

Experimentally realized memristive memory augmented neural network

Ruibin Mao

University of Hong Kong

Bo Wen

University of Hong Kong <https://orcid.org/0000-0002-6106-9509>

Arman Kazemi

University of Notre Dame <https://orcid.org/0000-0002-2009-5516>

Yahui Zhao

The University of Hong Kong

Ann Franchesca Laguna

De La Salle University <https://orcid.org/0000-0001-8267-1040>

Rui Lin

The University of Hong Kong

Ngai Wong

The University of Hong Kong <https://orcid.org/0000-0002-3026-0108>

Michael Niemier

University of Notre Dame

Xiaobo Hu

University of Notre Dame

Xia Sheng

Hewlett Packard Enterprise

Catherine Graves

Hewlett Packard Enterprise <https://orcid.org/0000-0002-0907-583X>

John Paul Strachan

Forschungszentrum Jülich GmbH

Can Li (✉ canl@hku.hk)

The University of Hong Kong <https://orcid.org/0000-0003-3795-2008>

Article

Keywords:

Posted Date: July 5th, 2022

DOI: <https://doi.org/10.21203/rs.3.rs-1821052/v1>

License:  This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Version of Record: A version of this preprint was published at Nature Communications on October 21st, 2022. See the published version at <https://doi.org/10.1038/s41467-022-33629-7>.

Experimentally realized memristive memory augmented neural network

Ruibin Mao¹, Bo Wen¹, Arman Kazemi^{2,3}, Yahui Zhao¹, Ann Franchesca Laguna^{3,6}, Rui Lin¹, Ngai Wong¹, Michael Neimier³, X. Sharon Hu³, Xia Sheng², Catherine E. Graves^{2,*}, John Paul Strachan^{4,5,*}, and Can Li^{1,*}

¹Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong SAR, China

²Hewlett Packard Labs, Hewlett Packard Enterprise, Milpitas, CA, USA

³Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA

⁴Peter Grünberg Institut (PGI-14), Forschungszentrum Jülich GmbH, Jülich, Germany

⁵RWTH Aachen University, Aachen, Germany

⁶Current address: Department of Computer Technology, De La Salle University, Manila, Philippines

*canl@hku.hk, j.strachan@fz-juelich.de, catherine.graves@hpe.com

Abstract

Lifelong on-device learning is a key challenge for machine intelligence, and this requires learning from few, often single, samples. Memory augmented neural network has been proposed to achieve the goal, but the memory module has to be stored in an off-chip memory due to its size. Therefore its practical use has been heavily limited. Previous works on emerging memory-based implementation had difficulties in scaling up because different modules with various structures were difficult to integrate on the same chip and the small sense margin of the content addressable memory for the memory module heavily limited the degree of mismatch calculation. In this work, we experimentally validated that all of the different structures in the memory augmented neural network architecture can be implemented in a fully integrated memristive crossbar platform and achieve an accuracy that closely matches standard software on digital hardware for the Omniglot dataset. The successful demonstration is supported by implementing new functions in crossbars in addition to widely reported matrix multiplications. In addition to crossbar-based content addressable memory modules (used to calculate degrees of mismatch), locality sensitive hashing operations (used to generate the signature stored in the associative memory) have also been implemented in crossbar arrays by exploiting the intrinsic stochasticity of memristor devices. Simulations based on experimentally validated models show such an implementation can be efficiently scaled up

for one-shot learning on the Mini-ImageNet dataset. The successful demonstration paves the way for practical on-device lifelong learning, and opens possibilities for novel attention-based algorithms not possible in conventional hardware.

Introduction

Deep neural networks (DNN) have achieved massive success in data-intensive applications but fail to tackle tasks with a limited number of examples. On the other hand, our biological brain can learn patterns from rare classes at a rapid pace, which could relate to the fact that we can recall information from an associative, or content-based addressable, memory. Inspired by our brain, recent machine learning models such as memory augmented neural networks (MANN)¹ have adopted a similar concept, where explicit external memories are applied to store and retrieve learned knowledge. While those models have shown the ability to generalize from rare cases, they have struggled to “scale up”^{2,3}. This is because the entire external memory module needs to be accessed from the memory to recall the learned knowledge, which greatly increases the memory overhead. The performance in a traditional von Neumann computing architecture⁴ is thus bottlenecked in hardware by memory bandwidth and capacity issues⁵⁻⁷, especially when they are deployed in edge devices, where energy sources are limited.

Emerging non-volatile memories, e.g., memristors⁸, have been proposed and demonstrated to solve the bandwidth and memory capacity issues in various computing workloads, including deep neural networks⁹⁻¹⁴, signal processing^{15,16}, scientific computing^{17,18}, solving optimization problems^{19,20}, and more. Those solutions are based on the memristor’s ability to directly process analog signals at the location where the information is stored. Most existing demonstrations mentioned above, however, mainly focus on executing matrix multiplications for accelerating deep neural networks with crossbar structures^{8-12,17,18,21}, whose experience cannot be directly applied to the models with explicit external memories in MANNs. Recently, several pioneering works aim to solve the problem with memristor-based hardware. One promising solution is to exploit the hyperdimensional computing paradigm^{22,23}. The recent prototype of this framework showcased few-shot image classification using more than 256k phase change memristors in mixed software-hardware experiments.²³ Ferroelectric device based ternary content addressable memories (TCAMs) have been proposed to be used as the hardware to calculate the similarity directly in the memory^{24,25}, but it is only suitable for degree of mismatch up to a few bits. Besides, the locality sensitive hashing (LSH) function that enables the degree of mismatch search in the TCAM was implemented in software, and the experimental demonstration was limited to a 2×2 TCAM array. More recently, a 2T-2R TCAM associative memory was used to demonstrate few-shot learning by calculating L_1 distance²⁶. In this work a 2-bit readout scheme is employed (requiring 64 cycles per row) which incurs high energy and latency overheads, and feature extraction is again relegated to a digital processor. The key challenge in this concept is the imperfections in the analog hardware, such as device variation, fluctuation, state drift, and readout noise during the massively

parallel operations in physical crossbar or TCAM arrays; all of the above represent obstacles to viable, deployable, and efficient hardware realizations of MANNs.

In this work, we experimentally demonstrate one- and few-shot learning with the different structures in MANN, including the CNN controller and the explicit memory, implemented in our integrated memristor hardware. To achieve this goal, we implement different functionalities in crossbars in addition to the widely reported matrix multiplication operations, and design the peripheral circuit to support those functionalities accordingly. One enabler is the locality-sensitive hashing (LSH) function in crossbars, where we exploit the intrinsic stochasticity of memristor devices. This is different from crossbars for matrix multiplications, where stochasticity needs to be minimized. Another innovation is implementing search functions by using the crossbar as a TCAM. In addition to what’s possible with conventional TCAMs, the proposed scheme can also measure the degree of mismatch reliably, which is crucial for few-shot learning implementation. Since the requirements for those functions are different from conventional matrix multiplications, here we introduce several hardware-software co-optimization methods, including the introduction of the wildcard ‘X’ bit in the crossbar-based LSH, and the careful choice of conductance range according to the device statistics.

Finally, with a fully integrated memristor computing system, we are able to *experimentally demonstrate* the few-shot learning with a complete MANN model, including a five-layer convolutional neural network (CNN), the hashing function, and the similarity searches. Taking into consideration all imperfections in the emerging system, our hardware achieves $94.9\% \pm 3.7\%$ accuracy in the 5-way 1-shot task with the Omniglot dataset²⁷, a popular benchmark for few-shot image classification, and $74.9\% \pm 2.4\%$ accuracy in 25-way 1-shot task, which is close to the software baseline ($95.2\% \pm 2.6\%$ for 5-way 1-shot and $76.0\% \pm 2.7\%$ for 25-way 1-shot). Our experimentally-validated model also shows that the proposed method is scalable with a 58.7 % accuracy to recognize new classes (5-way 1-shot) for the Mini-ImageNet dataset²⁸, where each image is a color (RGB) image of size 84×84 pixels – nine times larger than the size of images in the Omniglot dataset (28×28 pixels). This accuracy is only 1.3% below the software baseline. Owing to the in-memory analog processing capability and the massive parallelism, our hardware achieves more than $10,000\times$ improvement in latency and $2,800\times$ improvement in energy consumption of one search on crossbar arrays as compared to the general-purpose graphic processing unit (GPGPU) (Nvidia Tesla P100).

Memory augmented neural networks in crossbars

The MANN architecture commonly includes a controller, an explicit memory, and a content-based attention mechanism between the two. A controller is usually a traditional neural network structure such as a convolutional neural network, a recurrent neural network (RNN), or a combination of different neural networks. The explicit memory stores the extracted feature vectors as the key-value pairs so that the model can identify the values based on the similarity or distances between the keys. The access of the explicit memory is the performance bottleneck for models that run on conventional hardware, such as the general-purpose graphic processing unit (GPGPU). It is because the similarity search requires accessing all the content in the memory; thus the repeated data transfer process delays the readout process and consumes abundant energy, especially when the memory needs to be placed in a separate chip.

