has to be done as well as the number  $k$  of parallel digital processors available. The latency for nonlinear function approximation can then be expressed as:

$$T_{NL} = \frac{4 \times N_h \times N_{cyc}}{k} T_{clk} \quad (S5)$$

where the multiple of 4 is due to the four gates per neuron in an LSTM cell. The ratio  $\frac{T_{NL}}{T_{MAC}}$  is plotted in Fig. 1c for various values of  $k$  with  $b_{in} = b_{out} = 5$ ,  $N_{cyc} = 2$  or 5 and  $N_h = 512$  which is representative of common applications. Here, each digital processor handles  $\frac{N_h}{k}$  neurons. As an example, a recent work<sup>7</sup> shares one digital processor among 8 MAC cores with each core having 128 neurons. Thus each digital processor handles  $128 \times 8 = 1024$  neurons which is worse than all cases considered in Fig. 1c.

## Supplementary Note S7. Circuit implementation

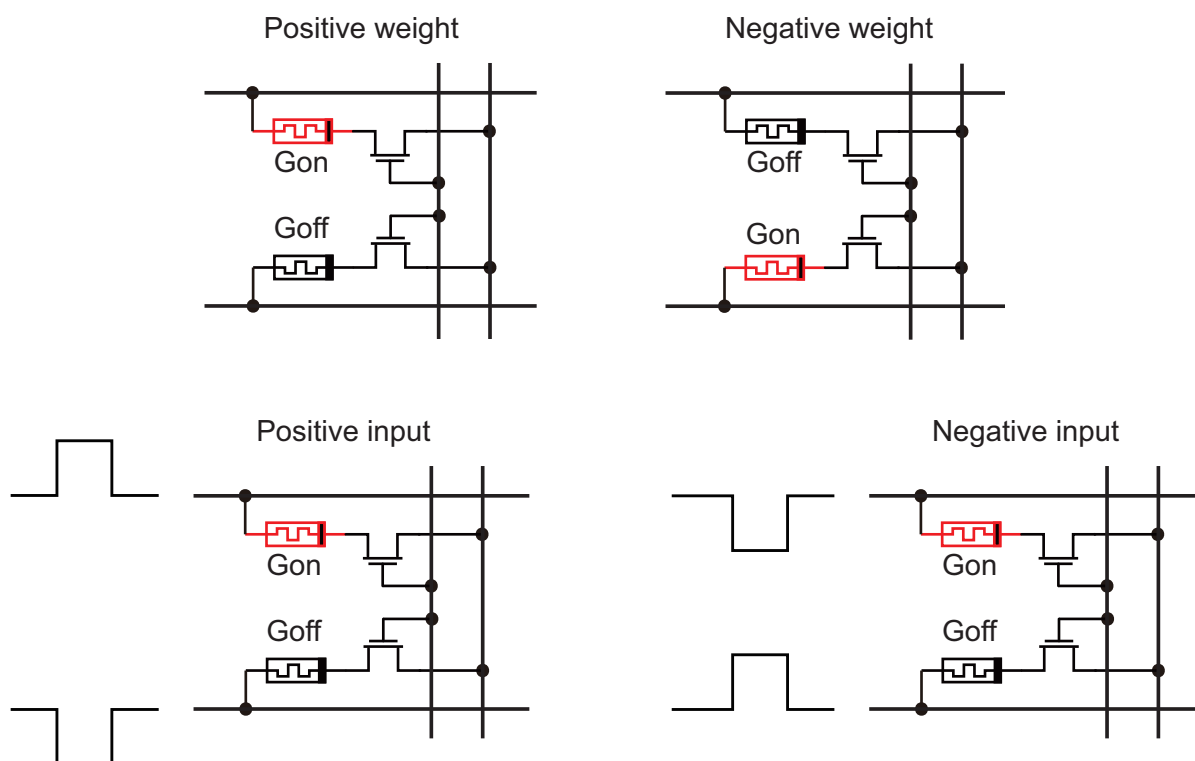


**Fig. S9.** This figure illustrates how we map both the positive and negative weight/inputs to the crossbar arrays. We use differential encoding by using two memristors in a single column to represent a single weight and 2 input lines to represent a single input signal.

### Supplementary Note S8. Analysis of Accuracy Drop with Write Noise

If we analyze the 5-bit NL-ADC case for the KWS task, there is a drop of 1.7% in accuracy in software simulation including hardware non-idealities (weight noise and NL-ADC noise in Fig. 4d) from the baseline. Chip testing resulted in only 0.9% drop from this earlier software simulation. This large drop is attributed to the fact that LSTMs are more sensitive to errors than conventionally reported fully connected (FC) or convolutional networks (CNN). To verify this, we have now simulated a 2 layer FC network with ReLU activations (mirroring the one in<sup>8</sup>) on the same KWS task. The software test accuracy, (exclusive of hardware non-ideal factors) was determined to be 86.1%. Following the introduction of hardware noise (derived from measured write noise) into this network, the accuracy decreased marginally to 85.3%. This experiment demonstrated a modest accuracy reduction of 0.8%, much less than the 1.7% drop for the case of LSTM with 5-bit NL-ADC, proving that the large drop is specific to recurrent architectures like LSTM. Further, we also tested the VGG-8 and VGG-16 networks on CIFAR-10 dataset with 5-bit ADC quantization and memristor write noise. For the VGG-16 networks, the accuracy reduced from 93.9% to 92.9%; in comparison, the accuracy for VGG-16 on CIFAR-10 in<sup>17</sup> is 92.57% when tested with hardware non-idealities. For the VGG-8 networks, the accuracy reduced from 93% to 92.6%; in comparison, the accuracy for VGG-8 on CIFAR-10 in<sup>18</sup> and<sup>19</sup> are 90.7% and 89% respectively when tested with hardware non-idealities. These results show that the hardware non-idealities and training methods used in our work are comparable to those used by other research groups.

