







Efficient nonlinear function approximation in analog resistive crossbars for recurrent neural networks

Received: 8 February 2024

Accepted: 13 January 2025

Published online: 29 January 2025

 Check for updates

Junyi Yang ^{1,4}, Ruibin Mao ^{2,4}, Mingrui Jiang ², Yichuan Cheng¹, Pao-Sheng Vincent Sun¹, Shuai Dong¹, Giacomo Pedretti ³, Xia Sheng³, Jim Ignowski ³, Haoliang Li¹, Can Li ²✉ & Arindam Basu ¹✉

Analog In-memory Computing (IMC) has demonstrated energy-efficient and low latency implementation of convolution and fully-connected layers in deep neural networks (DNN) by using physics for computing in parallel resistive memory arrays. However, recurrent neural networks (RNN) that are widely used for speech-recognition and natural language processing have tasted limited success with this approach. This can be attributed to the significant time and energy penalties incurred in implementing nonlinear activation functions that are abundant in such models. In this work, we experimentally demonstrate the implementation of a non-linear activation function integrated with a ramp analog-to-digital conversion (ADC) at the periphery of the memory to improve in-memory implementation of RNNs. Our approach uses an extra column of memristors to produce an appropriately pre-distorted ramp voltage such that the comparator output directly approximates the desired nonlinear function. We experimentally demonstrate programming different nonlinear functions using a memristive array and simulate its incorporation in RNNs to solve keyword spotting and language modelling tasks. Compared to other approaches, we demonstrate manifold increase in area-efficiency, energy-efficiency and throughput due to the in-memory, programmable ramp generator that removes digital processing overhead.

Artificial Intelligence algorithms, spurred by the growth of deep neural networks (DNN), have produced the state-of-the-art solutions in several domains ranging from computer vision¹, speech recognition², game playing³ to scientific discovery⁴, natural language processing (NLP)⁵, and more. The general trend in all these applications has been increasing the model size by increasing the number of layers and the number of weights in each layer. This trend has, however, caused growing concern in terms of energy efficiency for both edge applications and servers for training; power is scarce due to battery limits in edge devices while the total energy required for training large models

in the cloud raises environmental concerns. Edge devices have a further challenge posed by strong latency requirements in applications such as keyword spotting to turn on mobile devices, augmented reality and virtual reality platforms, anti-collision systems in driverless vehicles etc.

The bottleneck for implementing DNNs on current hardware arises due to the frequent memory access necessitated by the von Neumann architecture and the high memory access energy for storing the parameters of a large model⁶. As a solution to this problem, a new architecture of In-memory Computing (IMC) has become increasingly

¹Department of Electrical Engineering, City University of Hong Kong, Hong Kong SAR, China. ²Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong SAR, China. ³Hewlett Packard Labs, Hewlett Packard Enterprise, Milpitas, CA, USA. ⁴These authors contributed equally: Junyi Yang, Ruibin Mao. ✉ e-mail: canli@hku.hk; arinbasu@cityu.edu.hk

popular. Instead of reading and writing data from memory in every cycle, IMC allows neuronal weights to remain stationary in memory with inputs being applied to it in parallel and the final output prior to neuronal activation being directly read from memory. Among the IMC techniques explored, analog/mixed-signal IMC using non-volatile memory devices such as memristive⁷ ones have shown great promise in improving latency, energy and area efficiencies of DNN training^{8–10} and inference^{11–13}, combinatorial optimization^{14,15}, hardware security^{11,16}, content addressable memory¹⁷, signal processing^{18–20} etc. It should be noted that analog IMC does not refer to the input and output signals being analog; rather, it refers to the storage of multi-bit or analog weights in each memory cell (as opposed to using memristor for 1-bit storage^{21,22}) and using analog computing techniques (such as Ohm's law, Kirchoff's law etc.) for processing inputs. Analog weight storage²³ enables higher density of weights as well as higher parallelism (by enabling multiple rows simultaneously) compared to digital counterparts.

Comparing the energy efficiency and throughput of recently reported DNN accelerators (Fig. 1a) shows the improvements provided by IMC approaches over digital architectures. However, taking a closer look based on DNN architecture exposes an interesting phenomenon—while analog IMC has improved energy efficiency of convolutional and fully connected (FC) layers in DNNs, the same cannot be said for recurrent neural network (RNN) implementations such as long short term memory (LSTM)^{24–28}. Resistive memories store the layer weights in their resistance

values, inputs are typically provided as pulse widths or voltage levels, multiplications between input and weight happen in place by Ohm's law, and summation of the resulting current occurs naturally by Kirchoff's current law. This enables an efficient implementation of linear operations in vector spaces such as dot products between inputs and weight vectors. Early implementations of LSTM using memristors have focussed on achieving acceptable accuracy in network output in the presence of programming errors. A 128×64 1T1R array²⁹ was shown to be able to solve real-life regression and classification problems while 2.5 M phase-change memory devices³⁰ have been programmed to implement LSTM for language modeling. While these were impressive demonstrations, energy efficiency improvements were limited, since RNNs such as LSTMs have a large fraction of nonlinear (NL) operations such as sigmoid and hyperbolic tangent being applied as neuronal activations (Fig. 1b). With the dot products being very efficiently implemented in the analog IMC, the conventional digital implementation of the NL operations now serves as a critical bottleneck.

As an example, an RNN transducer was implemented on a 34-tile IMC system with 35 million phase-change memory (PCM) devices and efficient inter-tile communication³¹. While the system integration and scale of this effort³¹ is very impressive, the NL operations are performed off-chip using low energy efficiency digital processing reducing the overall system energy efficiency. Another pioneering research²³ integrated 64 cores of PCM arrays for IMC operations with on-chip digital processing units for NL operations. However, the serial

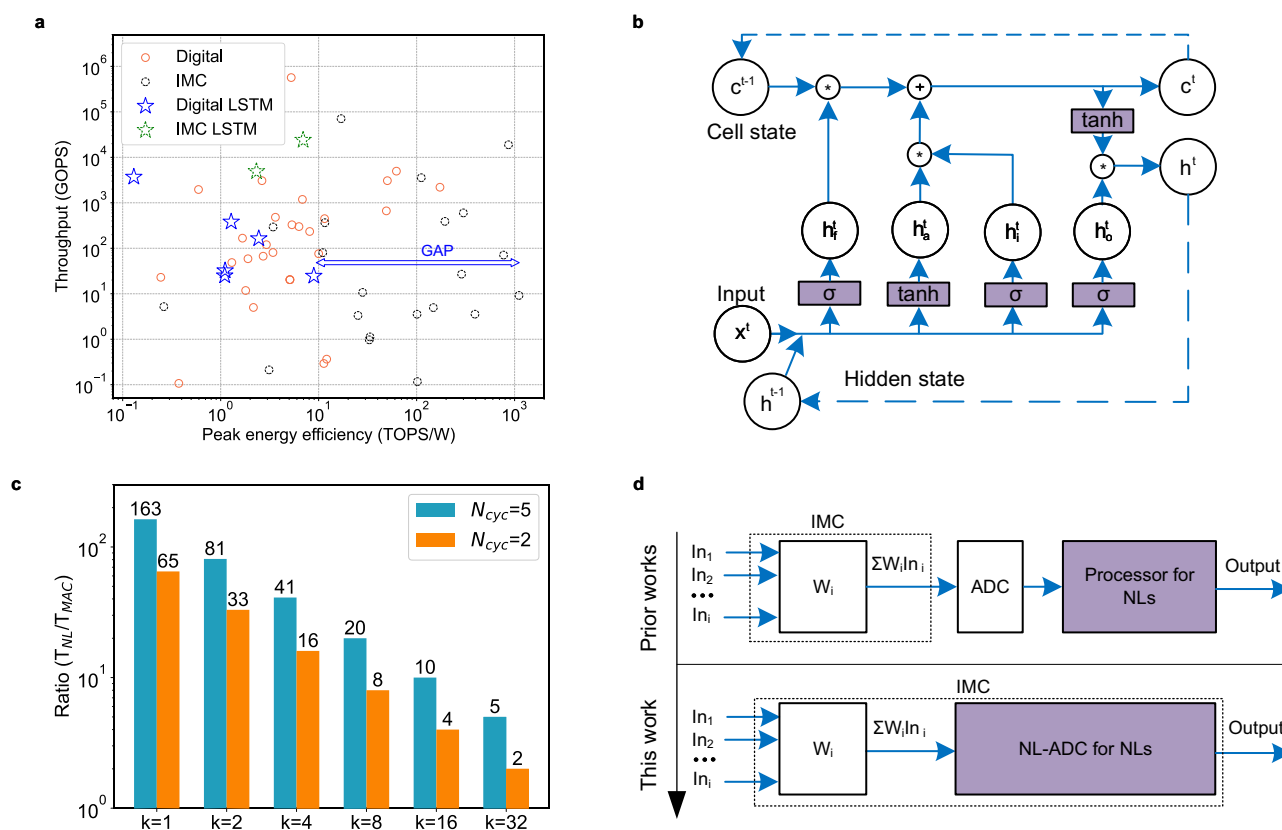


Fig. 1 | Limitation of current In-memory computing (IMC) for Recurrent Neural Networks and our proposed solution. **a** A survey of DNN accelerators show the improvement in energy efficiency offered by IMC over digital architectures. However, the improvement does not extend to recurrent neural networks (RNN) such as LSTM and there exists a gap in energy efficiency between RNNs and feedforward architectures. Details of the surveyed papers available here⁶⁶. **b** Architecture of a LSTM cell showing a large number of nonlinear (NL) activations such as sigmoid and hyperbolic tangent which are absent in feedforward architectures that mostly use simple nonlinearities like rectified linear unit (ReLU). **c** Digital implementation

of the NL operations causes a bottleneck in latency and energy efficiency since the linear operations are highly efficient in time and energy usage due to inherent parallelism of IMC. For a LSTM layer with 512 hidden unit and with $k = 32$ parallel digital processors for the NL operations, the NL operations still take 2–5X longer time for execution due to the need of multiple clock cycles (N_{cyc}) per NL activation. **d** Our proposed solution creates an In-memory analog to digital converter (ADC) that combines NL activation with digitization of the dot product between input and weight vectors.

nature of the digital processor, which is shared across the neurons in 8 cores, reduced both the energy efficiency and throughput of the overall system. This work used look-up tables (LUT), similar to other works^{32–34}; alternate techniques using cordic³⁵ or piece wise linear approximations^{24,36–38} or quadratic polynomial approximation³⁹ have also been proposed to reduce overhead and latency (N_{cyc}) of computing one function. However, it is the big difference in parallelism of crossbars versus serial digital processors which causes this inherent bottleneck. Even in a hypothetical situation with an increased number of parallel digital computing engines for the NL activations (Fig. 1c, Supplementary Note S6), albeit at a large area penalty, the latency of the NL operations still dominates the overall latency due to the extremely fast implementation of vector-matrix-multiplication (VMM) in memristive crossbars.