We implement all computations for end-to-end inference in our memristor-based hardware system to address the expensive data transfer issue in conventional digital hardware. The system aims to perform both matrix multiplication, for neural network controllers, and the similarity search, for the explicit memory, directly in the memristor crossbars, where the data is stored. Memristors have demonstrated great success in accelerating matrix multiplications in neural networks, which is also the controller part of the MANN, owing to their extreme density²⁹, the compatibility with CMOS integration^{9,30}, and the capability to process analog information directly in its memory. But, the demonstrations for explicit memories are still limited^{23–25}.

Fig. 1a illustrates how we implement the MANN in the crossbars. **First**, a regular crossbar-based convolutional neural network is used to extract the real-valued feature vector, and the method implementing this step has been widely reported previously^{11,31,32}. After that, distances are calculated between the extracted feature vector and those stored in a memory. Cosine distance (CD) is one of the most widely used distance metrics in the explicit memory of various MANN implementations, but it is not straightforward to implement with memristor-based crossbars. **Next**, on the other hand, the cosine distance between two real-valued vectors can be well approximated by the Hamming distance (HD) using locality sensitive hashing (LSH) codes of the two vectors^{24,33,34}. Accordingly, in this work, instead of being stored in a dynamic random access memory (DRAM) for future distance calculations, the features are hashed into binary/ternary signatures in a crossbar with randomly distributed conductance at each crosspoint exploiting the stochasticity of memristor devices. **Finally**, those signatures are then searched against those previously stored in another crossbar that acts as a content-addressable memory enabled by a newly proposed coding method, from which we can also calculate the degree of mismatch that approximates the cosine distance of the original real-valued vector. The three steps denoted above are experimentally demonstrated

here.

The idea is experimentally demonstrated in our integrated memristor chip. One of the tiled 64×64 memristor crossbars in our integrated chip that we used to experimentally implement the network is shown in Fig. 1b. The peripheral control circuits, including the driving, sensing, analog-to-digital conversions, and the access transistors, are implemented with a commercial 180 nm technology integrated chip (Fig. 1c). $50 \text{ nm} \times 50 \text{ nm}$ Ta/TaO_x/Pt memristors are integrated with back-end-of-the-line (BEOL) processing on top of the control peripheral circuits (1d). The fabrication details, the device characteristics and peripheral circuit designs were reported previously elsewhere^{35,36}, and the picture of our test chip and platform is shown in Supplementary Fig. 1, 2.

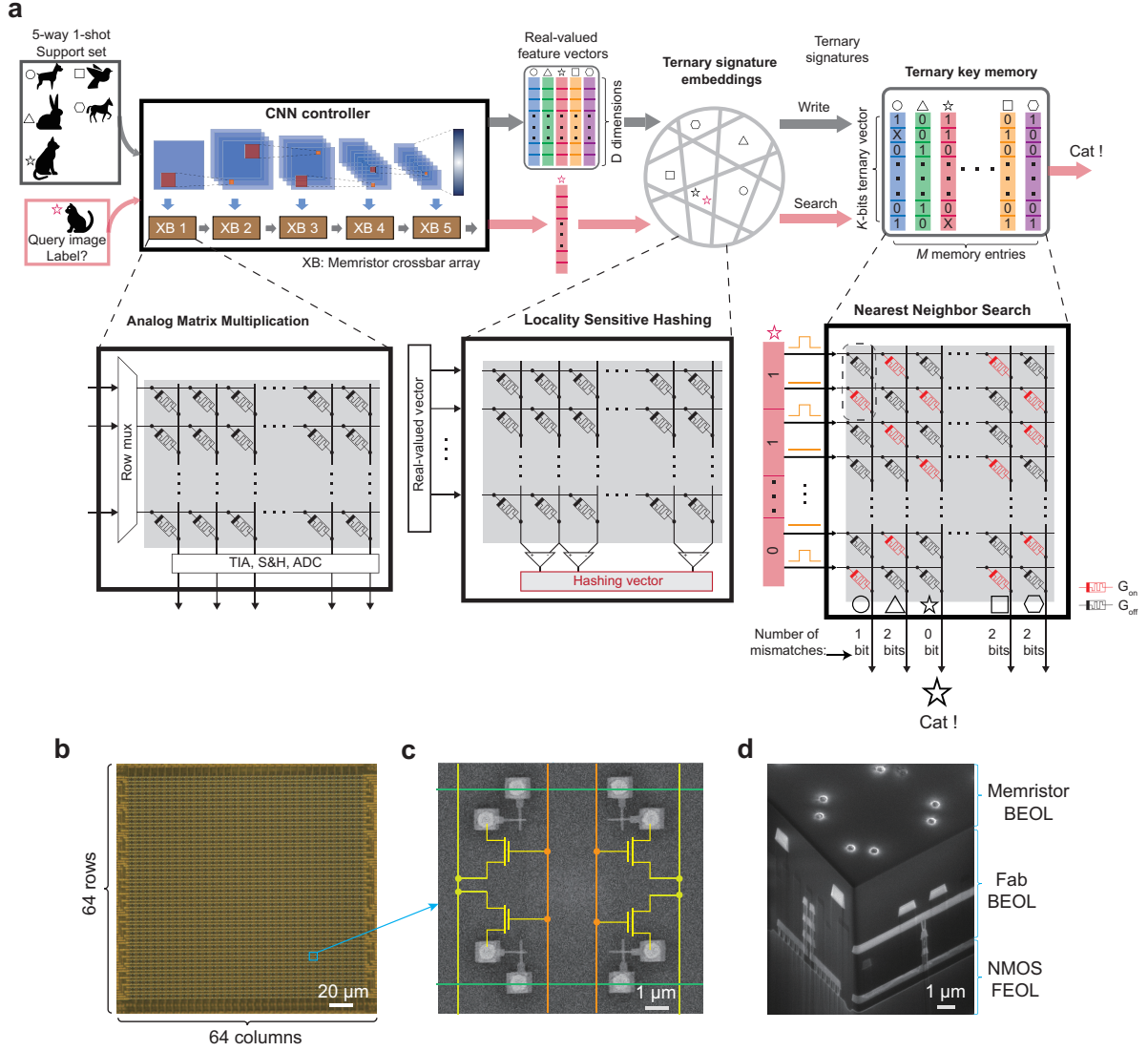


Figure 1: Memory augmented neural networks in crossbar arrays. **a**, The schematic of a crossbar-based MANN architecture. The expensive data transfer in a von Neumann architecture can be alleviated by performing analog matrix multiplication, locality sensitive hashing, and nearest neighbor searching directly in memristor crossbars, where the data is stored. **b**, Optical image of a 64×64 crossbar array in a fully integrated memristor chip. **c**, Top view of four $50 \text{ nm} \times 50 \text{ nm}$ integrated cross-point memristors. **d**, Cross-section of the memristor chip, where complementary metal-oxide-silicon (CMOS) circuits at the bottom, inter-connection in the middle, and metal vias on the surface for memristor integration with back-end processes. Animal figures in **a** are taken from www.flaticon.com

Locality sensitive hashing in crossbar array

Using crossbars for feature extraction has been implemented and well-discussed in prior works, therefore, we first discuss our proposals for performing hashing operations in crossbar-based hardware, by employing intrinsic stochasticity. We validate that the experimental implementation in memristor crossbars can approximate cosine distance. LSH^{37–39} is a hashing scheme that generates the same hashing bits with a high probability for the inputs that are close to each other. One of the hashing functions among the LSH family is implemented by random signed projections, *i.e.*, applying a set of hyperplanes to divide the Hilbert space into multiple parts such that similar inputs project to the same partition with a high probability (Fig. 2a). This random projection is mathematically expressed by a dot product of the input vector $\vec{a}$ and a random normal vector $\vec{n}$, so that ‘1’ is generated if $\vec{a} \cdot \vec{n} > 0$, and ‘0’ otherwise. Accordingly, LSH bits can be calculated by Equation 1 in a matrix form,

$$\vec{h} = H(\vec{a} \mathbf{N}) \quad (1)$$

where the $\vec{h}$ is the binarized hashing vector, $\vec{a}$ the input real-valued feature vector, $\mathbf{N}$ the random projection matrix with each column a random vector, and H the Heaviside step function.

The random projection matrix can be constructed physically by exploiting the stochastic programming dynamics of the memristor devices or the initial randomness after the fabrication (Fig. 2b). But it is still challenging to generate random vectors $\vec{n}$ with a zero mean value, as required by the LSH algorithm, because the conductance of the memristor device can only be positive values. Our solution is to take the difference between devices in the adjacent columns⁴⁰ in the crossbar array. The devices from the columns, assuming no interference in between, are independent of each other. Therefore, the distribution of the conductance difference will also be uncorrelated and random. In this way, the random normal vector $\vec{n}$ in the original equation can be represented by the difference of two conductance vectors, *i.e.* $\vec{n} = (\vec{g}^+ - \vec{g}^-)/k$. The zero mean value of the vector $\vec{n}$ is guaranteed as long as the distribution of the memristor conductance vectors $(\vec{g}^+, \vec{g}^-)$ have the same mean value. k is a scaling factor, which can be set as an arbitrary value because we only need to determine if the output is larger than zero or otherwise.