Second, the drop in accuracy is largely due to the error in programming and weights and not due to the NL-ADC, which is the focus of this paper. In the current Fig. 4d, the effect of both are present and hence, the contribution of each is not obvious. To tease apart the two effects, we did simulations where the write noise was added only in weights or the NL-ADC. For the 5-bit case, the accuracies were 90.7% (NL-ADC only) and 89.9% (weight only) compared to a baseline of 91.1%. For the 3-bit case, the accuracies were 88.9% (NL-ADC only) and 87.8% (weight only) compared to a baseline of 89.4%. As can be seen, the effect of write noise on NL-ADC is less critical. Further, this can be reduced even more by using the redundancy technique described in Section [Supplementary Note S11.](#)

Lastly, improving the quality of devices<sup>20</sup> as well as programming techniques<sup>21</sup> for better accuracies in writing to memristors are subjects of ongoing research in the community. We hope with future improvements in both these areas, the drop in accuracy from software to hardware can be further minimized in future.

## Supplementary Note S9. One-point calibration process

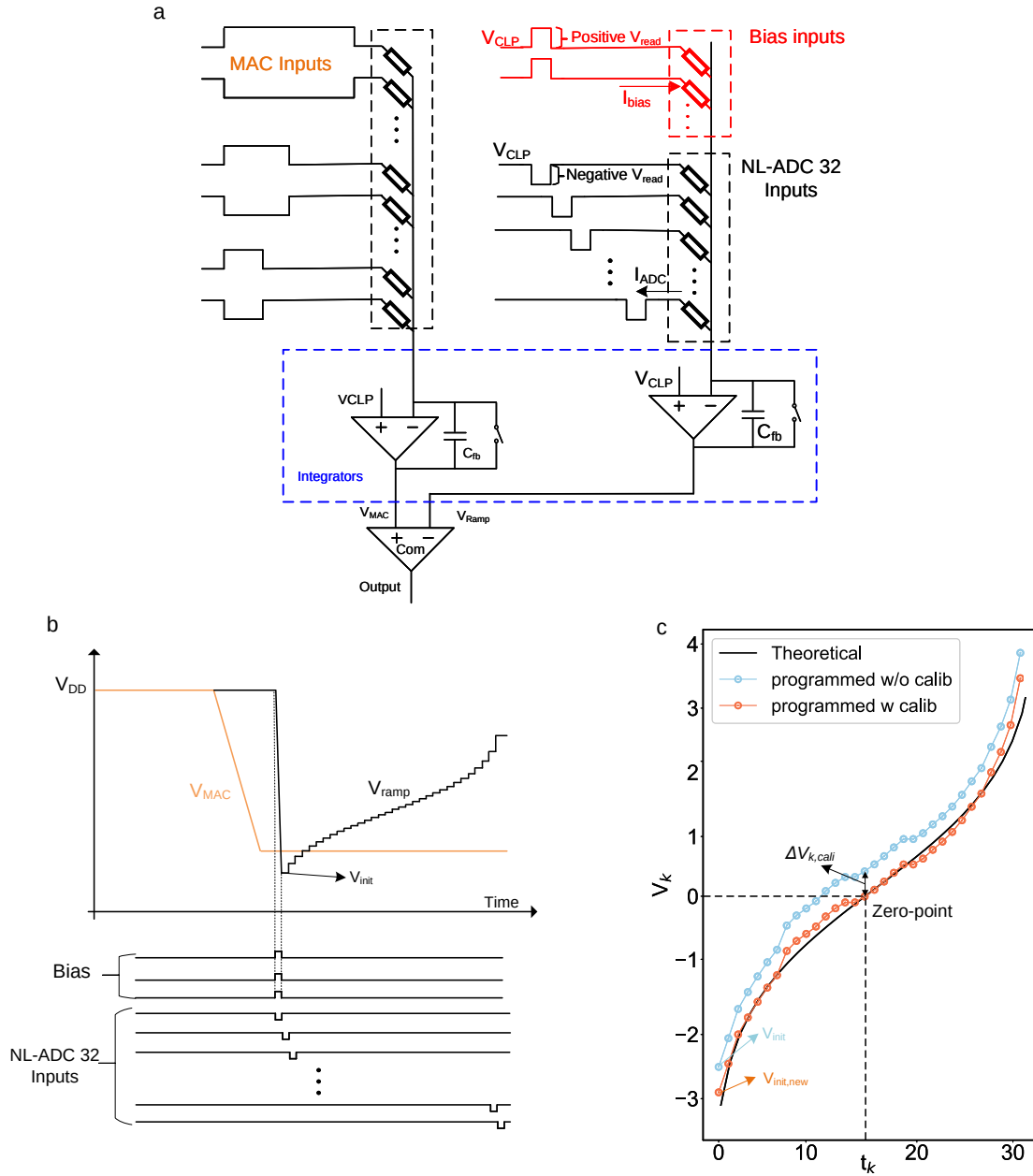


**Fig. S10. One-point calibration.** **a** Hardware block diagram of  $V_{MAC}$  and  $V_{ramp}$ . **b** Timing diagram of  $V_{MAC}$  and  $V_{ramp}$ . **c** Comparison of actual programmed  $V_k$  (same as  $V_{ramp}$ ) with calibration and without calibration.

In the experimental setup, where we are using a 5-bit ADC as an example, the initial step involves programming the conductance values of 32 ADCs and bias conductance value into a column of RRAM. Subsequently, 32 pulses are sequentially sent to the RRAM corresponding to each ADC, as depicted in Fig. S10a. The desired ramp voltage initially ( $V_{init}$  in Fig. S10b) starts from the smallest value in the first clock cycle and then increases progressively in every clock cycle after that to cover the full range of MAC values (Fig. S10b). The initial negative voltage drop is created using the bias memristors.

It is important to note that the pulses for bias are configured as positive input, while the pulses for the 32 NL-ADC cell inputs are configured as negative input (as illustrated in Fig. S10a). This polarity distinction arises due to the negative  $V_{read}$  of the NL-ADC pulses, resulting in the current flowing from the BL to the input of the RRAM. Consequently, these currents generate a positive voltage at the output after passing through the integrator, thereby causing all  $V_{ramp}$  to be positive. However,

in actuality,  $V_{ramp}$  typically spans both positive and negative values. Thus, the pulse for bias necessitates a positive  $V_{read}$  value to generate the desired negative voltage at the initial point ( $V_{init}$  in Fig. S10b). The current direction for bias RRAM cells is from the input of the RRAM to BL. Upon passing through the integrator, a negative voltage is generated, serving as the starting point ( $V_{init}$ ) for the NL-ADC.