In this work, we introduce an in-memory analog to digital conversion (ADC) technique for analog IMC systems that can combine nonlinear activation function computations in the data conversion process (Fig. 1d), and experimentally demonstrate its benefits in an analog memristor array. Utilizing the sense amplifiers (SA) as a comparator in ramp ADC, and creating a ramp voltage by integrating the current from an independent column of memristors which are activated row by row in separate clock cycles, an area-efficient in-memory ADC for memristive crossbars is demonstrated. However, instead of generating a linear ramp voltage as in conventional ADCs, we generated a nonlinear ramp voltage by appropriately choosing different values of memristive conductances such that the shape of the ramp waveform matches that of the inverse of the desired NL activation function. Using this method, we demonstrate energy-efficient 5-bit implementations of commonly used NL functions such as sigmoid, hyperbolic tangent, softsign, softplus, elu, selu etc. A one-point calibration scheme is shown to reduce the integral nonlinearity (INL) from 0.948 to 0.886 LSB for various NL functions. Usage of the same IMC cells for ADC and dot-product also gives added robustness to read voltage variations, reducing INL to ≈ 0.04 LSB compared to ≈ 5.0 LSB for conventional methods. Using this approach combined with hardware aware training⁴⁰, we experimentally demonstrate a keyword spotting (KWS) task on the Google speech commands dataset (GSCD)⁴¹. With a 32 hidden neuron LSTM layer (having 128 nonlinear gating functions) that uses 9216 memristors from the $3 \times 64 \times 64$ memristor array on our chip, we achieve 88.5% accuracy using a 5-bit NL-ADC with a $\approx 9.9\times$ and $\approx 4.5\times$ improvement of area and energy efficiencies respectively at the system level for the LSTM layer over previous reports. Moreover, compared to a conventional approach using the exact same configuration (input and output bit-widths) as ours, the estimated area and energy efficiency advantages are still retained at $\approx 6.2\times$ and $\approx 1.46\times$ respectively for system level of evaluation. Finally, we demonstrate the scalability of our system by performing a character prediction task on the Penn Treebank dataset⁴² using a LSTM model $\approx 100\times$ bigger than the one for KWS using experimentally validated nonideality models and achieving software equivalent accuracy. The improvements in area efficiency are estimated to be $6.6\times$ over a conventional approach baseline and $125\times$ over earlier work³¹ at the system level, with the drastic increase in performance due to the much higher number of nonlinear functions in the larger model.

Results

Nonlinear function approximation by ramp ADC

A conventional ramp ADC operates on an input voltage $V_{\text{in}} \in \{V_{\text{min}}, V_{\text{max}}\}$ and produces a binary voltage V_{out} whose time of transition from low to high, $t_{\text{in}} \in \{0, T_s\}$, encodes the value of the input. As shown in Fig. 2a, V_{out} is produced by a comparator whose positive input is connected to a time-varying ramp signal, $V_{\text{ramp}}(t) = f(t)$ and the negative input is connected to V_{in} . For the conventional case of linearly increasing ramp voltage, and denoting the comparator's operation by

a Heaviside function Θ , we can mathematically express V_{out} as:

$$V_{\text{out}} = \Theta(V_{\text{ramp}}(t) - V_{\text{in}}) = \Theta(f(t) - V_{\text{in}}) = \Theta(Kt - V_{\text{in}}) \quad (1)$$

The threshold crossing time t_{in} can be obtained as $t_{\text{in}} = f^{-1}(V_{\text{in}}) = \frac{1}{K} V_{\text{in}}$ by setting Equation (1) equal to zero and solving for 't'. The pulse width information in t_{in} may be directly passed to the next layers as pulse-width modulated input³¹ or can be converted to a digital code using a time-to-digital converter (TDC)⁴³. Now, suppose we want to encode the value of a nonlinear function of V_{in} , denoted by $g(V_{\text{in}})$ in t_{in} . Comparing with the earlier equation of t_{in} , we can conclude this is possible if:

$$t_{\text{in}} = f^{-1}(V_{\text{in}}) = g(V_{\text{in}}) \Rightarrow f() = g^{-1}() \quad (2)$$

where we assume that the desired nonlinear function $g()$ is bijective and an inverse exists in the defined domain of the function. Supplementary Note S1 shows the required ramp function for six different nonlinear activations. For the case of non-monotonic functions, this method can still be applied by splitting the function into sections where the function is monotonic. Examples of such cases are shown in Supplementary Note S12 for two common non-monotonic activations—Gelu and Swish.

In practical situations, the ramp function is a discrete approximation to the continuous function $f(t)$ mentioned earlier. For a b -bit ADC, the domain of the function $V_{\text{ramp}} = f(t)$ is split into $P = 2^b$ segments using $P+1$ points t_k such that $t_0 = 0$, $t_k = \frac{kT_s}{P}$ and $t_P = T_s$. The initial voltage of the ramp, $f(0) = V_0 = V_{\text{init}}$, defines the starting point while the other voltages V_k ($k = 1$ to P) can be obtained recursively as follows:

$$V_k = V_{k-1} + \Delta V_k = V_{k-1} + (g^{-1}(t_k) - g^{-1}(t_{k-1})) \quad (3)$$

Here, V_{init} may be selected appropriately to maximize the dynamic range of the function represented by the limited b -bits. Supplementary Table S2 demonstrates the choice of $(t_k, \Delta V_k)$ tuples for six different nonlinear functions commonly used in neural networks.

In-memory Implementation of nonlinear ADC and vector-matrix multiply in a crossbar array

Different from traditional IMC systems where the ADCs and nonlinear functions calculation are separated from the memory core, our proposed hardware implements the ADC with nonlinear calculation ability inside the memory along with the computation part. The nonlinear function approximating ADC described earlier is implemented using memristors with the following unique features:

1. We utilize the memristors to generate the ramp voltages directly within the memory array which incurs very low area overhead with high flexibility.
2. By leveraging the multi-level state of memristors, we can generate the nonlinear ramp voltage according to the x-y relationship of the nonlinear function as described in Equation (3).

Take the sigmoid function, ($y = g(x) = 1/(1 + e^{-x})$) as an example. To extract the step of the generated ramp voltages, we first take the inverse of the sigmoid function ($x = g^{-1}(y) = \ln \frac{y}{1-y}$) as shown in Fig. 2c. This function is exactly the ramp function that needs to be generated during the conversion process, as described in detail in the earlier section. The figure shows the choice of 32 (x,y) tuples for 5-bit nonlinear conversion. The voltage difference between successive points (ΔV_k in Equation (3)) is shown in Fig. 2d highlighting the unequal step sizes. Memristors proportional to these step values need to be programmed in order to generate the ramp voltage, as described next. We show our proposed in-memory nonlinear ADC circuits in Fig. 2b. In each memory core, only a single column of memristors will be utilized to generate $V_{\text{ramp}}(t) = f(t)$ with very low hardware cost as shown in Fig. 2e. The memory is separated into N columns for the multiplication-

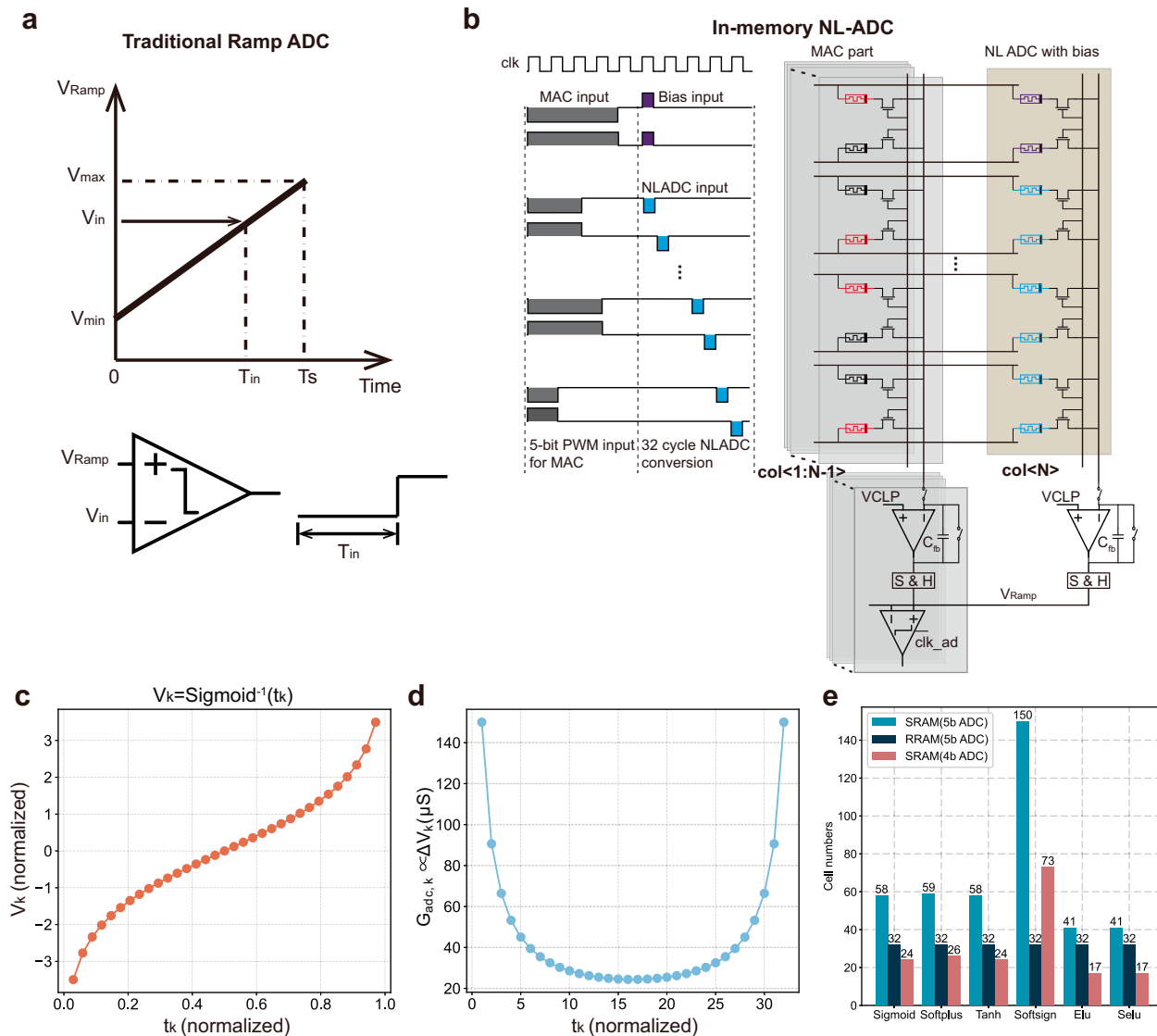


Fig. 2 | Overview of in-memory nonlinear ADC. **a** The concept of traditional ramp-based ADC. **b** The schematic and timing of in-memory computing circuits with embedded nonlinear activation function generation. **c** The Inverse of the sigmoid function illustrates the shape of the required ramp voltage. **d** The value of each step of the ramp voltage V_{ramp} denoted by ΔV_k is proportional to memristor conductances $G_{adc,k}$ used to program the nonlinear ramp voltage. The desired

conductances for a 5-bit implementation of a sigmoid nonlinear activation is shown. **e** Comparison of used cell numbers between 5-bit and 4-bit in-SRAM with 5-bit in-RRAM nonlinear function. The RRAM-based nonlinear function has an approximation error between the two SRAM-based ones due to write noise while using a smaller area due to its compact size.