Here, we experimentally program all devices in a crossbar array to the same target conductance state. The programming process of memristor devices is stochastic⁹ as the thinnest part of the conductive filament can be only a few atoms wide⁴¹. Accordingly, the final conductance values follow a random distribution with the mean roughly matching the target conductance. To lower the output current and thus the energy consumption, we reset all devices to the low conductance state (the lowest conductance was measured 17 nS at the read voltage of 0.2 V) from arbitrary initial states using a few pulses (see Methods for details). After programming, as expected, devices are

programmed to a low conductance state (Fig. 2c), and the difference between devices from adjacent columns follows a random distribution with a zero mean value (Fig. 2d, e). Hashing bits for an input feature vector are generated efficiently by performing multiplications with the randomly configured memristor crossbar array. After converting the real-valued input vector into the analog voltage vector, the dot product operations are conducted by applying the voltage vectors to the row wires of the crossbar and reading the current from the column wires. Thus, the hashing operation is completed by comparing the current amplitude from the adjacent columns (Fig. 2b) in one step, regardless of the vector dimension.

Imperfections in emerging memristive devices, such as conductance relaxation and fluctuation, limit experimental performance. This is mainly because the device conductance fluctuations incur instability of hashing planes implemented as adjacent column pairs in crossbar arrays. This causes hashing bits for input vectors that are close to hyperplanes to flip between 0 and 1 over time (Supplementary Fig. 5) and therefore leads to an inaccurate approximation of the cosine distance. It should be emphasized that this problem only becomes apparent when performing matrix multiplication in arrays, rather than multiplying with the readout conductances reported in other works²³.

To mitigate bit-flipping, we propose a software-hardware co-optimized ternary locality sensitive hashing scheme (TLSH). The scheme introduces a wildcard ‘X’ value to the hashing bits (Fig. 2a), in addition to ‘0’ and ‘1’ in the original hashing scheme. As the name implies, the Hamming distance between the wildcard ‘X’ and any values will always be zero. The ‘X’ is generated when the absolute value of the output current difference is smaller than a threshold value, *i.e.* I_{th} . The value for I_{th} is chosen to be small and close to the transient analog computing error from the crossbar, such that any bit-flipping minimally impacts the similarity search. One peripheral circuit design to realize said functionality is introduced in detail in the Supplementary Note 2. It should be noted that the speed and energy efficiency for hashing operations with the custom circuit is much higher than crossbars for matrix multiplication mainly because of the lack of analog-to-digital signal conversion.

We validate the proposed approach by conducting experiments in our integrated memristor crossbars. The hash outputs for 500 64-dimensional real-valued random vectors are computed in our memristor crossbars for binary and ternary hashing vectors. The I_{th} representing the threshold of the ‘X’ wildcard is set to 4 μ A in the ternary hashing implementation (TLSH). Fig. 2f shows that the cosine distance is closely correlated with the Hamming distance between the hashed vectors with 128 hashing bits in total, regardless of whether the hashing codes are generated by a 32-bit floating-point digital processor (“software LSH” in Fig. 2f), the analog crossbar (“hardware

LSH”), or the proposed co-designed ternary hashing codes by the crossbar (“hardware TLSH”). Note that the Hamming distance of the ternary hashing codes is smaller than that of the binary codes because the distance to a wild card ‘X’ is always zero. The effectiveness of the method is evaluated quantitatively by the linear correlation coefficient, as shown in Fig. 2g. The result shows that the proposed ternary hashing narrows the already small gap between the digital software approach and our analog hardware approach. The performance improvement results from significantly reduced unstable bits, which is experimentally demonstrated by the comparison shown in Supplementary Fig. 5. The results demonstrate that crossbar arrays, utilizing the proposed ternary scheme, can effectively and efficiently perform hashing operations, taking advantage of intrinsic stochasticity and massively parallel in-memory computing.

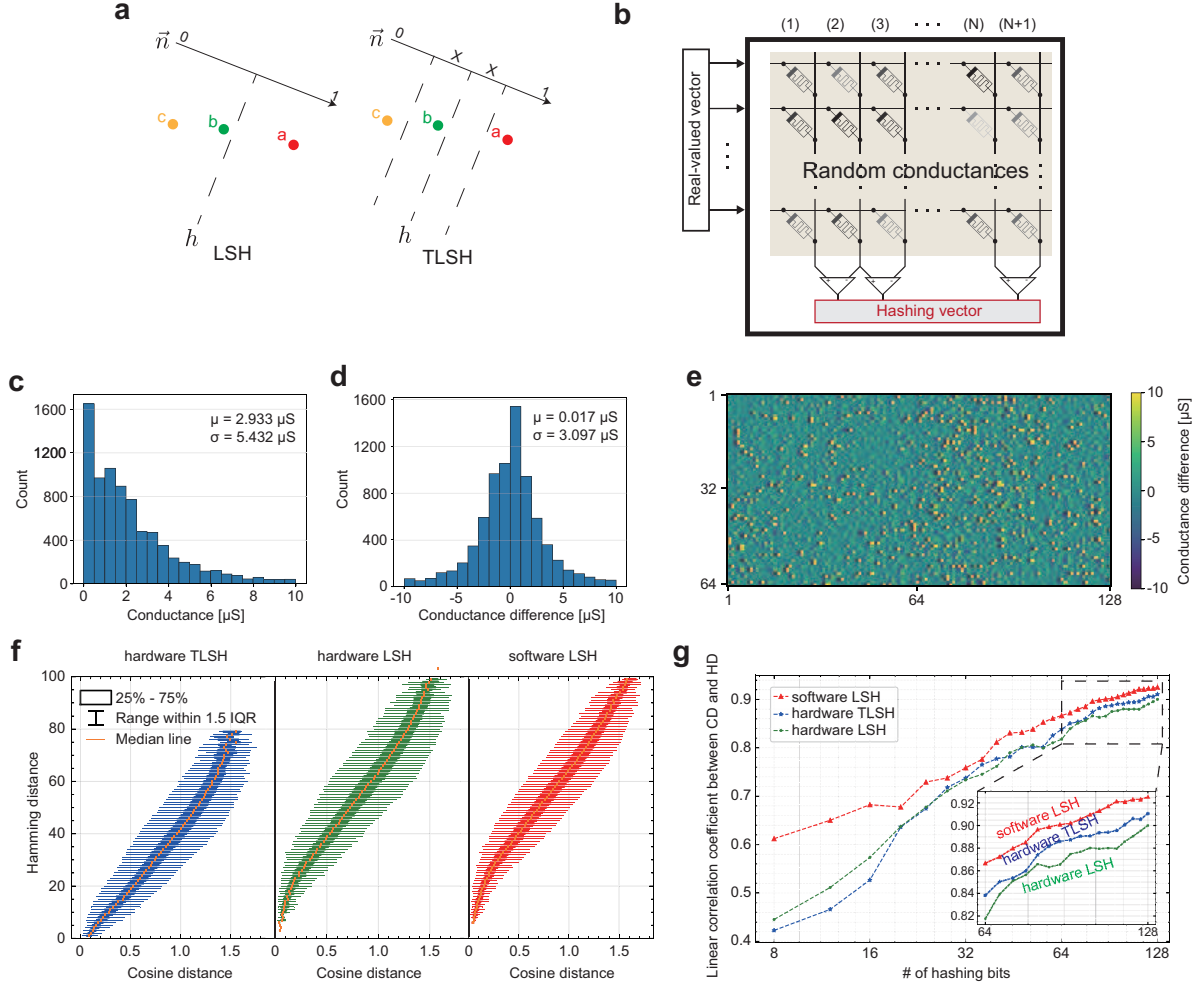


Figure 2: Robust ternary locality sensitive hashing in analog memristive crossbars. **a**, Illustration of the Locality Sensitive Hashing (LSH) and the Ternary Locality Sensitive Hashing (TLSH) concept. **b**, The LSH or TLSH implemented in memristor crossbars. Each adjacent column pair represents one hashing plane. So, crossbars with $N + 1$ columns can generate N hashing bits with this method. Greyscale colors on the memristor symbol represent random conductance states. **c**, A random memristor conductance distribution in a 64×129 crossbar after applying five RESET pulses to each device. The intrinsic stochastic behavior in memristor devices results in a lognormal-like distribution near $0 \mu\text{S}$. **d**, The distribution of the memristor conductance difference for devices in adjacent columns. The differential conductance distribution is random with zero-mean, matching the requirements of our hashing scheme. **e**, The conductance difference map of size 64×128 (including three crossbar arrays each of size 64×64). **f**, The correlation between cosine distance and Hamming distance with different hashing representations shows that the Hamming distance generated by both hardware and software can well approximate the cosine distance. IQR, interquartile range. **g**, The linear correlation coefficient between Hamming distance and cosine distance increases with the number of total hashing bits. The hardware TLSH approach shows a higher correlation coefficient than the hardware LSH approach due to the reduced number of unstable bits, as detailed in Supplementary Fig. 5.