The blue curve in Fig. S10c is the actual programmed value of  $V_{ramp}$  without calibration. We can see that it deviates a lot from the theoretical value. To reduce the average INL (or average deviation of programmed ramp from the desired ramp), we use the one-point calibration method where the original starting point  $V_{init}$  is modified to become  $V_{init,new}$ , such that the zero crossing point of the actual and desired ramps overlap. Based on the formula  $\Delta V_{k,cali} = \frac{1}{C_{fb}} V_{read} G_{cali} T_{adc}$  ( $\Delta V_{k,cali}$  is shown in Fig. S10c, which is the vertical distance between two middle points of blue curve and red curve), the required RRAM conductance ( $G_{cali}$ ) for calibration is determined. The new bias conductance value is obtained by adding the original bias conductance value to  $G_{cali}$ . Then we can get practical programmed  $V_k$  with calibration in Fig. S10c, where we can see  $V_k$  with calibration is much closer to the theoretical value than  $V_k$  without calibration.

### Supplementary Note S10. Capacitor-based accumulation method for high-dimensional inputs

The method shown earlier can only handle input vectors with dimension less than the number of rows in the RRAM Macro. We show here how this method can be extended to calculate partial dot products and combine them later. In this case, the partial dot products can be stored as charge on the integrating capacitor. As shown in Fig. S11, if the input vector dimension is more than the number of rows of the memory, multiple columns (in this case, 3 columns are used) may be used to store the weights. Then the input vector is also split into the same number of parts and is applied to the memory array in different clock cycles. A switch is used to connect the integrator to different columns in each of these cycles, where each of these columns store the weights for the respective part of the inputs. In the example shown, the input vector  $X$  is split into 3 parts— $X_1$ ,  $X_2$  and  $X_3$ . So in the first clock cycle, the dot product  $X_1 \cdot W_1$  is calculated and stored on the capacitor. In the 2nd cycle,  $X_2 \cdot W_2$  is computed and added to the same capacitor and so on. Thus by using partial products in the analog domain, our method can be extended to handle input vectors beyond the row-depth of the memory.

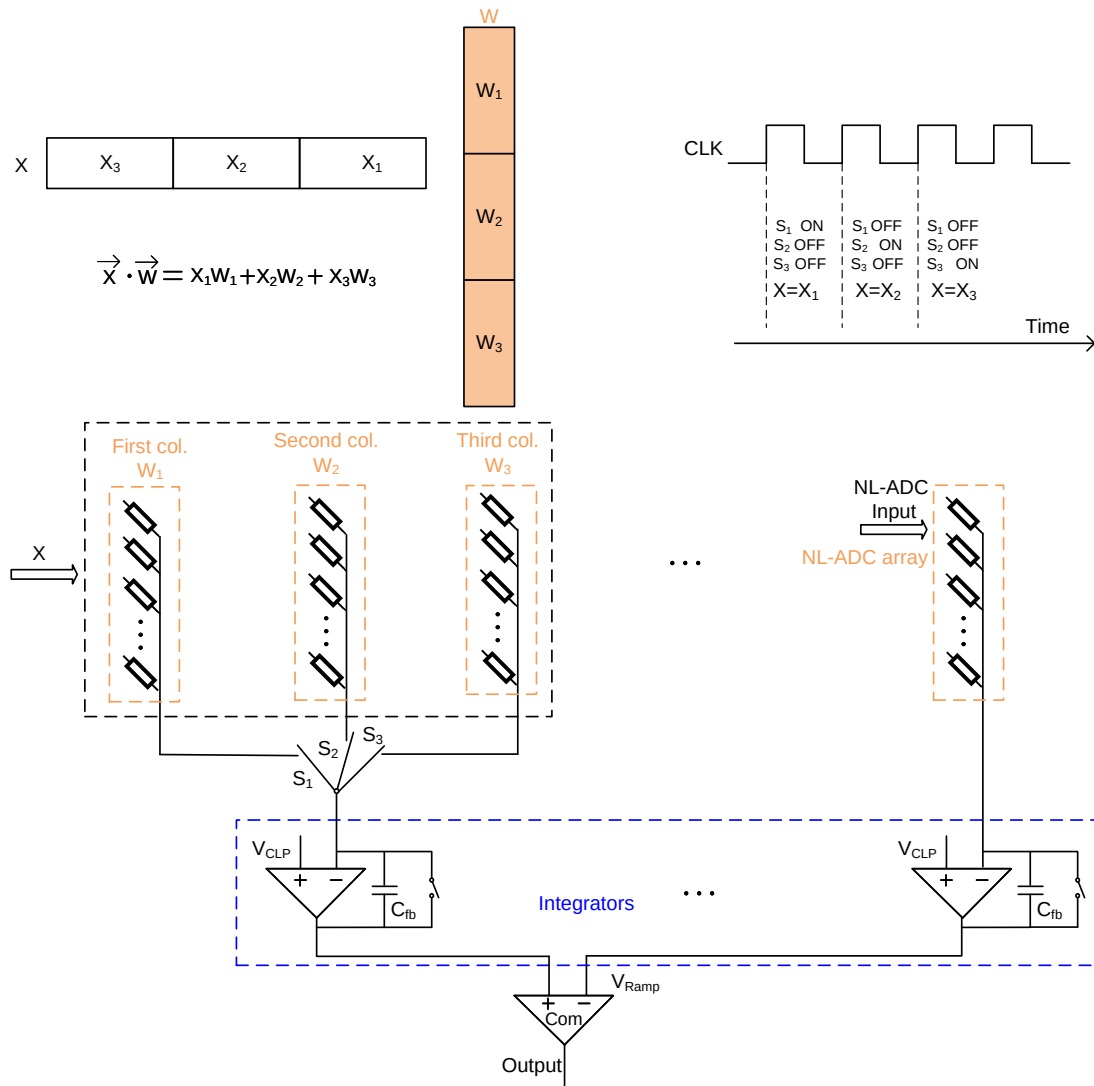


**Fig. S11. Capacitor-based accumulation method for large model.** Input vectors with dimension larger than the number of rows can be applied by splitting it into parts and applying them sequentially over time. The corresponding weights are programmed on different neighbouring columns and are switched to an integrator following the same sequence. The capacitor accumulates the final MAC value over multiple cycles.