and-accumulation (MAC) part and one column for the nonlinear ADC (NL-ADC) part. For the MAC part, the inputs are quantized to b -bit ($b = 3-5$ in experiments) if necessary and transferred into pulse width modulation (PWM) signals sent to the SL of the array. To adapt to the positive and negative weights or inputs in the neural networks, we encode one weight or input into differential 1T1R pairs and inputs shown in Supplementary Fig. S9. We propose a charge-based approach for sensing the MAC results where the feedback capacitor of an integrator circuit is used to store the charge accumulated on the BL. Denoting the feedback capacitor by C_{fb} , the voltage on the sample and hold (S&H) for the k -th column is:

$$V_{mac,k} = V_{CLP} + \frac{1}{C_{fb}} \sum_i V_{read} G_{ik} T_{in,i}$$

where V_{CLP} is the clamping voltage on the bitline enforced by the integrator, $T_{in,i}$ denotes the pulse width encoding the i -th input and G_{ik} is the memristive conductance on the i -th row and k -th column.

In the column of the NL-ADC part, we have two sets of memristors. One is the memristors for generating the initial bias voltage (V_{init} in the earlier section), and another set of memristors is for generating the nonlinear ramp voltages. The addition of bias memristors is used to set the initial ramp value as well as calibrate the result after programming the NL-ADC memristors due to programming not being accurate. The ‘one point’ calibration will move the ramp function generated by the NL-ADC column to intersect the desired, theoretical ramp function at the zero-point which leads to minimal error. This is described in detail in the next sub-section. As shown in the timing diagram in Fig. 2b, after the MAC results are latched on the S&H, the positive input with one cycle pulse width is first sent into NL-ADC column to generate a bias voltage (corresponding to the most negative voltage at the starting point V_{init}) for the ramp function on V_{ramp} . Then for each clock cycle, negative input is sent to the SL of NL-ADC memristors to generate the ramp voltages on V_{ramp} . Since the direction of each step voltage is known, only one memristor is used corresponding to the magnitude of the step while the polarity of the input sets the direction of the ramp

voltage. Using less memristors for every step provides the flexibility to use more devices for calibration or error correction if stuck devices are found. The ramp voltage generated at the q -th clock cycle, V_{ramp}^q , is given by the following equation:

$$V_{ramp}^q = V_{CLP} + V_{init} + \frac{1}{C_{fb}} \sum_{i=1}^q V_{read} G_{adc,i} T_{adc}$$

where T_{adc} are the pulse width of ADC read pulses and $\frac{V_{read} G_{adc,k} T_{adc}}{C_{fb}}$ equals ΔV_k from Equation (3). An example of the temporal evolution of the ramp voltage is shown in the Supplementary Note S9. The comparison between the V_{ramp} and MAC result is done by enabling the comparator using the clk_{ad} signal after each step of the generation of ramp voltages. The conversion continues for $P=2^b$ cycles producing a thermometer code or pulse width proportional to the nonlinear activation applied to the MAC result. The pulse-width can be directly transferred to other layers as the input as done in other works³¹; else, the thermometer code can be converted to binary code using a ripple counter²³.

To prove the effectiveness of this approach, we simulate it at the circuit level and the results are shown in Fig. S2. The fabricated chip does not include the integrators and the comparator which are implemented in software after obtaining the crossbar output. In-memory ramp ADC using SRAM has been demonstrated earlier⁴⁴ in a different architecture with a much larger (100%) overhead compared to the memory used for MAC operations; however, nonlinear functions have not been integrated with the ADC. Implementing the proposed scheme using SRAM would require many cells for each step due to the different step sizes. Since an SRAM cell can only generate two step sizes (+1 or -1) intrinsically, other step sizes need to be quantized and represented in terms of this unit step size (or LSB). Denoting this unit step by $\min(\Delta V_k)$, $\text{round}(\frac{\Delta V_k}{\min(\Delta V_k)})$ is the number of SRAM cells needed for a single step while the total number of steps needed is given by $\sum_{k=1}^{2^b} \text{round}(\frac{\Delta V_k}{\min(\Delta V_k)})$. Figure 2e illustrates this for several common nonlinear functions. However, thanks to the analog tunability of the conductance of memristors, we can encode each step of the ramp function ΔV_k into only one memristor $G_{adc,k}$ which leads to the usage of much lower number of bitcells (1.28X–4.68X) compared with nonlinear SRAM-based ramp function generation (Fig. 2e) for the 5-bit case. However, the write noise in memristors can result in higher approximation error than the SRAM version. Using Monte-Carlo simulations for the SRAM case, we estimate the mean squared error (MSE) for a memristive 5-bit version is generally in between that of the 5-bit and 4-bit SRAM versions (e.g., MSE for sigmoid nonlinearity is ≈ 0.0008 and ≈ 0.0017 respectively for the 5-bit and 4-bit SRAM versions, while that of the memristive 5-bit version is ≈ 0.0014). Hence, we also plot in Fig. 2e the number of SRAM bitcells needed in 4-bit versions as well. However, combined with the inherently smaller ($>2X$) bitcell size of memristors compared to SRAM, our proposed approach still leads to more compact implementations with very little overhead due to the ramp generator. The approximation accuracy for the memristive implementation is expected to improve in future with better devices⁴⁵ and programming methods⁴⁶. Details of the number of SRAM cells needed for six different nonlinear ramp functions are shown in Supplementary Table S2. The topology shown here is restricted to handle inputs limited to the dimension of the crossbar. We show in Supplementary Note S10 how it can be extended to handle input vectors with dimensions larger than the number of rows in the crossbar by using the integrator to store partial sums and splitting the weight across multiple columns.

Calibration procedures for accurate programming of nonlinear functions in crossbars

Mapping the NL-ADC into memristors has many challenges including device-to-device variations, programming errors, etc such that the

programmed conductance $G'_{adc,k}$ will deviate from the desired $G_{adc,k}$. To tackle these problems, we introduce the adaptive mapping method to calibrate the programming errors. For each nonlinear function, we first extract the steps ΔV_k of the function as illustrated in Fig. 2d. Then we normalize them and map them to the conductances with a maximum conductance of 150 μS . Detailed characterization of our memristive devices were presented earlier⁴⁷. We program the conductances using the iterative-write-and-verify method explained in methods. However, the programming will introduce errors, and hence, we reuse the N_{cali} ($=5$ in experiments) bias memristors to calibrate the programming error according to the mapped result. These memristors are also used to create the starting point of the ramp V_{init} , which is negative in general (assuming domain of $g()$ spans negative and positive values). Hence, positive voltage pulses are applied to these bias/calibration conductances while negative ones are applied to the $G_{adc,k}$ for the ramp voltage as shown in Fig. 2a. Based on this, the one-point calibration strategy is to match the zero crossing point of the implemented ramp voltage with the desired theoretical one by changing the value of V_{init} to $V_{init,new}$. The total calibration conductance is first found as:

$$G_{cali,tot} = \sum_{i=1}^m G'_{adc,i} \quad \text{where } m \in \mathbb{N} \text{ s.t. } V_m = 0$$

Here, m is the index where the function $g() = f^{-1}()$ crosses the zero point of the x -axis. We then represent this calibration term using only a few memristors with the largest conductance on the chip ($G_{max} = 150 \mu S$). The number of memristors used for calibration is $N_{cali} = \lceil G_{cali,tot} / 150 \mu S \rceil + 1$ in which $N_{cali} - 1$ memristors are 150 μS and the one left is $G_{cali} \bmod 150 \mu S$. More details about the calibration process including timing diagrams are provided in the Supplementary Note S9.

To demonstrate the effectiveness of our approach, we experimentally programmed two frequently used nonlinear functions in neural network models: sigmoid function and tanh function, with the calibration methods mentioned above, as shown Fig. 3a. We program 64 columns containing the same NL-ADC weights and calibration terms in one block of our memristor chip and set the read voltage to 0.2 V. In the left panel, we compare the transfer function with 3 different cases: (1) Ideal nonlinear function (2) Transfer function without on-chip calibration (3) Transfer function with on-chip calibration. We can find that the curve with calibration matches well with the ideal curve while the one without calibration has some deviation. We also show the programmed conductance map in the right panel for a block of 8 arbitrarily selected columns to showcase different types of programming errors that can be corrected. The first 32 rows are the NL-ADC weights representing 5-bit resolution. The remaining 5 rows are the calibration memristors after reading out the NL-ADC weights to mitigate the program error. In the conductance map on the right, we can also find some devices which are either stuck at OFF or have higher programming error. Fortunately, these errors happening during the NL-ADC weights programming can be compensated by the calibration part as evidenced in the reduced values of average INL. In addition to these two functions, we also show other nonlinear functions with in-memory NL-ADC in Supplementary Fig. S7 which covers nearly all activation functions used in neural networks. Further reductions in the INL may be achieved through redundancy. Briefly, the entire column used to generate the ramp has many unused memristors which may be used to program redundant copies of the activation function. The best out of these may be chosen to further reduce the effect of programming errors. Details of this method along with measured results from two examples of nonlinear activations are shown in Supplementary Note S11.