TCAM in crossbars with ability to output degree of mismatches

Following the LSH step, the binary or ternary hashing signatures will be searched against the hashed signatures previously stored in a memory to calculate the similarity and thus find the k -closest matches. As mentioned earlier, this is an extremely time- and energy-consuming step on conventional hardware such as GPUs. Content addressable memories (CAM) or the ternary version (TCAM) are direct hardware approaches that can find the exact match in the memory in one step. Still, existing static random access memory (SRAM) based CAM/TCAM implementations limit the available memory capacity and incur high power consumption. CAMs/TCAMs based on non-volatile memories have been developed recently, including those based on memristor/ReRAM (*e.g.* 2T-2R^{26,42}, 2.5T-1R⁴³), floating gate transistor (*e.g.* 2Flash⁴⁴), ferroelectric transistors (*e.g.* 2FeFET^{24,25}), *etc.* Although these studies demonstrated good energy efficiency, they are limited to at most a few bits mismatches which have difficulties serving as attentional memory modules for scaled-up MANNs^{24–26}.

We implement the TCAM functionality directly in an analog crossbar with the additional ability to output the degree of mismatch based on the Hamming distance, rather than only a binary match/mismatch. In contrast to conventional TCAM implementations which sense a mismatch by a discharged match-line, our crossbar-based TCAM searches through a simple encoding and a set of dot product operations computed in the output currents. Fig. 3a shows a schematic on how this scheme works. First, the query signature is encoded to use a pair of voltage inputs for 1-ternary-bit, so that one column wire is driven to a high voltage (*i.e.* , V_{search}), while the other is grounded. The corresponding memristor conductances that store previous signatures are encoded with one device set to a high conductance state (*i.e.* , G_{on}) and the other to the lowest conductance state (*i.e.* , $G_{\text{off}} \approx 0$). In this way, for a ‘match’ case, the high voltage will be applied to the device in the low conductance state, and therefore, very little current is added to the row wires. In a ‘mismatch’ case, the high voltage applied to the device in the high conductance state will contribute $V_{\text{search}} \times G_{\text{on}}$ to the output current of the column wires. The wildcard ‘X’ in the ternary implementation will be naturally encoded as two low voltages as input or two low conductance devices so that they contribute zero or very little current and thus always yield ‘match’. In this way, the degree of mismatch, the Hamming distance, between the query signature and all words stored in the crossbar is computed in a constant time step by sensing the column currents from the crossbar (see Fig. 3b). To minimize the energy consumption, we custom-designed two peripheral circuit approaches for the crossbar-based TCAM, as detailed in Supplementary Note 3, significantly reducing the energy consumed on memristors.

We have experimentally implemented the above TCAM for measuring Hamming distance in

memristive crossbars. First, eight different binary signatures, each having eight bits but a different number of ‘1’s (from one ‘1’ to eight ‘1’s), are encoded into conductance values as shown in Fig. 3c. The conductance values are then programmed to a crossbar with an iterative write-and-verify method (see ref.³⁵ and Methods for details), with the readout conductance matrix after successful programming shown in Fig. 3b. We choose $150\ \mu\text{S}$ as the G_{on} for a higher on/off conductance ratio and minimal relaxation drift (Supplementary Fig. 3). Fig. 3d shows both the distribution of G_{on} and G_{off} after programming.

After configuring the memory to store the previously generated signatures, 100 ternary signature vectors as queries are randomly chosen and the corresponding encoded voltages are applied to the column wires of the crossbar (Fig. 3c), to perform the search operation. The search voltage (V_{search}) is chosen to be $0.2\ \text{V}$ in this work, so each mismatched bit will contribute approximately $30\ \mu\text{A}$ ($=0.2\ \text{V} \times 150\ \mu\text{S}$) to the output current. In the experiment, however, results are deteriorated by non-ideal factors. For example, the memristor in a low conductance state still contributes a small current ($V_{\text{search}} \times G_{\text{off}} \neq 0$) in a ‘match’ case, imperfect programming of G_{on} results in deviations in output current for each ‘mismatch’ case, *etc.* Our device exhibits a large enough conductance on/off ratio³⁵, but for devices with a lower conductance on/off ratio, such as MRAM⁴⁵, the problem would be more significant. For such cases, we propose a 3-bit encoding that is discussed in detail in Supplementary Fig. 6. Additionally, non-zero wire resistances cause a voltage drop along wires, lowering the output current from what would be ideally expected. Despite these factors, the output current in our experiments exhibits a linear dependence on the number of mismatch bits, *i.e.* ternary Hamming distance (Fig. 3e). Fig. 3f shows separated distributions where each distribution represents a distinct number of mismatch bits ranging from 0 to 8. We have thus experimentally demonstrated a robust capability to store patterns, search patterns, and obtain the degree of mismatch which will enable determining the closest match that is stored in an array by simply comparing output currents.

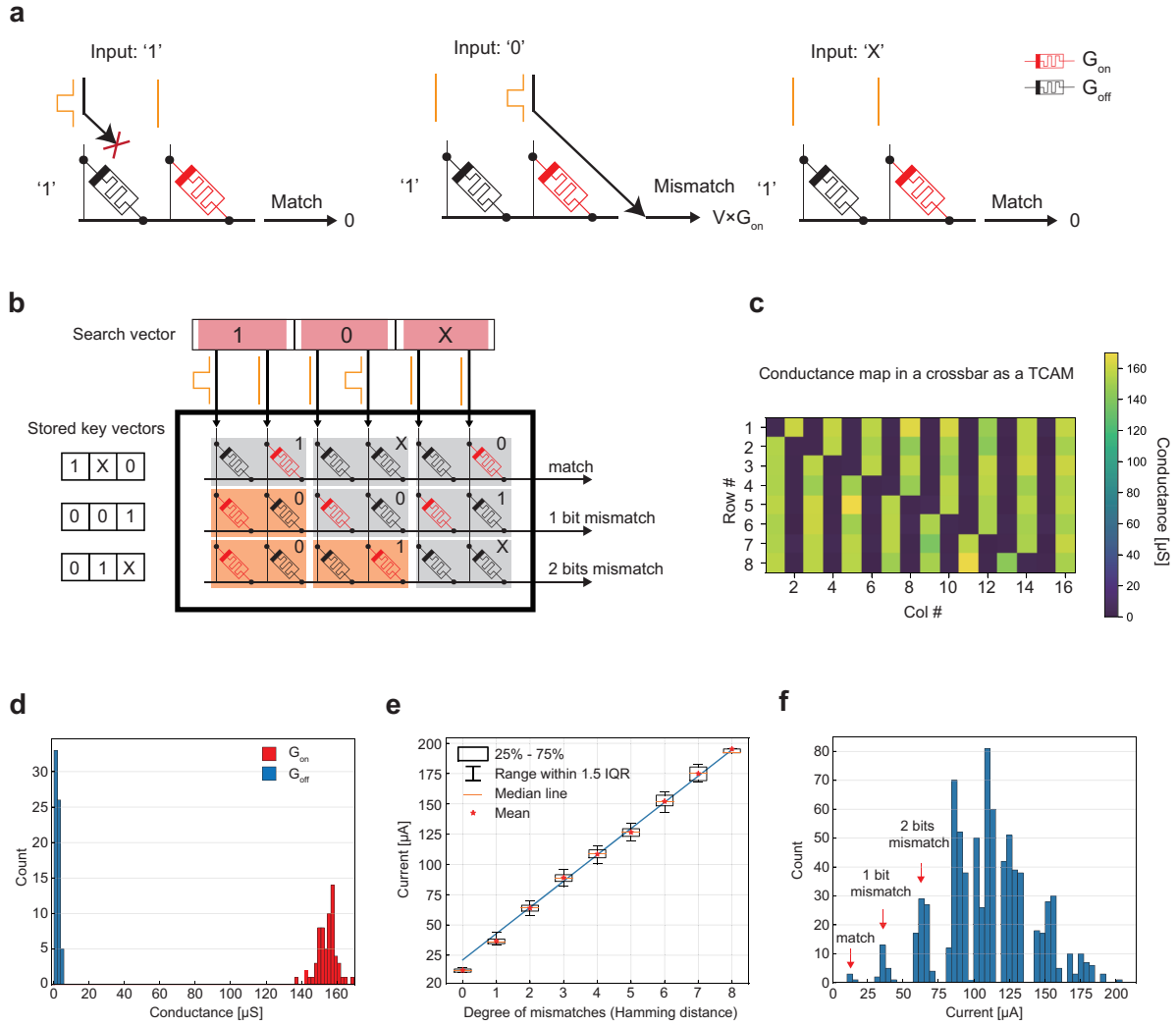


Figure 3: TCAM implemented in crossbar array capable of conducting Hamming distance calculation. **a**, Illustration of the basic principle for using dot product to distinguish 'match' and 'mismatch' cases. **b**, The schematic of calculating Hamming distance in a crossbar. The figure shows three 3-dimensional ternary key vectors stored in a 3×6 crossbar with a differential encoding. Differential voltages representing ternary bits in search vectors are applied to the source line and the output current from the bit line can represent the THD between the search vector and keys stored in the memory. **c**, The readout conductance map after eight binary vectors experimentally stored in the crossbar as memory. In the experiment, we set G_{on} as $150 \mu S$ and V_{search} as $0.2 V$. **d**, Distribution of G_{on} and G_{off} . **e**, Output current shows a linear relation with Hamming distance measuring the degree of mismatches. IQR, interquartile range. **f**, Current distributions are separated from each other through which we can obtain the number of mismatch bits (i.e., Hamming distance).