## Supplementary Note S11. Redundancy for improved NL-ADC programming

We reduce variability and improves accuracy a lot with a redundancy based method. Briefly, the column used to generate the ramp for the ADC consists of the same number of memristors as those used for MAC, i.e. 64. However, when we use only 32 out of these for a 5-bit NL ADC, the remaining memristors in the same column are unused. The number of unused memristors are even more for 4-bit and 3-bit versions of the NL-ADC. We propose to use them and program redundant copies of the ADC reference in the case of 3-bit and 4-bit ADCs. For the 5-bit ADC, the starting location (row address in that column) of the ramp can be varied as well as multiple programming attempts can be made to get the best NL-ADC characteristic. This requires minimal overhead of only an extra register of 6-bits to store the base or starting address of the ramp for every crossbar. Alternatively, additional columns may also be used for programming redundant copies of the NL-ADC—this is the approach we have taken to get the new measured results below. For larger crossbars like 128x128, many such redundant copies can be fit into the same column with no extra overhead. We show an example where a 5-bit NL ADC for the GELU function is programmed by choosing the best out of 4 possible NL-ADC characteristics(Fig. S12). The average INL reduces to -0.38 LSB from -1.14 LSB proving the efficacy of this method. More details of implementing non-monotonic functions such as GELU and Swish are provided in [Supplementary Note S12](#).

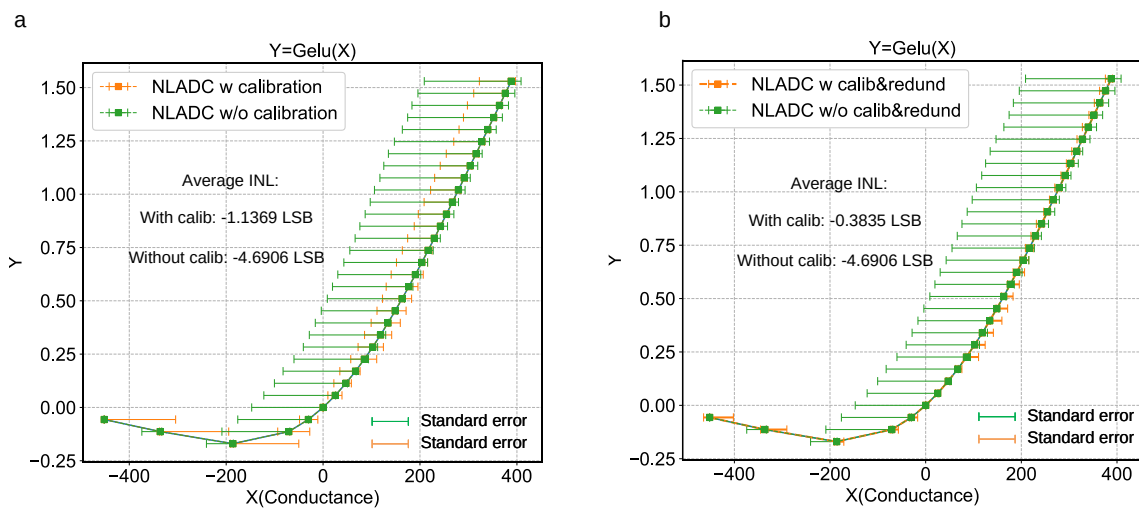


**Fig. S12. GELU function approximation by ramp ADC with/without redundancy method.** a GELU function approximation by ramp ADC without redundancy method. a GELU function approximation by ramp ADC with redundancy method. The average INL reduces to -0.38 LSB from -1.14 LSB.