In a real IMC system, the voltage variations on-chip could harm the performance. This can be a challenge when using traditional ADC due to the lack of ability to track the voltage variation. For example, if

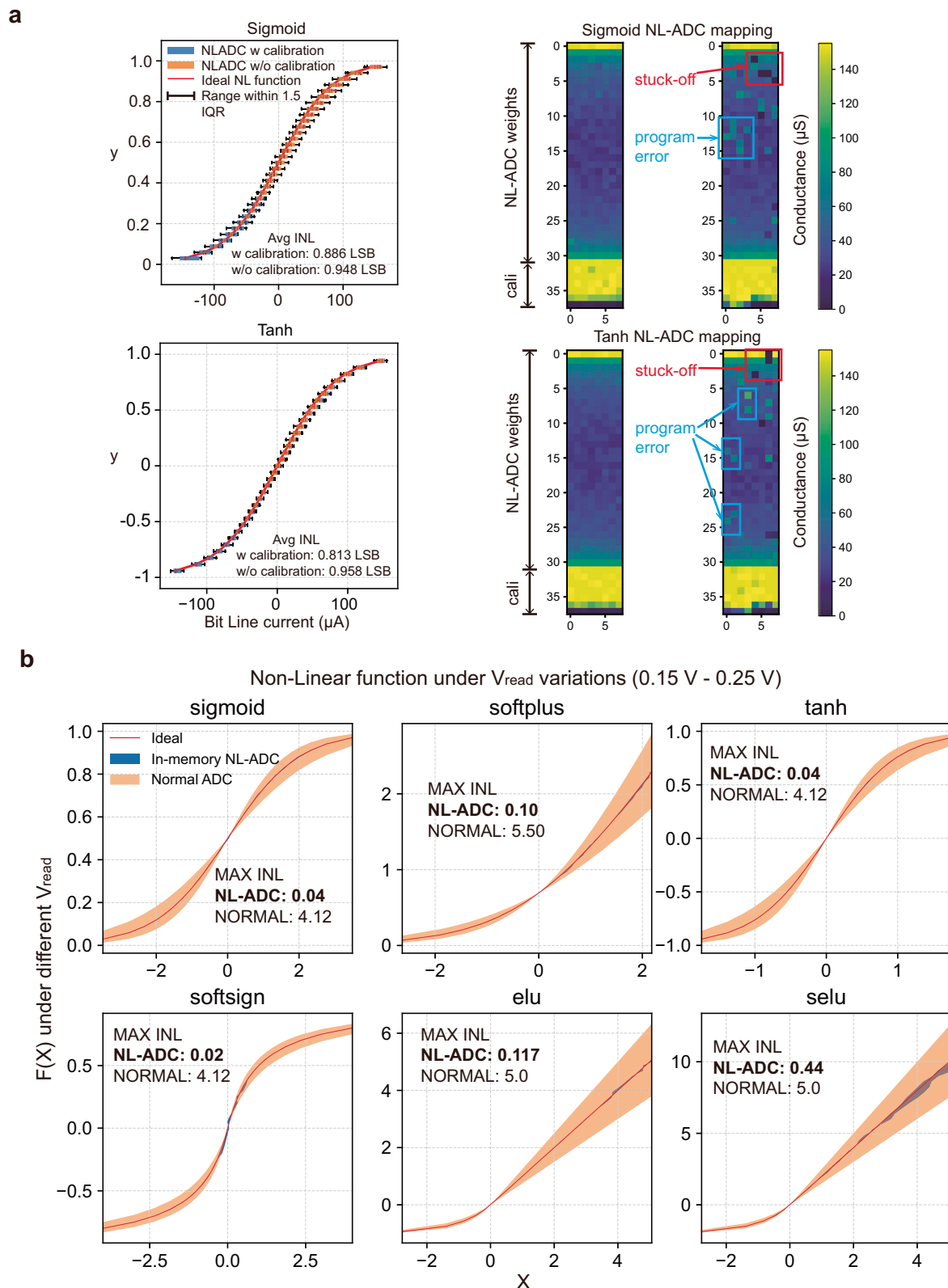


Fig. 3 | Experimentally demonstrated NL-ADC on crossbar arrays. a Calibration process for accurate NL-ADC programming. The left panel shows the ramp function of the ideal case, programming without bias calibration and with bias calibration. The case with bias calibration shows better INL performance. The right panel shows the actual conductance mapping on the crossbar arrays on two blocks of 8 arbitrary selected columns. The lower 5 conductances are for bias calibration while the top 32 are for the ramp generation. We show the cases when mapping of NL-ADC

weights doesn't have stuck-at-OFF devices and low programming error (left block), and the cases which have stuck-at-OFF devices and high programming error (right block). The results show that both cases can be calibrated by the additional 5 memristors. **b** Robustness of our proposed in-memory NL-ADC under V_{read} variations. We sweep the V_{read} from 0.15 V to 0.25 V to simulate noise induced variations in read voltage. Normal ADC has large variations while our in-memory NL-ADC can track the V_{read} .

the voltage we set for VMM is 0.2 V but the actual voltage sent to SL is 0.25 V (due to supply noise, voltage drop etc.), the final result read out from the ADC will deviate a lot from the real VMM result. Fortunately, our in-memory NL-ADC has the natural ability to track the voltage variations on-chip since the LSB of the ADC is generated using circuits matching those generating the MAC result. Hence, the read voltage is canceled out during the conversion and only the relative values of MAC and ADC conductances will affect the final result. To demonstrate that our proposed in-memory NL-ADC is robust under different read voltages, we run the experiment by setting different read voltages from 0.15 V to 0.25 V and measure the transfer function. From the result shown in Fig. 3b, we can see that the conventional ADC has large variations (maximum INL of 4.12–5.5 LSB) due to V_{read} variations while our in-memory NL-ADC only experiences a little effect (maximum INL of 0.02–0.44 LSB).

Long-short-term memory experiment for keyword spotting implemented in memristor hardware

After verifying the performance of the nonlinear activations on chip, we proceed to assess the inference accuracy obtained when executing neural networks using NL-ADC model on chip. The GSCD⁴¹, a common benchmark for low-power RNNs^{48,49}, is used to train and test for 12-class KWS task. The KWS training network consists of Mel-frequency cepstral coefficient (MFCC), standardization, LSTM layer (9216 weight parameters in a 72×128 crossbar) and FC layer; detailed parameters pertaining to the task are provided in “Methods”. Further training with weight noise-aware and NL-ADC noise-aware (Methods) is implemented.

On-chip inference shown in Fig. 4a is performed after training. After feature extraction (MFCC extraction and normalization), a single input audio signal is divided into 49 arrays, each with a feature length of 40. These extracted features, along with the previous 32-dimensional output vector h^t from the LSTM, are then sent to the memristor crossbar. The architecture of LSTM layer, along with the direction of data flow, is depicted in Fig. 4b. The equations of the LSTM layer are given by Equation (4) and Equation (5) as below:

$$\begin{bmatrix} h_f^t \\ h_a^t \\ h_i^t \\ h_o^t \end{bmatrix} = \begin{bmatrix} \sigma \\ \tanh \\ \sigma \\ \sigma \end{bmatrix} \begin{bmatrix} x^t & h^{t-1} \end{bmatrix} \begin{bmatrix} W \\ U \end{bmatrix} \quad (4)$$

$$\begin{aligned} h_c^t &= h_f^t \odot h_c^{t-1} + h_i^t \odot h_a^t \\ h^t &= h_o^t \odot \tanh(h_c^t) \end{aligned} \quad (5)$$

where x^t represents the input vector at the current step, h^t and h^{t-1} denote hidden state vector also known as output vector of the LSTM cell at the current and previous time steps, and h_c^t represents cell state vector at the current time step. The model parameters, including the weights W and recurrent weights U , are stored for h_f^t, h_i^t, h_o^t and h_a^t (forget gate, input/update gate, output gate, cell input, respectively). \odot denotes element-wise multiplication and σ is the sigmoid function. The parameters (W and U) and functions (σ and \tanh) in Equation (4) are programmed in the memristor crossbar (Methods) and the conductance difference map of the memristor crossbar is depicted in Fig. 4b. Therefore, all MAC and nonlinear operations specified in Equation (4) are executed on chip removing the need to transfer weights back and forth. The four vectors outputted by the chip ($h_f^t, h_a^t, h_i^t, h_o^t$) are all digital, allowing them to be directly read by the off-chip processor without requiring an additional ADC. This approach provides notable benefits, such as a substantial reduction in the latency due to nonlinearity calculation in a digital processor and decreased energy consumption.

Figure 4d depicts inference accuracy results for 12-class KWS. After adding the NL-ADC model to replace the nonlinear functions in the LSTM cell (Equation (4)), 91.1%, 90%, and 89.4% inference accuracy were obtained in the 5-bit, 4-bit, and 3-bit ADC models respectively which compare favorably with a floating-point baseline of 91.6%. To enhance the model's robustness against hardware non-ideal factors and minimize the decrease in inference accuracy from software to chip, we injected hardware noise (“Methods”) in the weight crossbar and NL-ADC crossbar conductances during the training process. The noise model data used in this process is obtained from the actual memristor crossbar and follows a normal distribution $\approx N(0.5 \mu\text{S})$ (Supplementary Fig. S8c). As a result, we attain inference accuracies in software of 89.4%, 88.2%, and 87.1% for the 5-bit, 4-bit, and 3-bit NL-ADC cases models respectively. Note that the drop in accuracy is much less for feedforward models as shown in Supplementary Note S8. Further, the robustness of the classification was verified by conducting 10 runs; the small standard deviation shown in Fig. 4d confirming the robustness. Through noise-aware training, we obtain weights that are robust against write noise inherent in programming memristor conductances. These weights are then mapped to corresponding conductance values through a scaling factor by matching the maximum weight after training to the maximum achievable conductance (Methods). Both the conductance values associated with the weights and the NL-ADC are programmed on the memristor crossbar, facilitating on-chip inference.

Figure 4c shows the performance of weight mapping after programming the memristor conductance using iterative methods. The error between the programmed conductance value and the theoretical value follows a normal distribution. The on-chip, experimentally measured inference accuracies achieved are 88.5%, 86.6%, and 85.2% for the 5-bit, 4-bit, and 3-bit ADC models, respectively where the redundancy techniques in Supplementary Note S11 were used for the 3-bit version. The experimental results indicate that 5-bit and 4-bit NL-ADC models can achieve higher inference accuracy than previous work^{31,48} (86.14% and 86.03%) based on same dataset and class number and are also within 2% of the software estimates, while that of the 3-bit version is marginally inferior. Nonetheless, it is important to highlight that the LSTM layer with the 3-bit ADC model significantly outperforms the 5-bit NL-ADC models in terms of area efficiency and energy efficiency as shown in Fig. 4e (31.33TOPS/W, 60.09 TOPS/W and 114.40 TOPS/W for 5-bit, 4-bit, and 3-bit NL-ADC models, respectively). The detailed calculation to assess the performance of our chip under different bit precision is done following recent work^{15,50} and shown in Supplementary Tab. S5. The earlier measurements were limited to the LSTM macro alone and did not consider the digital processor for pointwise multiplications (Eq. S3) and the small FC layer. The whole system level efficiencies are estimated in detail in Supplementary Note S4a for all three bit resolutions.