One- and few-shot learning experiment fully implemented in memristor hardware

We implemented the key components in a MANN to demonstrate the feasibility of one-shot and few-shot learning in crossbars. To evaluate and compare the performance of our method, we chose the Omniglot dataset²⁷, a commonly used benchmark for few-shot learning tasks. In this dataset, there are 1,623 different handwritten characters (classes) from various alphabets. Each character contains 20 handwritten samples from different people. Samples from 964 randomly chosen classes are used to train the CNN controller and the remaining 659 are used for one-shot and few-shot learning experiments. In an N -way K -shot learning experiment, the model should be able to learn to classify new samples from N different characters (classes) after being shown K handwritten images from each character (support set). The accuracy is evaluated by classifying an unseen sample (query set) after learning from the limited number of samples (only one sample each for the 1-shot problem) from each class.

In our experiment, the memristor CNN controller first extracts the feature vector from an image. Note that the weights in the CNN don't need to be updated after the meta-training process which is done in software offline. Our CNN consists of four convolutional layers, two max-pooling layers, and one fully connected layer (Fig. 4a). There are nearly 65,000 weights in convolutional layers altogether that are represented by 130,000 memristors, with the conductance difference of two memristors representing one weight value. The weights of convolutional layers are flattened and concatenated first (see Supplementary Fig. 9 and Methods for details) and then programmed to crossbar arrays with an iterative write-and-verify method. Limited by the available array size, we divide larger matrices into 64×64 tiles and reprogram the same arrays when needed to accomplish all convolutional operations in the crossbar. Experimental conductance maps (36 matrices of conductance values) for the CNN layers after each programming of an array are shown in Supplementary Fig. 10. The repeated programming of memristor arrays demonstrated good reliability of the memristor devices within crossbars. After programming the convolutional weights into the crossbar, the fully connected layer is retrained to adapt to the hardware defects.

Before a few-shot learning task is performed, the explicit memory stored in the crossbar-based TCAM is initialized with all '0's. During few-shot learning, the feature vectors, computed by the memristor CNN, are first hashed into 128 binary or ternary signatures and then the signatures are searched against the entries in the crossbar-based TCAM for the closest match, as described above. The label of the closest match will be the classification result. If correct, the nearest neighbor entry will be updated based on the new input query vector (see Methods). Otherwise, the signature along with the label is written to a new location in the TCAM using differential encoding. After learning K images from the support set, The conductance map that is stored in the crossbar-

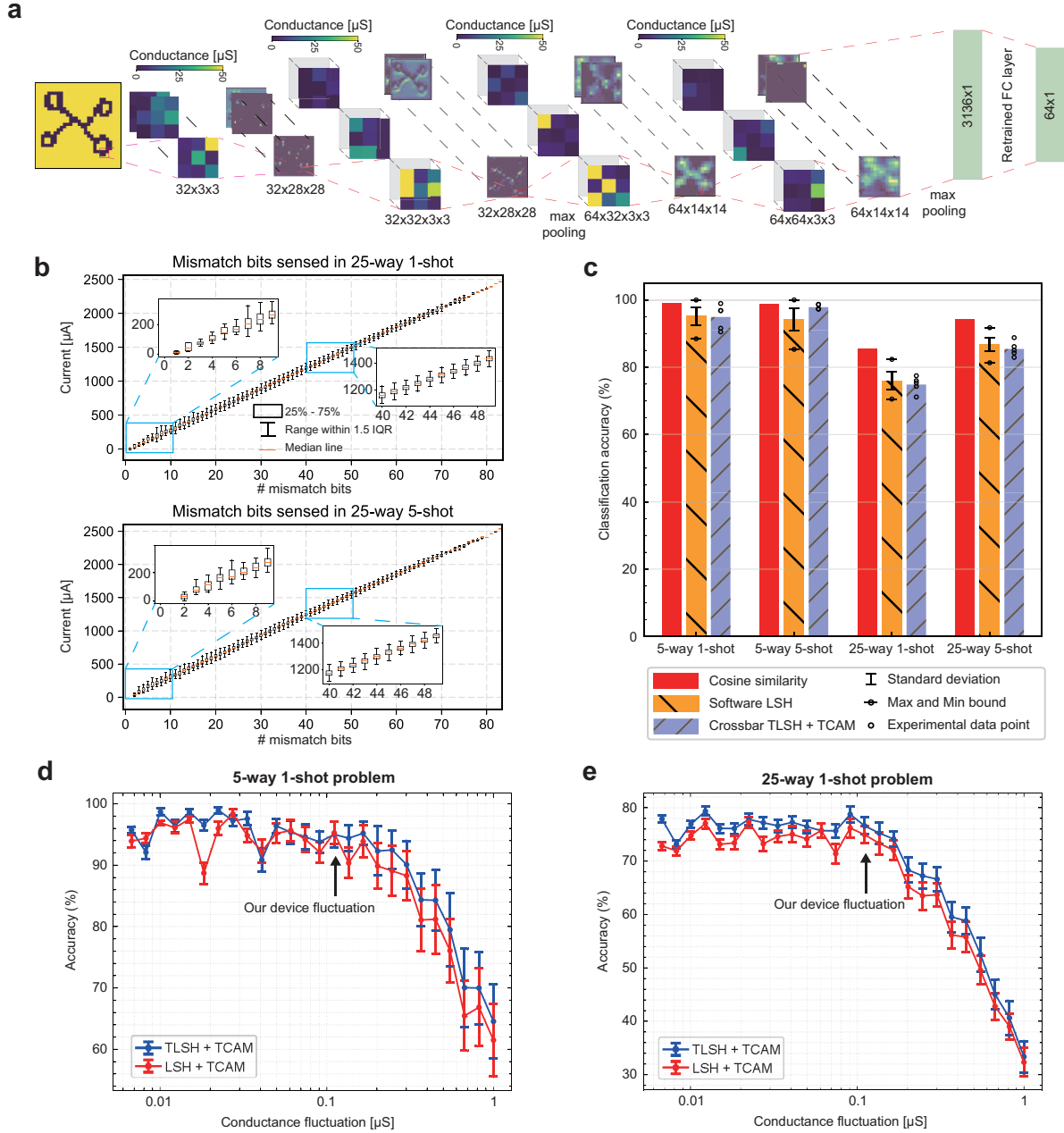


Figure 4: End-to-end experimental inference with memristive crossbar arrays. **a**, Schematic of CNN structure implemented in the memristor crossbar array. The conductance shows the weight mapping of CNN kernels. The format of dimension representations in the figure follows the Output channel (O), Input channel (I), Height (H), and Width (W). The conductance maps representing the whole CNN kernels are shown in Supplementary Fig. 10. **b**, Linear relationship between the sensing current from the crossbar-based TCAM and the number of mismatch bits during the search operations. **c**, Classification accuracy with cosine similarity, software-based LSH with 128 bits, and end-to-end experimental results on crossbar arrays. We provide 5 experimental data points for each task. Software LSH shows experimental variation due to different initializations of the hashing planes in each experiment. **d**, **e**, Simulations of classification accuracy of 5-way 1-shot problem (**d**) and 25-way 1-shot problem (**e**) as a function of device fluctuations in the memristor model for both TLSH and LSH. Fluctuations from nearly zero to 1 μS are shown. The actual experimental fluctuation level is shown with an arrow.

based TCAM is shown in Supplementary Fig. 11. Note that the CNN controller stays the same across all four few-shot learning tasks (5-way 1-shot, 5-way 5-shot, 25-way 1-shot, and 25-way 5-shot) once trained on the entire dataset. Therefore, in a future system with more crossbar tiles that can accommodate the CNN model, the CNN controller would not need to be re-programmed, and accordingly, the memory only needs to be updated during lifelong learning. Moreover, even for the memory module, the update is not frequent (1.3 times per bit for 20-shot) throughout the learning process, as demonstrated in Supplementary Fig. 12. Considering a conservative statistical endurance (about 10^6) for a relatively high switching window reported previously^{46,47}, we estimate about 4000-years lifetime of a face recognition system that sees each face 10 times in one day which is much longer than human life.