## Supplementary Note S12. Non-monotonic nonlinear function approximation by ramp ADC

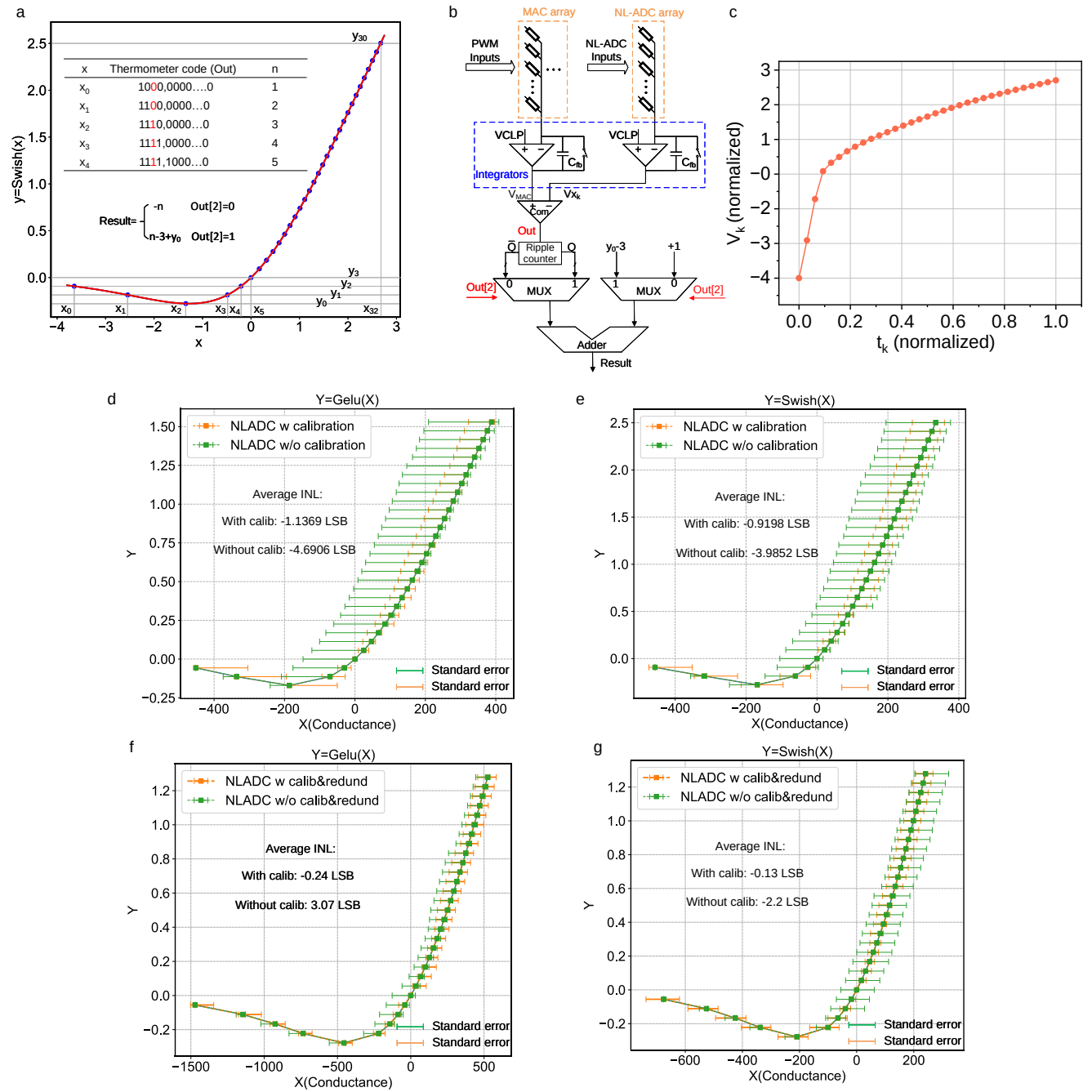


**Fig. S13. Non-monotonic nonlinear function approximation by ramp ADC** **a** Method of non-monotonic nonlinear function approximation by ramp ADC requires splitting the curve into monotonic sections and using different linear equations for each section. Which section is to be chosen depends on the bit corresponding to the minima (Out[2] in this case) and two separate equations are used for the final “result” depending on which section is selected. **b** Circuit diagram to implement the above method only requires the addition of one flip-flop to store the second output bit, one full-adder and two multiplexors. **c** The ramp voltage waveform for the Swish function obtained using the proposed method. **d** Swish function approximation by programmed ramp ADC. Average INL reduces to -1.1 LSB. **e** Gelu function approximation by programmed ramp ADC. Average INL reduces to 0.9 LSB. **f** Gelu function approximation by programmed ramp ADC with more points on negative side and redundancy method. Average INL reduces to -0.24 LSB. **g** Swish function approximation by programmed ramp ADC with more points on negative side and redundancy method. Average INL reduces to 0.13 LSB.

For non-monotonic functions, the inverse function does not exist and hence the method of using the ramp ADC to approximate the nonlinear function cannot be directly used. *However, the concept can still be applied by splitting the function into sub-parts where it is monotonic and using a logic to decide which sub-part has to be used.* Therefore, we directly divide the original function by selecting the extrema (maxima or minima) as the key points and then obtain the conductance values for each sub-part following the earlier technique. Taking the Swish function<sup>22</sup> (Fig. S13a) and 5-bit NL-ADC as an example, first the minima is identified as  $(x_2, y_0)$  and this is used to split the function into two parts—the left and right of the minima. As before, starting from the minimum point  $y_0$  of the function, the range of the function is divided into 30 equal intervals  $(y_0-y_{30})$ . The spacing between two consecutive ‘y’ values is the resolution of the NL-ADC. For  $y_1$  and  $y_2$ , each value corresponds to two x values ( $x_1$  and  $x_3$ ,  $x_0$  and  $x_4$ ), so 33 x values can be obtained. Using the formula  $\Delta X_k = X_{k+1} - X_k \quad k \in [0, 31]$ , 32  $\Delta X_k$  can be obtained. Then, according to the formula  $G_k = X_k 150 / \max(\Delta X_k) \quad k \in [0, 31]$ , 32  $G_k$  values can be obtained as conductance to be programmed into 32 RRAM cells.

After obtaining 32 conductance values, the other operations are the same as the previous monotonic function method. The control timing of the pulse is exactly the same as in Fig.S10. In this way, a ramp waveform starting from  $x_0$  and passing through  $x_1, x_2, \dots, x_{31}$  can be generated as shown in Fig. S13c. The output result is still the thermometer code that generates positive and negative values by comparing the MAC value and the step wave of  $x_0-x_{32}$ . However, the output has to be obtained by two different equations depending on which sub-part has to be chosen corresponding to the input x. This can be done easily based on the output bits of the thermometer code. In the example shown, the minima is at  $(x_2, y_0)$ ; hence, we have to use the left sub-part of the function if  $Out[2] = 0$  and the right sub-part otherwise. Here,  $Out[k]$  denotes the k-th bit of the thermometer code produced by the sense amplifier (SA). In the case of monotonic functions, these bits of thermometer code can be converted to a binary code (representing the decimal number  $n$ ) using a ripple counter (with bits denoted by  $Q[i]$ ) that counts the number of 1’s in the thermometer code. However, in this case of non-monotonic functions, two different equations have to be used for the two sub-parts to produce the final “result”. For the Swish and Gelu functions, it is given by:

$$result = \begin{cases} -n, & Out[2] = 0 \\ n + y_0 - 3, & Out[2] = 1 \end{cases} \quad (S6)$$

as also shown in Fig. S13a. In this case,  $y_0 = -3$  as seen in S13a.

The hardware circuit implementation for this is shown in Fig. S13b. The inputs to the comparator are the MAC values and the ramp waveform corresponding to  $x_0-x_{32}$  as earlier. Now, in addition we need a flip-flop to store  $Out[2]$  that determines which sub-part has to be selected as explained earlier. The output of this flip-flop controls two multiplexors (MUX) to create the final result according to equation S6. If  $Out[2] < 0$ , the result is given by  $result = -n$  represented by  $\bar{Q} + 1$  in two’s complement. On the other hand, if  $Out[2] = 1$ , the result is given by  $result = n - 3 + y_0$ . We use two MUXs and one adder to implement this simple comparison and addition as shown in Fig. S13b. The mathematical relationship can be summarized as follows:

$$result = \begin{cases} \bar{Q} + 1, & Out[2] = 0 \\ Q + y_0 - 3, & Out[2] = 1 \end{cases} \quad (S7)$$

For example, when MAC value is  $x_1$ ,  $n$  is equal to 2 as shown in the inserted table in Fig.S13a. In that case,  $Out[2] = 0$  and we can get the two results from left MUX and right MUX: 11101(i.e.,  $\bar{Q}$ ) and 1 respectively, obtaining the sum result of 11110 equivalent to  $-2$  in two’s complement. Similarly, when MAC value is  $x_4$ ,  $n$  is 5,  $Out[2] = 1$  and we can get the two results from left MUX and right MUX: 00101(i.e.,  $Q$ ) and  $y_0 - 3 = -6$  respectively. Then the sum result is  $5 + y_0 - 3 = -1$  as desired.