Table 1 compares the performance of our LSTM implementation with other published work on LSTM showing advantages in terms of energy efficiency, throughput and area efficiency. For a fair comparison of area efficiencies, throughput and area are normalized to a 1 GHz clock and 16 nm process respectively. Figure 4e also graphically compares the energy efficiency and normalized area efficiency of the LSTM layer in our chip for KWS task with other published LSTM hardware. The results demonstrate that our chip with 5-bit NL-ADC exhibits significant advantages in terms of normalized area efficiency ($\approx 9.9\times$) and energy efficiency ($\approx 4.5\times$ system level) compared to the closest reported works. It should be noted that comparing the raw throughput is less useful since it can be increased by increasing the number of cores. In order to dig deeper into the reason for the superiority of our chip compared to conventional linear ADC chips^{51,52}, detailed comparisons with a controlled baseline were also done using two models as shown in Fig. S3 where $k=1$ digital processor is assumed. While their inputs and outputs remain the same, the key

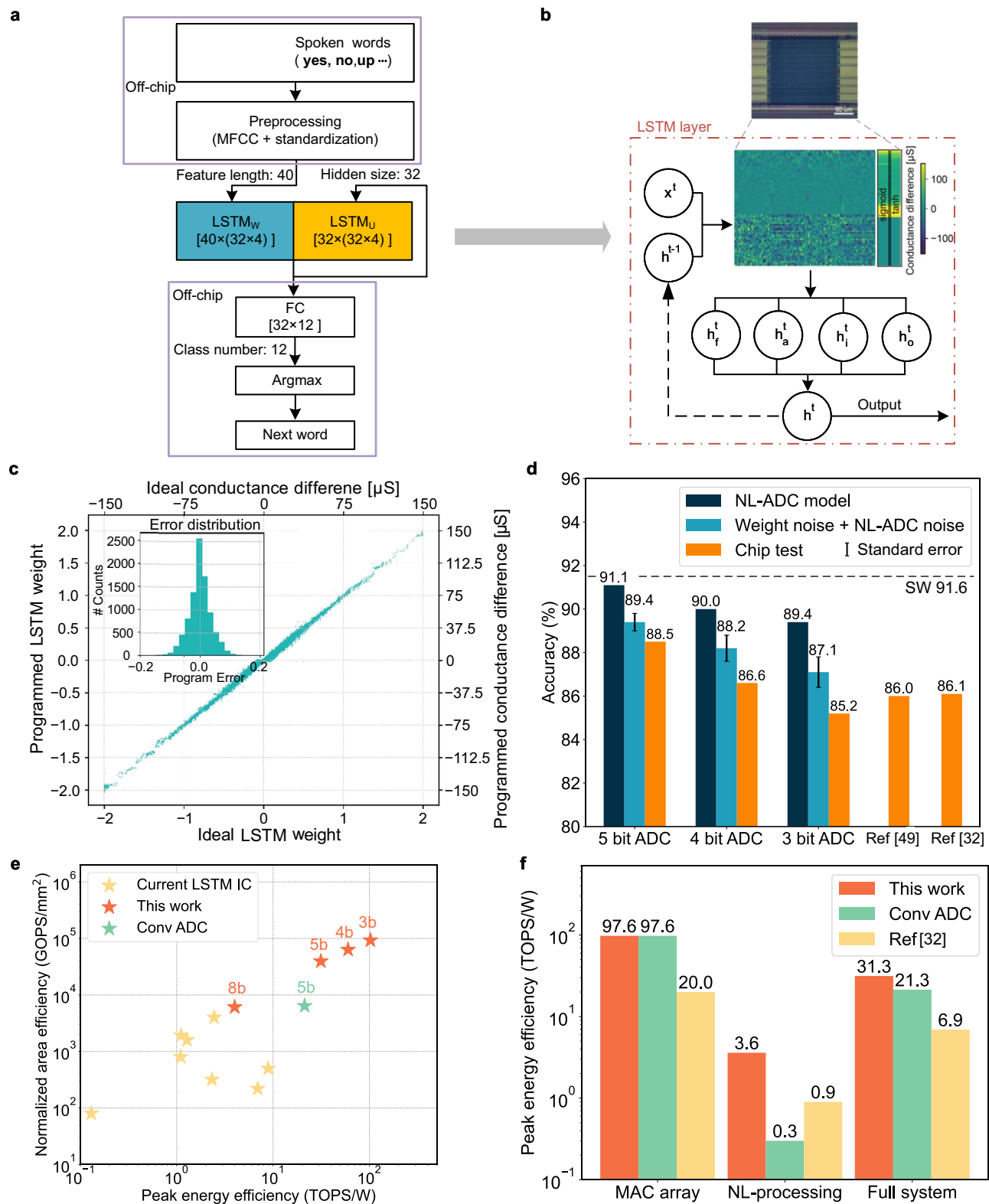


Fig. 4 | LSTM for KWS task. a Architecture of LSTM network on-chip inference. **b** Mapping of LSTM network onto the chip. Weights and nonlinearities (Sigmoid and Tanh) of LSTM layer are programmed crossbar arrays as conductance. Input and output (I/O) data of LSTM layer are sent from/to the integrated chip through off-chip circuits. **c** Weight conductance distribution curve and error. **d** The measured inference accuracy results obtained on the chip are compared with the software baseline using the ideal model, as well as simulation results under different bit NL-ADC models and hardware-measured weight noise. **e** Energy efficiency and area efficiency comparison: our LSTM IC, conventional ADC model and

recently published LSTM ICs from research papers^{23,25–28,31,67,68}. Energy efficiency and throughput under 8 bit, 5 bit, 4 bit and 3 bit NL-ADC are calculated based on 16 nm CMOS technology and clock frequency of 1 GHz. Detailed calculations are shown in Supplementary Note S3, Supplementary Note S4 and Tab. S5. Area efficiency of all works are normalized to 1 GHz clock and 16 nm CMOS process. **f** Energy efficiency comparison (this work, conventional ADC model, a chip for speech recognition using LSTM model³¹) at various levels: MAC array, NL-processing, full system. Full system includes MAC and NL-processing and other modules that assist MAC and NL-processing.

Table 1 | Comparison of LSTM performance with previous works

Metric	Our KWS task	Our NLP task	Nature'23 ³¹	Nat. Elec.'23 ²³	VLSI'17 ²⁸	JSSC'20 ²⁵	ISSCC'17 ²⁶	CICC'18 ²⁷
CMOS technology	16 nm	16 nm	14 nm	14 nm	65 nm	65 nm	65 nm	65 nm
Memory technology	RRAM	RRAM	PCM	PCM	SRAM	SRAM	SRAM	SRAM
Operation Frequency (MHz)	1000	1000	1000	1000	200	80	200	168
IMC	Y	Y	Y	Y	N	N	N	N
Input/weight/output precision	5(8)/Analog/5(8)	5(8)/Analog/5(8)	8/Analog/8	8/Analog/8	8/8/–	13/6/13	16/16/–	8/8/8
Memory size (kB)	1.125	623	4250	272	348	288	10	82
KWS task on GSCD (Accuracy %)	88.5 (12 classes)	–	86.1 (12 classes)	–	–	–	–	–
NLP task on PTB (BPC)	–	1.349	–	1.439	–	–	–	–
Area (mm ²)	0.003	0.71	111.18	144	19	7.74	2.6	0.93
Power (mW)	3.7 (5b), 4.6(8b)	406.5 (5b), 766.8(8b)	3450	3465	296	65	2.3	29.03
Peak Throughput (TOPS)	0.11(5b), 0.02(8b)	19.5(5b), 5.5(8b)	23.94	4.9	0.38	0.16/0.02	0.025	0.03
Energy Efficiency (TOPS/W)	31.0(5b), 4.0(8b)	47.9(5b), 7.2(8b)	6.94	1.96	1.28	2.45/8.93	1.1	1.11
Area Efficiency (TOPS/mm ²)	39.48 (5b), 6.1(8b)	27.6 (5b), 7.8(8b)	0.17	0.32	0.02	0.02/0.0025	0.01	0.02
Normalized Area Efficiency (TOPS/mm ² , 1 GHz, 16 nm)	39.48 (5b), 6.1(8b)	27.6 (5b), 7.8(8b)	0.22	0.32	1.6	4/0.5	0.8	1.92

Table 2 | Comparison of ADC performance with previous works

	This work	Trans. on Electron Devices'20 ⁶²	SSCL'20 ⁶³	Nat. Elec.'19 ⁶⁴	Nat. Elec.'23 ²³	Nat. Elec.'22 ⁶⁵	JSSC'22 ⁶⁵	Nature'20 ¹²	Science'23 ⁵²
ADC type	Ramp	Flash	Flash	SAR	CCO-based	SAR	Flash	SAR	Ramp
ADC resolution (bit)	5	3	1	9	12	8	3	8	8
ADC clk freq. (MHz)	1000	150	140	148	3300	8	100	20	200
# of column per ADC	1	8	8	1	1	64	8	4	1
Effective fs (MHz)	31.25	18.75	17.5	16.44	7.93	0.015	12.5	0.625	0.78
Effective ADC latency (# clock)	32	8	8	9	128	512	8	32	256
AF latency (# clock, KWS/NLP)	32/32	257/1025	257/1025	265/1033	384/1152	264/1032	257/1025	264/1032	512/1280
Power (μW)	9.3	–	–	–	–	33.18	–	51	11.9
FOM (pJ)	0.0186	–	–	–	–	0.1296	–	0.01	0.06
Process (nm)	16	90	90	180	16	55	40	130	130
Replica Bias	Y	N	N	N	N	N	Y	N	N
AF included	Y	N	N	N	N	N	N	N	N
PWM mode	Y	N	N	N	Y	N	N	N	Y

distinction lies in the nonlinear operation component. Utilizing these two architectures, we conducted evaluations on the energy consumption and area of the respective chips (Tab. S10 and Tab. S11) to find the proposed one has $\approx 1.5X$ and $\approx 2.4X$ better metrics at the system level respectively. Figure 4f presents a comprehensive comparison of energy efficiency for its individual subsystems among this work (5-bit NL-ADC), the conventional ADC model, and a reference chip³¹ using IMC—the MAC array, NL-processing, and the full system comprising the MAC array, NL-processing, and other auxiliary circuits (Tab. S13). Our approach exhibits significant energy efficiency advantages, particularly in NL-processing, with a remarkable 3.6 TOPS/W compared to 0.3 TOPS/W and 0.9 TOPS/W for the other two chips. This substantial improvement in NL-processing energy efficiency is a crucial factor contributing to the superior energy efficiency of our chip, as depicted in Fig. 4e.