Accuracy is evaluated experimentally in classifying new samples after few-shot learning with four standard tasks: 5-way 1-shot, 5-way 5-shot, 25-way 1-shot, and 25-way 5-shot, respectively. We found that the experimental sensing currents during few-shot learning experiment are highly linear with the number of mismatch bits, *i.e.* hamming distance, as shown in Fig. 4b. This is partly enabled by the introduced wildcard ‘X’ from our TLSH method, as discussed in detail in Supplementary Fig. 16 and Supplementary Note 1. The classification results shown in Fig. 4c demonstrate that for 5-way problems, our crossbar-based MANN achieves an accuracy very close to the software baseline implemented in digital hardware with cosine similarity as the distance metric. For 25-way problems, we find no difference between results from our analog hardware and that from the digital hardware implementing the same LSH plus Hamming distance algorithm. Though there exists some accuracy drop compared to the cosine baseline, the performance can be improved to match the baseline accuracy by increasing the number of hashing bits from 128 to 512 (see Supplementary Fig. 7). The experimental results demonstrate that the MANN implemented in crossbar arrays can achieve similar accuracy as our software baseline and recently reported state-of-art model, as compared in detail in Supplementary Note 5

Device imperfections analysis

Accuracy can be affected by many non-idealities in emerging memory devices, among which the two most prominent are conductance fluctuations and relaxation. We noted that the conductance of memristor devices fluctuates up and down (see Supplementary Fig. 4a, b) even within a very small period (at the scale of nanoseconds). The fluctuation leads to frequent changes in convolutional kernels, hyperplane locations in LSH operations, and stored signature values in TCAMs, which negatively impact the accuracy. The data (shown in Supplementary Fig. 4c) measured from our integrated array shows that the degree of device fluctuation increases with the conductance value. This behavior is consistent with previous reports on single device measurement^{35,48}. In addition

to the conductance fluctuations, the programmed value may also change permanently (relaxation) over time, which is characterized in detail in Supplementary Fig. 3c. From these results, we find that conductance relaxation is larger when device conductance is programmed to a certain range (from around 25 μS to 75 μS). Therefore, in our implementation, we try to avoid this range as much as possible to achieve the software equivalent accuracy. For example, in the LSH part, we chose lower conductance levels to minimize both conductance fluctuation and relaxation. In the TCAM part, we chose 0 μS and 150 μS as the low and high conductance levels to minimize the impact of conductance relaxation. In the CNN part, we observe that most weight values are very small (near zero), so with the differential encoding method (details described in Methods), we can guarantee that most memristor conductance values are below the range with higher relaxation.

To analyze our software-competitive accuracy results and evaluate if our method is scalable, we built an empirical model describing experimental conductance-dependent fluctuation behavior and deviation after programming⁴⁹. With the experimental calibrated model (see Methods and Supplementary Fig. 4c, d for more details), we can match the simulation results with the experiments in few-shot learning on Omniglot handwritten images. The detailed comparison is shown in Supplementary Fig. 13a). The simulation also enables us to analyze how different device fluctuations impact classification accuracy. We conducted simulations assuming the device conductance fluctuation spanning from nearly no fluctuation-to- 1 μS which is about ten times larger than our device behavior. The results in Fig. 4e, f show that with the experimental fluctuation value, the accuracy stays almost the same as the software equivalent value, but the accuracy will sharply drop if the fluctuation is more than three times larger than our experimental value. The results also show that our proposed TLSH method exhibits better performance compared to the conventional LSH, especially for more significant device fluctuation scenarios (Fig. 4e, f and Supplementary Fig. 14). In addition to the higher tolerance to the device fluctuation, the comparison shown in Supplementary Fig. 13b, c also demonstrates the TLSH’s advantages in search energy. These simulations, with experimental calibration, elucidate the experimentally observed defect tolerance and software-equivalent accuracy. Though there exist other defects such as stuck-at-fault, I/V nonlinearities for high resistance states, and device-to-device variation in active conductance range, we found these have negligible impact on the final performance. With this tool, we are able to analyze scaling up to more complex real-world problems.

Scaled-up MANN for Mini-ImageNet

The methodology of crossbar-based TLSH and TCAM can be applied in many fields of deep learning that require distance calculation and attention mechanisms. To show the scalability of our crossbar-based MANN, we conducted simulations based on our experimentally-calibrated model

for one-shot learning using the Mini-ImageNet dataset²⁸. This dataset is derived from the ImageNet dataset with 100 classes of 600 images of size 84×84 color pixels per class. The task is known to be much more difficult than that of the Omniglot handwritten dataset. A more sophisticated ResNet-18 model is used as the controller following the state-of-the-art structure in few-shot learning models⁵⁰, which has more than 11 million weights, 44 times larger than the controller used to classify images from the Omniglot dataset.

A challenge for this network is the required crossbar sizes (larger than 512 in one dimension), and thus the voltage drops along the wire would significantly reduce the computing accuracy. This is solved by partitioning large arrays (for hyperplanes and memories) into smaller 256×256 tiled crossbars (Fig. 5a) to accommodate the model. In the simulation, we consider the experimental device fluctuations (see Supplementary Table 2) and use the same threshold current ($4\mu\text{A}$) for the TLSH approach as in the smaller Omniglot problem. The result in Fig. 5b shows that the classification accuracy for the 5-way 1-shot problem increases with the number of hashing bits, and reaches 58.7% with 4,096 hashing bits, only 1.3% smaller than the model implemented in digital hardware with cosine similarity as the distance metric. We also explored the performance with different partitioned array sizes in Supplementary Fig. 15a, b, which achieves nearly equivalent performance with arrays smaller than or equal to 256×256 , and drops slightly with the 512×512 array. Encouragingly, the TLSH function can be implemented with a larger array (512×512) because of lower conductance and smaller voltage drops along the wires. From these results, we can see that the performance of our crossbar-based MANN can scale up effectively to at least Mini-ImageNet problems.

Discussion

Compared to conventional von Neumann based implementations, the key advantage of crossbar-based MANNs is lower latency and higher energy efficiency through co-located computing and memory, energy-efficient analog operations, and intrinsic stochasticity. To evaluate the strength of the approach, we run the same 5-way 1-shot problem with Omniglot and Mini-ImageNet datasets on a digital graphic processing unit (GPU) (Nvidia Tesla P100). The time required to classify a single image increases dramatically after the size of the MANN’s external memory capacity reaches a certain threshold (only several MB) because of the repeated off-chip data movement (see Fig. 5c). This problem on conventional hardware has been the major bottleneck preventing the widespread adoption of few-shot learning. The approach of directly computing in the memory, or crossbar, provides a plausible solution to address this bottleneck. In the crossbar-based MANNs, the matrix multiplication in the convolutional layer, the hashing in TLSH, and the searching operation in TCAM are all computed with single-step current readout operations. With our current proof-of-

concept experimental system, readouts take about 100 ns, but in a future system with more crossbar tiles that are fabricated with a more advanced technology node, the readout time can be reduced to 10 ns. We also considered the time latency for the peripherals that include but are not limited to the digital-to-analog converters (DACs), TLSH sensing block, and the analog adder for summing up the voltages in different tiled crossbar arrays. With these forecasts, we compared the latency of the nearest neighbor search operation on a GPU with our analog in-memory hardware. The results shown in Fig. 5d indicate latency improvements of $(10,466\times$ for Omniglot and $26,002\times$ for Mini-ImageNet) when the memory size (number of entries) is 8,192. Additionally, our approach also offers high energy efficiency compared with the conventional GPU ($2,857\times$ for Omniglot and $50,970\times$ for Mini-ImageNet) in the forecasted system. Detailed analysis about energy and latency estimations can be found in Supplementary Note 4.

In summary, we have experimentally demonstrated the viability of a complete MANN architecture, from the controller to distance calculation, in an analog in-memory platform with proven high robustness and scalability. We utilize the analog behavior of memristor devices to perform convolution operations for CNNs and exploit the inherent stochasticity of devices to perform hashing functions. A novel hardware-friendly hashing function (TLSH) is developed to provide better analog computing error tolerance and lower power consumption. In addition, a differential encoding method for a crossbar-based TCAM is applied to adapt to the ternary Hamming distance calculations requirements. In our experiment, all dot-product operations are performed in physical crossbars, which exhibit experimental imperfections, such as device state fluctuations, device nonlinearities, voltage drops due to wire resistance, and peripheral circuits. The hardware-implemented MANNs delivered similar accuracy compared to software on few-shot learning with the widely used Omniglot dataset. Simulation results on Mini-ImageNet show the ability of crossbar-based MANNs to execute real-world tasks, with much-improved latency and energy consumption. We demonstrate that analog in-memory computing with memristive crossbars efficiently supports many different tasks, including convolution, hashing, and content-based searching. The successful demonstration of these functions opens possibilities with other machine learning algorithms, such as attention-based algorithms, or reaching scales that are currently prohibited by conventional hardware (e.g, Fig. 5c). Additionally, there are many opportunities for future software-hardware co-optimization to improve the accuracy and efficiency results further.