Two different commonly used non-monotonic activation functions of Gelu<sup>23</sup> and Swish<sup>22</sup> were programmed on the memristor array and the results are shown in S13d and S13e respectively. As done earlier in Fig. 3, 64 copies of the same function were programmed to see the variability and the one point calibration was used to reduce average INL to  $-1.1$  LSB for Gelu and  $-0.91$  LSB for Swish respectively.

We reprogrammed the memristors for both Gelu and Swish taking more number of sample points corresponding to negative outputs ( Fig. S13f and Fig. S13g). This programmability is an advantage of our memristive ADC over other ones with fixed reference. Combined with the earlier described redundancy methods, the achieved average INL are  $-0.24$  LSB and  $-0.13$  LSB respectively.

Finally, to assess whether our method of approximating the non-monotonic AF with uniform distribution of Y is effective in practical situations, two experiments are conducted using a Vision Transformer<sup>24</sup> (26 layers with 86M parameters) with the Gelu function on the CIFAR-100 dataset and a mixed-depth convolutional network<sup>25</sup> (124 layers with 2.6M parameters) with the Swish function on the CIFAR-10 dataset. Firstly, we train these two networks to obtain the software-level baseline accuracy: 92.2% and 93.7%. When implementing these models on hardware, the reference voltage range of ADC is limited, which leads to clipping in the MAC (Multiply-Accumulate) results. So the accuracy of Vision Transformer and mixed-depth convolutional

network degrade to 91.4% and 93.2%. In this case, we modified the training method to be: quantized activation functions are employed for forward propagation, while unquantized activation functions are used for backward propagation. As a reference, ReLU functions are also used to check if the quantized non-monotonic AF have any advantage over simpler but high precision AF. The networks utilizing 5-bit Gelu and Swish achieve 91.3% and 93.2% accuracy, respectively. This represents a reduction of only 0.9% and 0.5% compared to the SW baseline, while also outperforming the use of ReLU in place of the non-monotonic functions (accuracy when using ReLU was 91% and 91.65% respectively for these datasets). These results prove that 5-bit approximations of the AF incur very low loss in accuracy even for complicated networks such as vision transformers.

### Supplementary Note S13. Effect of long-term drift of the RRAM conductances

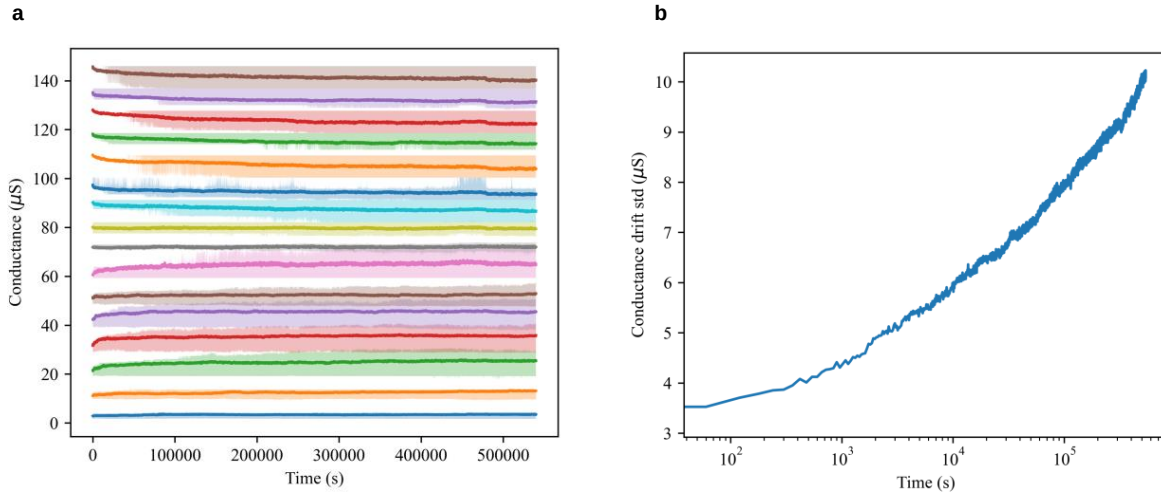


**Fig. S14. Long-term drift effect of the RRAM conductances.** **a** RRAM conductance change over time for 16 different initial values. These are taken as reference values for the later simulations on classification accuracy change with time. **b** Standard deviation of RRAM conductance change over time.

The initial step involves acquiring the drift data of RRAM conductance. The conductance range of our RRAM device spans from 0 to 150  $\mu\text{S}$ . This range is divided into 16 equidistant intervals, each with a resolution of 9.375  $\mu\text{S}$ . Consequently, 16 distinct RRAM conductance values can be obtained, such as 0  $\mu\text{S}$ , 9.375  $\mu\text{S}$ , 18.75  $\mu\text{S}$ , 28.125  $\mu\text{S}$ , and so on. Subsequently, a 64x64 RRAM array is partitioned into 16 sub-arrays, each measuring 16x16. Within each sub-array, the 256 RRAM cells are programmed with the same conductance value, selected from the aforementioned set of 16 values. Following the programming phase, the conductance value of each sub-array is measured every 60 seconds. The average and standard deviation of these conductance values are calculated and visualized through graphical representation, as depicted in Fig. S14. The total duration of the measurement spans 500,000 seconds.