The improvements in area efficiency also come about due to the improved throughput of the NL-processing. Fig. S4 shows the energy and area breakdown of the main chip components of this work (5-bit NL-ADC) and the conventional model (5-bit ADC). Our work demonstrates superior area efficiency, with a value of 130.82 TOPS/mm², compared to the conventional ADC model's 9.56 TOPS/mm² in Tab. S5. This $\approx 13X$ improvement is attributed mostly to throughput improvement of 4.6X over the digital processor in conventional systems. Table 2 compares our proposed NL-ADC with other ADC used in IMC systems. While some works have used Flash ADCs that require single cycle per conversion, they have a higher level of multiplexing (denoted by # of columns per ADC) since these require exponentially more comparators than our proposed ramp ADC. Hence, the effective AC latency in terms of number of clock cycles for our system is comparable with others. Moreover, since our proposed NL-ADC is the

only one with integrated activation function (AF) computation, the latency in data conversion followed by AF computation (denoted by AF latency in the table) is significantly lower for our design. Here, we assume LUT based AF computation using $N_{\text{cyc}} = 2$ clocks and 1 digital processor per 1024 neurons like other work²³. Compared with other ramp converters, the area occupied by our 5-bit NL-ADC is merely 558.03 μm^2 due to usage of only one row of memristors, while the traditional 5-bit Ramp-ADC^{51,52} together with nonlinear processor occupy an area of 4665.47 μm^2 . This substantial disparity in ADC area due to a capacitive DAC-based ramp generator^{51,52} leads to a further difference in the area efficiency of the two chips. Using oscillator-based ADCs^{53,54} instead of ramp-based ones will reduce the area of the ADC in traditional systems, but the throughput, area and energy-efficiency advantages of our proposed method will still remain significant. Also, note that due to the usage of memristors for reference generation, our system is robust to perturbations (such as changes in temperature or read voltage) similar to other designs using replica biasing⁵⁵. Lastly, for monotonic AF, our design can directly generate pulse width modulated (PWM) output (like other ramp based designs³¹) which can be passed to the next stage avoiding the need for DAC circuits at the input of the next stage as well as counters in the ADC. This is known to further increase energy efficiencies⁵⁶, an aspect we have not explored yet.

Scaling to large RNNs for natural language processing

Although KWS is an excellent benchmark for assessing the performance of small models^{31,57}, our nonlinear function approximation method is also useful in handling significantly larger networks for applications such as character prediction in NLP. To demonstrate the scalability of our method, we conduct simulations using a much larger LSTM model on the Penn Treebank Dataset⁴² (PTB) for character prediction. There are a total of 50 different characters in PTB. Each character is embedded into a unique random orthogonal vector of 128 dimensions, which are taken from a standard Gaussian distribution. Additionally, at each timestep, a loop of 128 cycles will be executed. The training method is shown in Methods and Fig. 5a illustrates the inference network architecture. The number of neurons and parameters in the PTB character prediction network (6,112,512 weight parameters and 8568 biases in LSTM and projection layers) are $\approx 66\times$ and $\approx 600\times$ more than the corresponding numbers in the KWS network (Fig. 5b) leading to a much larger number of operations per timestep.

To map the problem onto memristive arrays, the LSTM layer alone needs a 633×8064 prohibitively large crossbar. Instead, we partition the problem and map each section to a 633×512 crossbar (similar to recent approaches³¹) with 16 such crossbars for the entire problem. Within each crossbar, only 256 input lines are enabled at one phase to prevent large voltage drops along the wires, with a total of 3 phases to present the whole input, with the concomitant cost of $3\times$ increase in input presentation time. An architecture for further reduced crossbars of size 256×256 is shown in Supplementary Note S10. To assess the impact of on-chip buffers and interconnects in performing data transfer between tiles, Neurosim⁵⁸ is used to perform system level simulations where the ADC in the tile is replaced by our model (details in Supplementary Note S4(b)). First, the accuracy of the character prediction task is assessed using bits per character (BPC)²³, which is a metric that measures the model's ability to predict samples from the true underlying probability distribution of the dataset, where lower values indicate better performance. Similar to the earlier KWS case, memristor write noise and NL-ADC quantization effects from the earlier hardware measurements are both included in the simulation. The results of inference are displayed in Fig. 5c, with a software baseline of 1.334. BPC results of 1.345, 1.355, and 1.411 are obtained with the 5-bit, 4-bit, and 3-bit ADC models, respectively when considering perfect weights, and exhibits a drop of only 0.011 for the 5-bit NL-ADC model compared to the software baseline. Finally, the write noise of the

memristors are included during both the training and testing phases using the same method as the KWS model resulting in BPC values of 1.349, 1.367, and 1.428 are obtained with the 5-bit, 4-bit, and 3-bit NL-ADC models (with error bars showing standard deviations for 10 runs). Compared to other recent work²³ on the same dataset that obtained a BPC of 1.358, our results are promising and show nonlinear function approximation by NL-ADC can be successfully applied to large-scale NLP models. There may be some other applications which are more sensitive to AF variations; for those cases, an iteration of chip-in-the-loop training may be necessary after hardware-aware training.

The throughput, area, and energy efficiencies are estimated next (details in Supplementary Note S3, Tab. S6 at the macro level and in Supplementary Note S4b at the system level) and compared with a conventional architecture (Tab. S7, S8) for two different cases of $k = 1$ and $k = 8$ digital processors. These are compared along with current LSTM IC metrics in Fig. 5d and Fig. 5e. Considering the 5-bit NL-ADC, our estimated throughput and energy efficiency of 19.5 TOPS and 47.9 TOPS/W at the system level are $\approx 4.0\times$ and $24.4\times$ better for system level than earlier reported metrics²³ of 4.9 TOPS and 2 TOPS/W respectively. Lastly, the normalized area efficiency of our NL-ADC based LSTM layer is $\approx 86\times$ better at system level than earlier work²³ (reporting results on same benchmark) due to the increased throughput and reduced area (we also estimated that an 8-bit version of our system will still be $24\times$ more area efficient and $3.6\times$ more energy efficient). Compared to the conventional IMC architecture baseline for this LSTM layer, we estimate an energy efficiency advantage of $1.1\times$ at the system level similar to the KWS case, but the throughput and area efficiency advantages of $\approx 4.8\times$ and $\approx 6.6\times$ for system level respectively remain even for $k = 8$ digital processors.

Discussion

In conclusion, we proposed and experimentally demonstrated a novel paradigm of nonlinear function approximation through a memristive in-memory ramp ADC. By predistorting the ramp waveform to follow the inverse of the desired nonlinear activation, our NL-ADC removes the need for any digital processor to implement nonlinear activations. The analog conductance states of the memristor enable the creation of different programmable voltage steps using a single device, resulting in great area savings over a similar SRAM-based implementation. Moreover, the in-memory ADC is shown to be more robust to voltage fluctuations compared to a conventional ADC with memristor crossbar based MAC. Using this approach, we implemented a LSTM network using 9216 weights programmed in the 72×128 memristor chip to solve a 12-class keyword spotting problem using the GSCD. The results for the 5-bit ADC show better accuracy of 88.5% than previous hardware implementations^{31,49} with significant advantages in terms of normalized area efficiency ($\approx 9.9\times$) and energy efficiency ($\approx 4.5\times$) compared to previous LSTM circuits. We further tested the scalability of our system by simulating a much larger network (6,112,512 weights) for NLP using the experimentally validated models. Our network with 5-bit NL-ADC again achieves better performance in terms of BPC than recent reports²³ of IMC based LSTM ICs while delivering $\approx 86\times$ and $\approx 24.4\times$ better area and energy efficiencies at the system level. Our work paves the way for very energy efficient in-memory nonlinear operations that can be used in a wide variety of applications.

Methods

Memristor integration

The memristors are incorporated into a CMOS system manufactured in a commercial foundry using a 180 nm technology node. The integration process starts by eliminating the native oxide from the surface metal through reactive ion etching and a buffered oxide etch dip. Subsequently, chromium and platinum are sputtered and patterned with e-beam lithography to serve as the bottom electrode. This is followed by the application of reactively sputtered 2 nm tantalum oxide

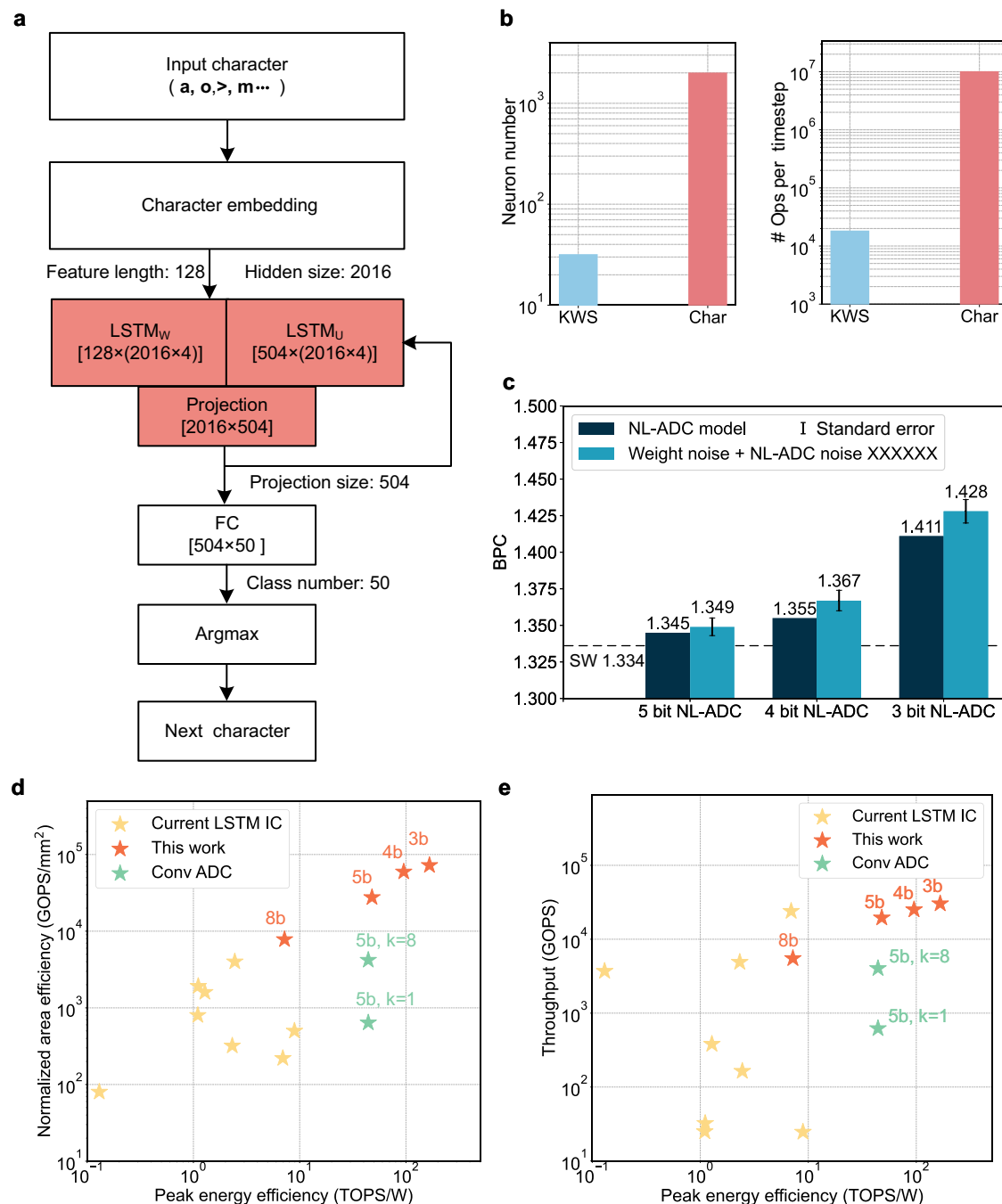


Fig. 5 | LSTM for NLP task. a Architecture of LSTM network for on-chip inference in character prediction task. **b** Comparison in the LSTM layer between the number of neurons and operations per timestep in the NLP model for character prediction and the KWS model. **c** Simulation results under different bit resolution of NL-ADC models and hardware-measured weight noise compared with software baseline using the ideal model. BPC results follow the “smaller is better” principle, meaning that lower values indicate better performance. **d** Energy efficiency and area

efficiency comparison: our LSTM IC, conventional ADC model and recently published LSTM ICs from research papers^{23,25–28,31,67,68}. Detailed calculation of energy efficiency and throughput for both macro and system levels are shown in Supplementary Note S3, Supplementary Note S4 and Tab. S9. Area efficiency of all works are normalized to 1 GHz clock and 16 nm CMOS process. **e** Energy efficiency and throughput comparison: our LSTM IC, conventional ADC model and recently published LSTM ICs from research papers^{23,25–28,31,67,68}.

as the switching layer and sputtered tantalum metal as the top electrode. The device stack is completed with sputtered platinum for passivation and enhanced electrical conduction.