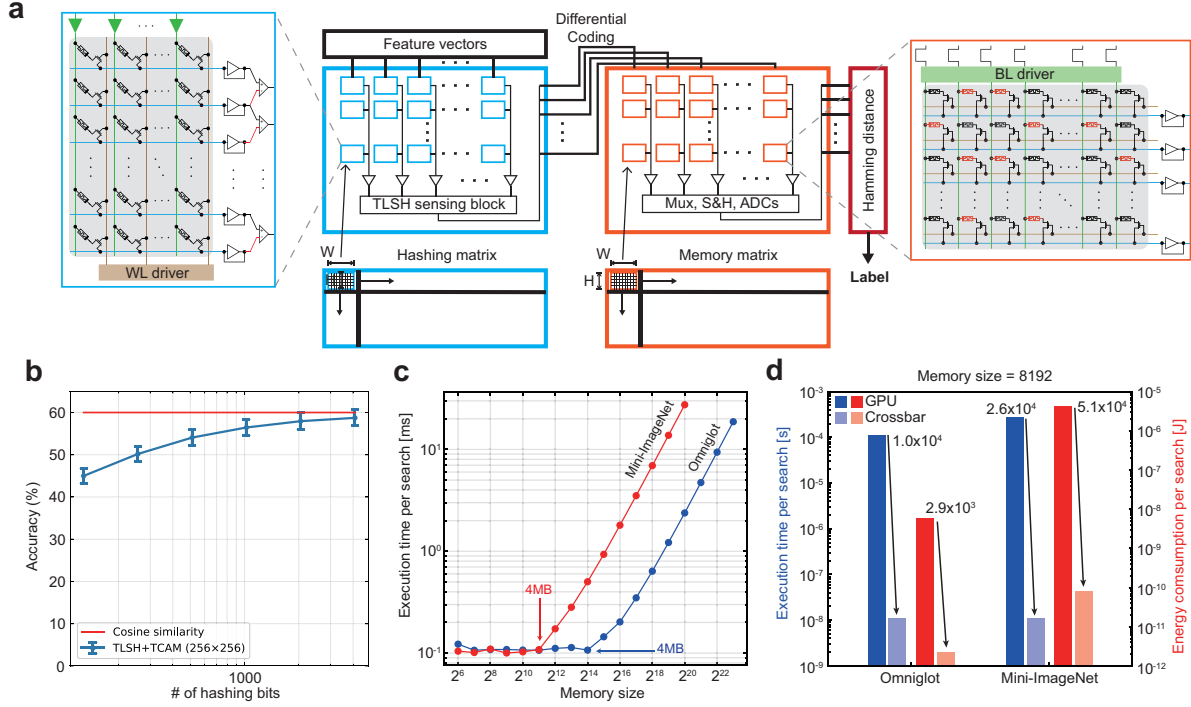


Figure 5: Experiment validated simulation results of Mini-ImageNet dataset. **a**, The architecture of TLSH and TCAM for the scaled-up MANN. Both matrices for hashing and external memory are partitioned into $H \times W$ memristor crossbar tiles, to mitigate the voltage drop problem in large crossbars and to increase the utilization rate. **b**, The accuracy performance from our experiment-validated models on Mini-ImageNet dataset. The error bar shows the 95% confidence interval among 100 repeated inference experiments. **c**, The execution time of search operations per inference on a GPU drastically increases when external memory size reaches a threshold, confirming the operation is memory intensive. **d**, The comparison of the search latency and energy consumption for 5-way 1-shot learning on both the Omniglot and Mini-ImageNet datasets. For GPU, the models for both datasets stores the same number of entries (8,192), but Mini-ImageNet uses a larger memory capacity due to the higher dimension (64 vs. 512) of feature vectors, leading to even better improvement on latency and energy efficiency. The number of hashing bits used in crossbar arrays is 128 and 4096 for Omniglot and Mini-ImageNet, respectively.

Methods

Memristor integration The memristors are monolithically integrated on CMOS fabricated in a commercial foundry in a 180 nm technology node. The integration starts with the removal of native oxide on the surface metal with reactive ion etching (RIE) and a buffered oxide etch (BOE) dip. Chromium and platinum are then sputtered and patterned with e-beam lithography as the bottom electrode, followed by reactively sputtered 2 nm tantalum oxide as the switching layer and sputtered tantalum metal as the top electrode. The device stack is finalized by sputtered platinum for passivation and improved electrical conduction.

Iterative write-and-verify programming method In this work, we use the iterative write-and-verify method to program memristor devices to the target conductance value. First, we set a target conductance matrix and the corresponding tolerant programming error range. After that, successive SET and RESET pulses are applied to the target devices followed by conductance readout with READ pulses. If the device conductance is below the target conductance minus the tolerant error, a SET pulse is applied. A RESET pulse is applied for conductance above the tolerant values, while the device has been programmed within the tolerant values are skipped to pertain the state. For the crossbar-based MANN in this work, we apply the write-and-verify method to map the weights of the CNN controller and memories in the TCAM structure. During the programming process, we gradually increase the programming voltage and gate voltage as shown in Supplementary Table 1. The pulse width for both the SET and RESET process is 1 μ s. The tolerant range we set is 5 μ S above or below the target conductance value.

Adjacent Connection Matrix We apply the Adjacent Connection Matrix (ACM)⁴⁰ to map the conductance of memristors in crossbar arrays to weights in hashing planes. ACM subtracts the neighboring columns as shown in Fig. 2b to generate the hash codes. Hence, for a crossbar array with N+1 columns, the output of differential encoding contains N values which immensely saves the area. The mathematical representation is as follows: Provided that we get a random conductance map after programming which is:

$$G_{map} = \begin{pmatrix} G_{1,1} & G_{1,2} & \cdots & G_{1,N} \\ G_{2,1} & G_{2,2} & \cdots & G_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ G_{N,1} & G_{N,2} & \cdots & G_{N,N} \end{pmatrix} \quad (2)$$

then the ACM method is equivalent to multiplying G_{map} by a transformation matrix:

$$G_{hash} = G_{map} \times \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ -1 & 1 & \cdots & 0 & 0 \\ 0 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & -1 & 1 \\ 0 & 0 & \cdots & 0 & -1 \end{pmatrix} = \begin{pmatrix} G_{1,1} - G_{1,2} & G_{1,2} - G_{1,3} & \cdots & G_{1,N-1} - G_{1,N} \\ G_{2,1} - G_{2,2} & G_{2,2} - G_{2,3} & \cdots & G_{2,N-1} - G_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ G_{N,1} - G_{N,2} & G_{N,2} - G_{N,3} & \cdots & G_{N,N-1} - G_{N,N} \end{pmatrix} \quad (3)$$

Memristor model and simulations We build a memristor model to simulate the conductance fluctuation, which is the most dominant non-ideality of our crossbar-based MANN. The behavior of conductance fluctuation is assumed to be a Gaussian nature which is as follows:

$$G = G_0 + \sigma \cdot \mathcal{N}(0, 1) \quad (4)$$

where G_0 is the conductance after programming, σ describes the standard deviation of the fluctuation range. In the simulation we assume that the device only fluctuates for different VMM processes since in the real experiment the execution time of one VMM is very small (10 ns) which is negligible compared to time between successive input vectors (1 μ s). After considering the device-to-device variations and fitting the parameters (see Supplementary Table 2) to experimental measurements (see Supplementary Fig. 4), Equation 4 becomes:

$$G = G_0 + \exp(a \cdot \ln(G_0) + b + s \cdot \mathcal{N}(0, 1)) \cdot \mathcal{N}(0, 1) \quad (5)$$

with $\mathcal{N}(\mu, \sigma^2)$ being the normal distribution with mean μ and standard deviation σ . In the simulation, we also consider the program error to the initial conductance G_0 which is shown as:

$$G_0 = G_t + \mathcal{N}(0, \tilde{G}_{err}^2)$$

where G_t is the target conductance we want to program to and $\tilde{G}_{err}$ is the program error which we set to 5 μ S in the simulation.

To get the parameters of the memristor model in terms of the effect of device fluctuation, we SET 4,096 devices to 16 distinct analog states and READ each device for 1,000 times. The relation between the mean value and standard deviation of 1,000 reads is shown in Supplementary Fig. 4a and 4b. We further analyze the standard deviation distribution for each conductance state from 5 μ S to 50 μ S, plot the distributions in logarithmic scale, and fit them with Gaussian distribution. The results are shown in Supplementary Fig. 4c. The mean value of s for each distribution gives us

the parameter for the model. In addition, we fit a linear curve with conductance states and standard deviation in a log-log regime of measurements (see Supplementary Fig. 4d). The fitted parameters a and b are used in the simulation.

Ternary locality sensitive hashing Ternary locality-sensitive hashing introduces a wildcard 'X' to the hashing vector to alleviate the analog computing error from nonideal factors. We have demonstrated that this modified hashing scheme can achieve software-equivalent performance (LSH with the same hashing bits) on our crossbar arrays. The threshold current I_{th} applied in the experiment should be carefully chosen according to the typical value of the computing error caused by device fluctuation. The value we chose throughout the experiment is $4\mu A$. We also show the dependence of classification accuracy on different threshold currents in Supplementary Fig. 8.

For the simulation results in Fig. 4d and 4e, where the device fluctuation varies, we chose different threshold currents I_{th} according to the fluctuation levels. Specifically, for our memristor model which can be described by Equation 4, we empirically set the threshold current to be $5\sigma \cdot V_{in}$ where V_{in} is the maximum input voltage to the row line when performing VMM. The V_{in} is chosen to be 0.2 V in our experiment.

To generate random hashing planes in crossbar arrays (Fig. 2c), we *RESET* the devices from an arbitrary high conductance state to near $0\mu S$, where the conductance is ultimately decided by the intrinsic stochastic switching process. Regardless of the initial states, we use 5 *RESET* pulses with an amplitude of 1.5 V and a width of 20 ns. The *RESET* voltage is carefully controlled to protect memristor devices because larger voltages may cause devices to be stuck at low conductance states.

CNN architecture The convolutional neural network (CNN)s in crossbars is applied as the controller in the MANN to extract features from incoming images.

The CNN structure for the Omniglot dataset is composed of:

- 2 convolutional layers, each with 32 channels of shape 3x3
- A 2x2 max-pooling layer
- 2 convolutional layers, each with 64 channels of shape 3x3
- A 2x2 max-pooling layer
- A fully connected layer with 64 outputs

Each convolutional layer is followed by a rectified linear unit (ReLU) activation layer.