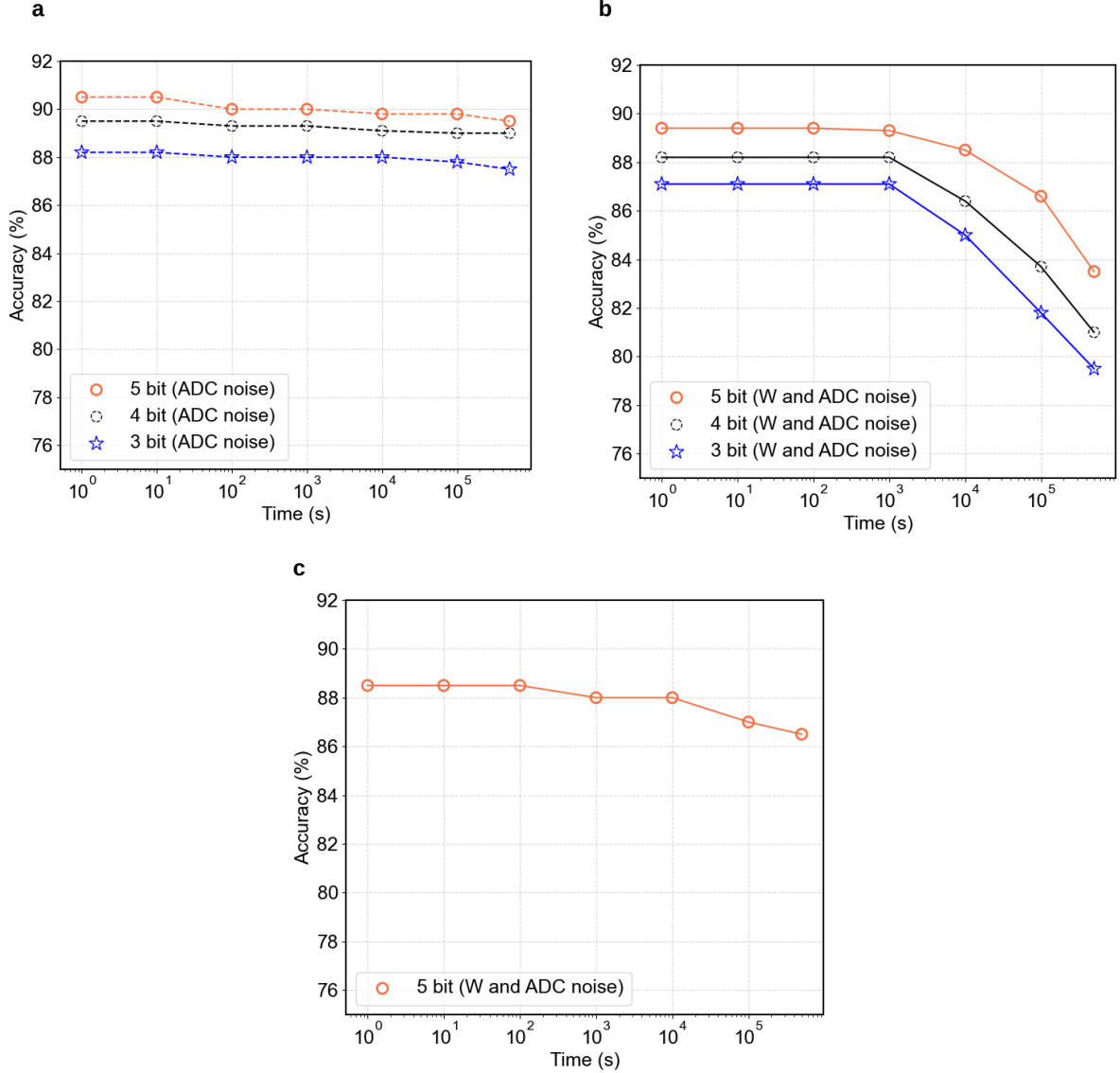


**Fig. S15. Test accuracy with drift effect of the RRAM conductances (KWS model)** **a** Drift noise is added only in NL-ADC module for different resolutions. This shows minimal drop in accuracy over the entire simulation duration. **b** Drift noise is added in both NL-ADC module (for different resolutions) and weight module. Accuracy starts degrading after  $\approx 1000$  seconds with a maximum degradation of  $\approx 6\%$  at  $5 \times 10^5$  seconds. **c** Modifying the training by adding a larger amount of noise ( $N(0, 8\mu S)$ ) during training, and then testing with added drift noise in both NL-ADC and weights show reduced drop in accuracy and stable performance over time.

During the inference stage, in addition to incorporating write noise  $N(0, 2.67/75)$  (Fig. S8c, where  $\gamma = 75$  represents the scaling factor from RRAM conductance to weight as described in Equation (7)) into the weights, we account for the impact of conductance drift using the weighted weight method. Initially, the 16 average RRAM values from Fig. S14a are divided by the scaling factor (75) and mapped to the weights, obtaining a set of 16 average weight curves depicting their temporal evolution ( $w_i$ , where  $i \in [0, 15]$ ). Subsequently, the drift effect of conductance is introduced to the weights using the following formula.

Using this data in Fig. S14, we simulate the neural network for the KWS task to check the degradation of performance over time. To do this simulation, all the conductances in the KWS task are written as weighted average of the two nearest conductance values among the initial 16 reference conductances in this plot. For example, we can write for the  $k$ -th conductance,

$$G_k = a * G_{\text{ref}, p}(0) + b * G_{\text{ref}, p+1}(0) \quad (\text{S8})$$

Where  $G_{\text{ref}, p}(0) < G_k < G_{\text{ref}, p+1}(0)$ , and  $G_{\text{ref}, p}(0)$  indicates the values of the p-th reference conductance at time  $t = 0$ .

$a = (G_{\text{ref}, p+1}(0) - G_k) / (G_{\text{ref}, p+1}(0) - G_{\text{ref}, p}(0))$  and  $b = (G_k - G_{\text{ref}, p}(0)) / (G_{\text{ref}, p+1}(0) - G_{\text{ref}, p}(0))$  are weighting coefficients. Then, the value of  $G_k$  at time  $t_k$  is obtained by the same weighted average of the drifted values of these reference conductances at time  $t_k$  as follows:  $G_k(t_k) = a * G_{\text{ref}, p}(t_k) + b * G_{\text{ref}, p+1}(t_k)$ .

Using this method, we show that if the RRAM drift affects the NL-ADC alone, the drop in accuracy is negligible ( $<1\%$ ) for the 5-bit ADC (Fig. S15a). However, if the drift affects both the weights and the NL-ADC, then the drop in accuracy starts increasing to 6% for the 5-bit ADC after 500,000 seconds (Fig. S15b). In our work, we show that modifying the training by adding a larger amount of noise during training, the drop in accuracy can be restricted to  $<2\%$  even at 500,000 seconds (Fig. S15c). This is much larger than the time needed for ADC operation during programming ( $\approx 3$  minutes) even when one single ADC is used for read operations during programming, as shown in Supplementary Note S5.. Hence, the programming can be finished much before conductance drift starts affecting results.