Memristor programming methods

In this work, we adopt the iterative-write-and-verify programming method to map the weights to the analog conductances of memristors. Before programming, a tolerance value (5 μ S) is added to the desired

conductance value to allow certain programming errors. The programming will end if the measured device conductance is within the range of 5 μ S above or below the target conductance. During programming, successive SET or RESET pulses with 10 μ s pulse width are added to each single 1T1R structure in the array. Each SET or RESET pulse is followed by a 20 ns READ pulse. A RESET pulse is added to the device if its conductance is above the tolerated range while a SET pulse will be added if its conductance is below the range. We will gradually

increase the amplitude of the SET/RESET voltage and the gate voltage of transistors between adjacent cycles. For SET pulse amplitude, we start from 1 V to 2.5 V with an increment of 0.1 V. For RESET pulse amplitude, we start from 0.5 V to 3.5 V with an increment of 0.05 V. For gate voltage of SET process, we start from 1.0 V to 2.0 V with an increment of 0.1 V. For gate voltage of RESET process, we start from 5.0 V to 5.5 V with an increment of 0.1 V. Detailed programming process is illustrated in Supplementary Fig. S9.

Hardware aware training

Directly mapping weights of the neural network to crossbars will heavily degrade the accuracy. This is mainly due to the programming error of the memristors. To make the network more error-tolerant when mapping on a real crossbar chip, we adopted the defect-aware training proposed in previous work⁴⁰. During training, we inject the random Gaussian noise into every weight value in the forward parts for gradient calculation. Then the back-propagation happens on the weights after noise injection. Weight updating will occur on the weight before the noise injection. We set the standard deviation to 5 μS which is relatively larger than experimentally measured programming error (2.67 μS shown in Supplementary Fig. S8c) to make the model adapt more errors when mapping to real devices. Detailed defect-aware training used in this work is described in Algorithm 1. In this work, we set σ to 5 μS and g_{max} to 150 μS .

Algorithm 1. Defect-aware training

Data: Weight matrix at training iteration: \mathbf{W}_μ^t ; input data X ; learning rate: α ; weight-to-conductance ratio g_{ratio} ; maximum conductance in crossbar g_{max} ; injected noise σ ; loss function L

Result: Weight at time step $t+1$: \mathbf{W}_μ^{t+1}

```

 $\mathbf{W}_\mu[g_{\text{ratio}} \mathbf{W}_\mu^t > g_{\text{max}}] = g_{\text{max}}/g_{\text{ratio}}$ 
 $\mathbf{G}_\mu^t \leftarrow |\mathbf{W}_\mu^t| \cdot g_{\text{ratio}}$  (differential mapping)
Initialize  $\mathbf{G}_\sigma^t$ : Conductance standard deviation
 $\mathbf{W}_\sigma^t \leftarrow \mathbf{G}_\sigma^t/g_{\text{ratio}}$ 
 $\mathbf{W}^t \leftarrow \mathbf{W}_\mu^t + \epsilon \cdot \mathbf{W}_\sigma^t$   $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
Compute loss  $L(X, \mathbf{W}^t)$ 
Update  $\mathbf{W}_\mu$  through back propagation
 $\mathbf{W}_\mu^{t+1} \leftarrow \mathbf{W}_\mu^t - \alpha \cdot \frac{\partial L}{\partial \mathbf{W}_\mu}$ 

```

Weight clipping and mapping

We clip weights between -2 and 2 to avoid creating excessively large weights during training. Weights can be mapped to the conductance of the memristors when doing on-chip inference, nearly varying from 0 μS to 150 μS . The clipping method is defined according to Equation (6) and the mapping method is shown in Equation (7).

$$w = \begin{cases} -2 & w < -2 \\ w & -2 \leq w \leq 2 \\ 2 & w > 2 \end{cases} \quad (6)$$

$$g = \gamma w \left(\gamma = \frac{g_{\text{max}}}{|w|_{\text{max}}} \right) \quad (7)$$

where w is weights during training, g is the conductance value of memristors and γ is a scaling factor used to connect the weights to g . The maximum conductance of memristors (g_{max}) is 150 μS and the maximum absolute value of weights ($|w|_{\text{max}}$) is 2, therefore, the scaling factor (γ) is equal to 75 μS .

Training for LSTM keyword spotting model

The training comprises of two processes: preprocessing and LSTM model training.

Preprocessing: The GSCD⁴¹ is used to train model. It has 65,000 one-second-long utterances of 30 short words and several background noises, by thousands of different people, contributed by members of the public⁵⁷. We reclassify the original 31 classes into the following 12 classes⁴⁹: yes, no, up, down, left, right, on, off, stop, go, background noise, unknown. The unknown class contains the other 20 classes. For every one-second-long utterance, the number of sampling points is 16,000. MFCC⁵⁷ is applied to extract Mel-frequency cepstrum of voice signals. 49 windows are used to divide a one-second-long audio signal and extract 40 feature points per window.

LSTM model training: The custom LSTM layer is the core of this training model. The custom layer is necessary for modifying the parameters, including adding the NL-ADC algorithm to replace the activation function inside, adding weight noise training, quantizing weights, etc. Although the custom LSTM layer will increase the training time of the network, this is acceptable by weighing its advantages and disadvantages. The input length is 40 and the hidden size is 32. The sequence length is 49, which means the LSTM cell will iterate 49 times in one batch size of 256.

A FC layer is added after the LSTM layer to classify the features output by LSTM. The input size of the FC layer is 32 and the output size is 12 (class number). Cross-entropy loss is used to calculate loss. We train the ideal model (without NL-ADC and noise) for 128 epochs and update weights using the Adam optimizer with a learning rate (LR = 0.001). After finishing the ideal model training, NL-ADC-aware training and hardware noise-aware training are added. All models' performance is evaluated with top-1 accuracy.

Training for LSTM character prediction model

Preprocessing: The PTB⁴² is a widely used corpus in language model learning, which has 50 different characters. Both characters in the training dataset and the validation dataset of PTB are divided into many small sets. Each set consists of 128 characters and each character is embedded into a random vector (dimension $D = 128$) obtained from the standard Gaussian distribution and then perform Gram-Schmidt orthogonalization on these vectors.

LSTM model training: We use a one-layer custom LSTM with projection²³. The input length is 128 and the hidden size is 2016. The length hidden state and the LSTM output are both 504. The sequence length is 128, which means the LSTM cell will loop 128 times in one batch (batch size = 8).

The FC layer after the LSTM layer will further extract the features output by LSTM and convert them into an output of size 50 (class number). Cross-entropy loss is used to calculate loss. We train the model for 30 epochs and update weights using the Adam⁵⁹ optimizer with a learning rate (LR = 0.001). The model's performance is evaluated through the BPC⁶⁰ metric and the data of BPC is smaller the better. After finishing the ideal model training, we use the same training method to train the model after adding NL-ADC and hardware noise.

Inference with the addition of write noise and read noise

During the inference stage, we performed 10 separate simulations with different write noise following the measured distribution $\mathcal{N}(0, 2.67 \mu\text{S})$ (Fig. S8c) in each case (simulating 10 separate chips). For each of the simulation, read noise following measured read noise distribution $\mathcal{N}(0, 3.5 \mu\text{S})$ (Fig. S14b) is included. It is worth noting that in each chip simulation, a consistent write noise was introduced into the inference process applied to the entire test dataset. However, in relation to read noise, the normal distribution $\mathcal{N}(0, 3.5 \mu\text{S})$ is employed for each mini-batch to generate distinct random noises. These noises are subsequently incorporated into the simulation. Then we obtain the inference accuracy with the addition of write noise and read noise.

Data availability

The data that support the findings of this study are available in an online repository⁶¹ and other findings of this study are available with reasonable requests made to the corresponding author.

Code availability

The code used to train the model and perform the simulation on crossbar arrays is publicly available in an online repository⁶¹.