#### **Supplementary Note S14. Memristor programming circuits and overhead**

In our prototype system, the write operation of memristors is done by serially accessing one device at each time. Multiplexers at each row and column of the crossbar arrays are used to select one memristor device at each time. For the read process, a constant 0.2V voltage drop is applied on the memristor device. The current flowing through the device is collected by a transimpedance amplifier and converted to voltage signal, which is then digitized by a conventional ADC. For the writing process, a positive/negative voltage drop is applied on the memristor cell to SET/RESET the device. The pulse width of SET/RESET voltage is set to 20 ns. The amplitude of SET/RESET voltage and the voltage on the gate terminal of the transistor of the 1T1M cell (which is used for current compliance) are adaptively changed in the write-and-verify process. It is worth noting that the ADC/DAC needed in the read/write process can be shared across all the rows and columns of the array, causing very limited overhead. To accurately tune the conductance of the memristor to an arbitrary value, multiple iterations of write-and-verify might be needed, which consumes relatively long time (100 iterations is a conservative estimate<sup>11</sup>). However, thanks to the non-volatile property of memristor devices, the conductance tuning is a one-time overhead and does not influence the inference latency and throughput. Once the weights of the neural network and references of the ADCs are programmed to the memristor array, they can be retained for any later usage without the need of programming the memristors again. Nevertheless, we assume usage of one ADC per crossbar array for better scalability in programming and have included its overhead (area of  $\approx 280\mu m^2$ ) in area calculations at system level.

## References

1. H, L. & et al. Resistive RAM-centric computing: Design and modeling methodology. *IEEE Trans. on Circuits Syst. - I (TCAS-I)* 2263–2273 (2017).
2. Jiang, M., Shan, K., He, C. & Li, C. Efficient combinatorial optimization by quantum-inspired parallel annealing in analogue memristor crossbar. *Nat. Commun.* **14**, 5927 (2023).
3. Cai, F. & et. al. Power-efficient combinatorial optimization using intrinsic noise in memristor hopfield neural networks. *Nat. Electron.* **3**, 409–418 (2020).
4. Yu, C., Yoo, T., Chai, K. T. C., Kim, T. T.-H. & Kim, B. A 65-nm 8T SRAM compute-in-memory macro with column ADCs for processing neural networks. *IEEE J. Solid-State Circuits* **57**, 3466–3476 (2022).
5. Liu, Q. *et al.* A fully integrated analog ReRAM based 78.4 TOPS/W compute-in-memory chip with fully parallel MAC computing. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, 500–502 (IEEE, 2020).
6. Raut, G. & et. al. A CORDIC based Configurable Activation Function for ANN Applications. In *International Symposium on VLSI (ISVLSI)* (IEEE, 2020).
7. 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).
8. Ambrogio, S. *et al.* An analog-AI chip for energy-efficient speech recognition and transcription. *Nature* **620**, 768–775 (2023).
9. Chen, P.-Y., Peng, X. & Yu, S. Neurosim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning. *IEEE Transactions on Comput. Des. Integr. Circuits Syst.* **37**, 3067–3080 (2018).
10. “DNN+NeuroSim framework”. [https://github.com/neurosim/DNN\\_NeuroSim\\_V2.1](https://github.com/neurosim/DNN_NeuroSim_V2.1).
11. Esmanhotto, E. & et al. Experimental Demonstration of Multilevel Resistive Random Access Memory Programming for up to Two Months Stable Neural Networks Inference Accuracy. *Adv. Intell. Syst.* (2022).
12. Park, J. S. & et al. A 2.2mW 12-bit 200MS/s 28nm CMOS Pipelined SAR ADC with Dynamic Register-Based High-Speed SAR Logic. In *2020 IEEE Asian Solid-State Circuits Conference digest of technical papers (A-SSCC)* (IEEE, 2020).
13. Yan, B., Q., Y. & et. al. RRAM-based spiking nonvolatile computing-in-memory processing engine with precision-configurable in situ nonlinear activation. In *2019 Symposium on VLSI Technology (SOVC)*, T86–T87 (IEEE, 2019).
14. Yi, C., Wang, Z., Patil, A. & Basu, A. A 2.86-TOPS/W Current Mirror Cross-Bar Based Machine-Learning and Physical Unclonable Function Engine for Internet-of-Things Applications. *IEEE Trans. on CAS-I* **66**, 2240–52 (2018).
15. Giraldo, J. S. P. & Verhelst, M. Laika: A 5  $\mu$ W programmable LSTM accelerator for always-on keyword spotting in 65nm CMOS. In *ESSCIRC 2018-IEEE 44th European Solid State Circuits Conference (ESSCIRC)*, 166–169 (IEEE, 2018).
16. Chong, Y. & et. al. Efficient implementation of activation functions for LSTM accelerators. In *VLSI System on Chip (VLSI-SOC)* (IEEE, 2021).
17. Charan, G. & et al. Accurate Inference with Inaccurate RRAM Devices: Statistical Data, Model Transfer, and On-line Adaptation. In *2020 Design Automation Conference (DAC)* (IEEE, 2020).
18. W, L. & et al. A 40-nm MLC-RRAM Compute-in-Memory Macro With Sparsity Control, On-Chip Write-Verify, and Temperature-Independent ADC References. *IEEE J. Solid-State Circuits (JSSC)* 2868–77 (2022).
19. Jiang, H. & et al. A 40nm Analog-Input ADC-Free Compute-in-Memory RRAM Macrowith Pulse-Width Modulation between Sub-arrays. In *2022 Symposium on VLSI Circuits* (IEEE, 2022).
20. Rao, M. & et al. Thousands of conductance levels in memristors integrated on CMOS. *Nature* (2023).
21. Song, W. & et al. Programming memristor arrays with arbitrarily high precision for analog computing. *Science* (2024).
22. R., P. & et. al. SWISH: A SELF-GATED ACTIVATION FUNCTION. *arXiv preprint arXiv:1710.05941* (2017).
23. Hendrycks, D. & et. al. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415* (2016).
24. Dosovitskiy, A. & et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).
25. Tan, M. & Le, Q. V. Mixed depthwise convolutional kernels. *arXiv preprint arXiv:1907.09595* (2019).