References

- Kolesnikov, A. et al. An image is worth 16×16 words: transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)* (OpenReview.net, 2020).
- Graves, A., Mohamed, A.-R. & Hinton, G. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* 6645–6649 (IEEE, 2013).
- Silver, D. et al. Mastering the game of go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016).
- Senior, A. W. et al. Improved protein structure prediction using potentials from deep learning. *Nature* **577**, 706–710 (2020).
- Vaswani, A. et al. Attention is all you need. In *Conference and Workshop on Neural Information Processing Systems (NIPS)* (Association of Computational Machinery, 2017).
- Horowitz, M. Computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)* 10–14 (IEEE, 2014).
- Yang, J., Strukov, D. & Williams, S. Memristive devices for computing. *Nat. Nanotechnol.* **8**, 13–24 (2013).
- Prezioso, M. et al. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* **521**, 61–64 (2015).
- Alibart, F., Zamanidoost, E. & Strukov, D. Pattern classification by memristive crossbar circuits using ex situ and in situ training. *Nat. Commun.* **4**, 2072 (2013).
- Can, L. et al. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nat. Commun.* **9**, 2385 (2018).
- Sebastian, A., Le Gallo, M., Khaddam-Aljameh, R. & Eleftheriou, E. Memory devices and applications for in-memory computing. *Nat. Nanotechnol.* **15**, 529–544 (2020).
- Peng, Y. Fully hardware-implemented memristor convolutional neural network. *Nature* **577**, 641–646 (2020).
- Kiani, F. et al. A fully hardware-based memristive multilayer neural network. *Sc. Adv.* **7**, eabj4801 (2021).
- Yang, K. et al. Transiently chaotic simulated annealing based on intrinsic nonlinearity of memristors for efficient solution of optimization problems. *Sci. Adv.* **6**, eaba9901 (2020).
- 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).
- John, R. A. et al. Halide perovskite memristors as flexible and reconfigurable physical unclonable functions. *Nat. Commun.* **12**, 3681 (2021).
- Mao, R. et al. Experimentally validated memristive memory augmented neural network with efficient hashing and similarity search. *Nat. Commun.* **13**, 6284 (2022).
- Sheridan, P. M. et al. Sparse coding with memristor networks. *Nat. Nanotechnol.* **12**, 784–789 (2017).
- Zidan, M. A. et al. A general memristor-based partial differential equation solver. *Nat. Electron.* **1**, 411–420 (2018).
- Le Gallo, M. Mixed-precision in-memory computing. *Nat. Electron.* **1**, 246–253 (2018).
- Khwa, W. S. et al. A 40-nm, 2M-cell, 8b-precision, hybrid SLC-MLC PCM computing-in-memory macro with 20.5-65.0TOPS/W for tiny-AI edge devices. In *2022 IEEE International Solid-State Circuits Conference-(ISSCC)* 1–3 (IEEE, 2022).
- J.-M, H. A four-megabit compute-in-memory macro with eight-bit precision based on CMOS and resistive random-access memory for AI edge devices. *Nat. Electron.* **4**, 921–930 (2021).
- 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).
- Giraldo, J. S. P. & Verhelst, M. Laika: A 5 μ 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).
- Kadetotad, D., Yin, S., Berisha, V., Chakrabarti, C. & Seo, J.-S. An 8.93 TOPS/W LSTM recurrent neural network accelerator featuring hierarchical coarse-grain sparsity for on-device speech recognition. *IEEE J. Solid State Circuits* **55**, 1877–1887 (2020).
- Shin, D., Lee, J., Lee, J. & Yoo, H.-J. DNPU: An 8.1 TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)* 240–241 (IEEE, 2017).
- Conti, F., Cavigelli, L., Paulin, G., Susmeli, I. & Benini, L. Chipmunk: a systolically scalable 0.9 mm $2, 3.08$ Gop/s/mW@ 1.2 mW accelerator for near-sensor recurrent neural network inference. In *2018 IEEE Custom Integrated Circuits Conference (CICC)* 1–4 (IEEE, 2018).
- Yin, S. et al. A 1.06-to-5.09 TOPS/W reconfigurable hybrid-neural-network processor for deep learning applications. In *2017 Symposium on VLSI Circuits C26–C27* (IEEE, 2017).
- Li, C. et al. Long short-term memory networks in memristor crossbar arrays. *Nat. Mach. Intell.* **1**, 49–57 (2019).
- Tsai, H. et al. Inference of long-short term memory networks at software-equivalent accuracy using 2.5 M analog phase change memory devices. In *2019 Symposium on VLSI Technology T82–T83* (IEEE, 2019).
- Ambrogio, S. et al. An analog-AI chip for energy-efficient speech recognition and transcription. *Nature* **620**, 768–775 (2023).
- Xie, Y. et al. A twofold lookup table architecture for efficient approximation of activation functions. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **28**, 2540–2550 (2020).
- Arvind, T. K. et al. Hardware implementation of hyperbolic tangent activation function for floating point formats. In *2020 24th International Symposium on VLSI Design and Test (VDATE)* 1–6 (IEEE, 2020).
- Kwon, D. et al. A 1ynm 1.25 v 8gb 16gb/s/pin gddr6-based accelerator-in-memory supporting 1tflops mac operation and various activation functions for deep learning application. *IEEE J. Solid State Circuits* **58**, 291–302 (2022).
- Raut, G. et al. A CORDIC based Configurable Activation Function for ANN Applications. In *International Symposium on VLSI (ISVLSI)* (IEEE, 2020).
- Chong, Y. et al. Efficient implementation of activation functions for LSTM accelerators. In *VLSI System on Chip (VLSI-SOC)* (IEEE, 2021).
- Pasupuleti, S. K. et al. Low complex & high accuracy computation approximations to enable on-device RNN applications. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)* 1–5 (IEEE, 2019).
- Feng, X. et al. A high-precision flexible symmetry-aware architecture for element-wise activation functions. In *2021 International Conference on Field-Programmable Technology (ICFPT)* 1–4 (IEEE, 2021).
- Li, Y., Cao, W., Zhou, X. & Wang, L. A low-cost reconfigurable nonlinear core for embedded DNN applications. In *2020 International Conference on Field-Programmable Technology (ICFPT)* 35–38 (IEEE, 2020).
- Mao, R., Wen, B., Jiang, M., Chen, J. & Li, C. Experimentally-validated crossbar model for defect-aware training of neural networks. *IEEE Trans. Circuits Syst. II Express Briefs* **69**, 2468–2472 (2022).

41. Warden, P. Speech commands: a dataset for limited-vocabulary speech recognition. arXiv preprint arXiv:1804.03209 (2018).
42. Marcus, M., Santorini, B. & Marcinkiewicz, M. A. Building a large annotated corpus of English: The Penn Treebank. *Comput. Linguist.* **19**, 313–330 (1993).
43. Allen, P. E & Holberg, D. *CMOS Analog Circuit Design* (Oxford University Press, 2011).
44. 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).
45. Rao, M. et al. Thousands of conductance levels in memristors integrated on CMOS. *Nature* **615**, 823–829 (2023).
46. Song, W. et al. Programming memristor arrays with arbitrarily high precision for analog computing. *Science* **383**, 903–910 (2024).
47. Sheng, X. et al. Low-conductance and multilevel CMOS-integrated nanoscale oxide memristors. *Adv. Electron. Mater.* **5**, 1800876 (2019).
48. Kim, K. et al. A 23 μ W solar-powered keyword-spotting ASIC with ring-oscillator-based time-domain feature extraction. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)* Vol. 65, 1–3 (IEEE, 2022).
49. Kim, K. et al. A 23- μ W keyword spotting IC with ring-oscillator-based time-domain feature extraction. *IEEE J. Solid State Circuits* **57**, 3298–3311 (2022).
50. Cai, F. Power-efficient combinatorial optimization using intrinsic noise in memristor hopfield neural networks. *Nat. Electron.* **3**, 409–418 (2020).
51. 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).
52. Zhang, W. et al. Edge learning using a fully integrated neuro-inspired memristor chip. *Science* **381**, 1205–1211 (2023).
53. 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. Circuits Syst. I Regul. Pap.* **66**, 2240–2252 (2018).
54. 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).
55. Li, W. et al. A 40-nm MLC-RRAM compute-in-memory macro with sparsity control, on-chip write-verify, and temperature-independent ADC references. *IEEE Journal of Solid-State Circuits (JSSC)* 2868–2877 (IEEE, 2022).
56. Jiang, H. et al. A 40nm analog-input ADC-free compute-in-memory RRAM macro with pulse-width modulation between sub-arrays. In *2022 Symposium on VLSI Circuits* (IEEE, 2022).
57. Shan, W. et al. A 510-nW wake-up keyword-spotting chip using serial-FFT-based MFCC and binarized depthwise separable CNN in 28-nm CMOS. *IEEE J. Solid State Circuits* **56**, 151–164 (2020).
58. “DNN+NeuroSim framework”. https://github.com/neurosim/DNN_NeuroSim_V2.1.
59. Kingma, D. & J. B. Adam: A Method for Stochastic Optimization. In *The International Conference on Learning Representations (ICLR)* (OpenReview.net, 2014).
60. “Evaluation Metrics for Language Modeling,”. <https://thegradient.pub/understanding-evaluation-metrics-for-language-models/>. Accessed: 2024-1-31.
61. Yang, J. et al. Efficient nonlinear function approximation in analog resistive crossbars for recurrent neural networks. Zenodo. <https://doi.org/10.5281/zenodo.14609235> (2025).
62. Yin, S., Sun, X., Yu, S. & Seo, J.-s. High-throughput in-memory computing for binary deep neural networks with monolithically integrated rram and 90-nm cmos. *IEEE Trans. Electron. Devices* **67**, 4185–4192 (2020).
63. He, W. et al. 2-bit-per-cell rram-based in-memory computing for area-/energy-efficient deep learning. *IEEE Solid State Circuits Lett.* **3**, 194–197 (2020).
64. Cai, F. et al. A fully integrated reprogrammable memristor-cmos system for efficient multiply-accumulate operations. *Nat. Electron.* **2**, 290–299 (2019).
65. Huo, Q. et al. A computing-in-memory macro based on three-dimensional resistive random-access memory. *Nat. Electron.* **5**, 469–477 (2022).
66. “Survey of neuromorphic and machine learning accelerators in SOVC, ISSCC and Nature/Science series of journals from 2017 onwards,” accessed 23 December 2023. https://docs.google.com/spreadsheets/d/1j-R-QigJTuk6W5Jg8w2Yl85Tn2J_S-x/edit?usp=drive_link&oid=117536134117165308204&rtpof=true&sd=true.
67. Yue, J. et al. A 65nm 0.39-to-140.3 TOPS/W 1-to-12b unified neural network processor using block-circulant-enabled transpose-domain acceleration with 8.1 \times higher TOPS/mm² and 6T HBST-TRAM-based 2D data-reuse architecture. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)* 138–140 (IEEE, 2019).
68. Jouppi, N. P. et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* 1–12 (Association for Computing Machinery, 2017).

Acknowledgements

This work was supported in part by CityU SGP grant 9380132 and ITF MSRP grant ITS/018/22MS; in part by RGC (27210321, C1009-22GF, T45-701/22-R), NSFC (62122005) and Croucher Foundation. Any opinions, findings, conclusions, or recommendations expressed in this material do not reflect the views of the Government of the Hong Kong Special Administrative Region, the Innovation and Technology Commission, or the Innovation and Technology Fund Research Projects Assessment Panel.

Author contributions

J.Y. and A.B. conceived the idea. J.Y. performed software experiments with help from Y.C., H.L., and P.S.V.S. with software baselines for KWS and NLP tasks. R.M. and M.J. performed hardware experiments with help from X.S., G.P., and J.I. on device fabrication, IC design and system setup respectively. S.D. helped with system simulations using Neurosim. J.Y., R.M., C.L., and A.B. wrote the manuscript with inputs from all authors.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at <https://doi.org/10.1038/s41467-025-56254-6>.

Correspondence and requests for materials should be addressed to Can Li or Arindam Basu.

Peer review information *Nature Communications* thanks Huaqiang Wu, and the other, anonymous, reviewer(s) for their contribution to the peer review of this work. A peer review file is available.

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

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

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

© The Author(s) 2025