The ResNet-18 for the Mini-ImageNet dataset is composed of 8 residual blocks. Each residual block has two 3x3 convolutional layers with size $[layer\ channel, input\ channel, 3, 3]$ and $[layer\ channel, layer\ channel, 3, 3]$ respectively. Each convolutional layer is followed by a batch normalization layer and a ReLU layer. The overall architecture for ResNet-18 is:

- 1 convolutional layer with 64 channels of shape 3x3
- 2 residual blocks with 64 channels and stride 1
- 2 residual blocks with 128 channels and stride 2
- 2 residual blocks with 256 channels and stride 2
- 2 residual blocks with 512 channels and stride 2
- 1x1 adaptive average pooling layer

We map the weights of convolutional layers in the CNN to the conductance of memristor devices using differential encoding³¹. To elaborate, in a differential column pair, we program the positive weight to the left column and the absolute value of negative weights to the right column, while keeping the other at the low conductance state. The weight-to-conductance ratio we set in our experiment is 1:50 μ S. The feature maps collected from the output current in Fig. 4a are converted into voltages and then sent to another crossbar array corresponding to the subsequent convolutional layer. The fully connected layer is retrained after mapping convolutional layers on crossbar arrays and it's computed in the digital domain.

Memory update rules In an N -way K -shot learning, the memory module is updated based on the $N \times K$ images in the support set. If the label of the new input image label doesn't match the label of the nearest neighbor entry, we simply find a new location in the memory and write the input image to that location. Conversely, if the input image label matches, we need to update the memory of the nearest neighbor. In the GPU, we assign cosine distance as the metric to identify the label of input images and update the real-valued vectors at the same location³³. However, in this work, we use ternary Hamming distance as our metric, and we apply the following rules to update the ternary vectors in the external memory: We introduce a scoring vector to evaluate the majority of '1' and '0' for each bit of each memory vector. An element-wise mapping function is applied to each ternary vector stored in the memory module:

$$f : \{1, 0, X\}^D \rightarrow \{1, -1, 0\}^D$$

where D is the dimension of storing vectors. For example, vector $(1, 0, 1, X, 0)$ is mapped to $(1, -1, 1, 0, -1)$. We assume the scoring vector for each storing vector as: $s_i = f(a_i), i = 1, 2, 3, \dots, M$, where s_i is the scoring vector, a_i is the hashing vector stored in TCAM and M is the total number of memory entries. When there is a match case happening at memory location k , we first update its scoring vector as below:

$$s_k^* = s_k \oplus f(v) \quad (6)$$

where s_k^* is the new scoring vector, v is the hashing vector of the new image, $\oplus$ is element-wise add operation. Then we update the memory at the same location using the following rules:

$$a_i^* = L(a_i); L(x) = \begin{cases} 1 & x > 0 \\ X & x = 0 \\ 0 & x < 0 \end{cases} \quad (7)$$

where a_i^* is the updated memory and L is an element-wise operator. Therefore, bits of the memory stored in TCAM are decided by the majority of '1' and '0' of incoming vectors which match the storing vectors.

Omniglot training The Omniglot dataset contains 1623 different handwritten characters from 50 different alphabets. Each character was drawn online by 20 different people using Amazon's Mechanical Turk. In the experiment, we augment the 964 different characters in the training set to 3856 through rotation. The character types in the test set remain unchanged at 659. There is no overlap between the training set and the test set. We use the episode training method during the training process. Episode training is to select $N \times M$ instances from the training set during each training, where N represents different classes and M represents the number of instances in each class. The purpose of episode training is to enable the learned model to focus on the common parts, ignoring tasks, so as to achieve the purpose of learning to learn. The specific settings in the training process are as follows: memory size is 2048, batch size is 16, episode width is 5, episode length is 30; The length of the output key is 64, and the validation set is used for verification every 20 times.

Mini-ImageNet training The Mini-ImageNet dataset contains 100 classes randomly chosen from the ImageNet dataset. We randomly split the dataset into a training set, a validation set, and a test set containing 64, 16, and 20 classes, respectively. We take the pre-trained model in ref⁵⁰ and fine-tuned it using cosine distance as the meta-training metric. Once the meta-training process is done, the weights for the controller won't be updated. We use the ResNet-18 model as the CNN controller and the output feature vector of the CNN is 512-dimensional.

Data availability

The data supporting plots within this paper 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 at https://github.com/Jaylenne/TLSH_MANN.

References

1. Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D. & Lillicrap, T. Meta-learning with memory-augmented neural networks. In Balcan, M. F. & Weinberger, K. Q. (eds.) *Proceedings of The 33rd International Conference on Machine Learning*, vol. 48 of *Proceedings of Machine Learning Research*, 1842–1850 (PMLR, New York, New York, USA, 2016). URL <http://proceedings.mlr.press/v48/santoro16.html>.
2. Stevens, J. R., Ranjan, A., Das, D., Kaul, B. & Raghunathan, A. Manna: An accelerator for memory-augmented neural networks. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 794–806 (2019).
3. Rae, J. W. *et al.* Scaling memory-augmented neural networks with sparse reads and writes (2016). 1610.09027.
4. Von Neumann, J. First draft of a report on the edvac. *IEEE Annals of the History of Computing* **15**, 27–75 (1993).
5. Strubell, E., Ganesh, A. & McCallum, A. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 3645–3650 (Association for Computational Linguistics, Florence, Italy, 2019). URL <https://www.aclweb.org/anthology/P19-1355>.
6. Li, D., Chen, X., Becchi, M. & Zong, Z. Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus. In *2016 IEEE international conferences on big data and cloud computing (BDCloud), social computing and networking (SocialCom), sustainable computing and communications (SustainCom)(BDCloud-SocialCom-SustainCom)*, 477–484 (IEEE, 2016).
7. Ranjan, A. *et al.* X-MANN: A Crossbar based Architecture for Memory Augmented Neural Networks 1–6 (2019).
8. Chua, L. Memristor-the missing circuit element. *IEEE Transactions on circuit theory* **18**, 507–519 (1971).
9. Xia, Q. & Yang, J. J. Memristive crossbar arrays for brain-inspired computing. *Nature materials* **18**, 309–323 (2019).
10. Li, C. *et al.* Long short-term memory networks in memristor crossbar arrays. *Nature Machine Intelligence* **1**, 49–57 (2019).

11. Wang, Z. *et al.* In situ training of feed-forward and recurrent convolutional memristor networks. *Nature Machine Intelligence* **1**, 434–442 (2019).
12. Prezioso, M. *et al.* Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* **521**, 61–64 (2015). 1412.0611.
13. Ambrogio, S. *et al.* Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature* **558**, 60–67 (2018). URL <http://dx.doi.org/10.1038/s41586-018-0180-5>.
14. Chen, W. H. *et al.* CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors. *Nature Electronics* **2**, 420–428 (2019). URL <http://dx.doi.org/10.1038/s41928-019-0288-0>.
15. Li, C. *et al.* Analogue signal and image processing with large memristor crossbars. *Nature electronics* **1**, 52–59 (2018).
16. Sheridan, P. M. *et al.* Sparse coding with memristor networks. *Nature Nanotechnology* **12**, 784–789 (2017).
17. Le Gallo, M. *et al.* Mixed-precision in-memory computing. *Nature Electronics* **1**, 246–253 (2018).
18. Zidan, M. A. *et al.* A general memristor-based partial differential equation solver. *Nature Electronics* **1**, 411–420 (2018).
19. Yang, K. *et al.* Transiently chaotic simulated annealing based on intrinsic nonlinearity of memristors for efficient solution of optimization problems. *Science Advances* **6** (2020).
20. Cai, F. *et al.* Power-efficient combinatorial optimization using intrinsic noise in memristor Hopfield neural networks. *Nature Electronics* **3**, 409–418 (2020). URL <http://dx.doi.org/10.1038/s41928-020-0436-6>.
21. Hu, M. *et al.* Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication. In *2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 1–6 (IEEE, 2016).
22. Karunaratne, G. *et al.* In-memory hyperdimensional computing. *Nature Electronics* **3** (2020). URL <http://dx.doi.org/10.1038/s41928-020-0410-3>. 1906.01548.

23. Karunaratne, G. *et al.* Robust high-dimensional memory-augmented neural networks. *Nature communications* **12**, 1–12 (2021).
24. Ni, K. *et al.* Ferroelectric ternary content-addressable memory for one-shot learning. *Nature Electronics* **2**, 521–529 (2019).
25. Laguna, A. F., Yin, X., Reis, D., Niemier, M. & Hu, X. S. Ferroelectric FET based in-memory computing for few-shot learning. *Proceedings of the ACM Great Lakes Symposium on VLSI, GLSVLSI* 373–378 (2019).
26. Li, H. *et al.* Sapiens: A 64-kb rram-based non-volatile associative memory for one-shot learning and inference at the edge. *IEEE Transactions on Electron Devices* (2021).
27. Lake, B., Salakhutdinov, R., Gross, J. & Tenenbaum, J. One shot learning of simple visual concepts. In *Proceedings of the annual meeting of the cognitive science society*, vol. 33 (2011).
28. Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K. & Wierstra, D. Matching networks for one shot learning. *arXiv preprint arXiv:1606.04080* (2016).
29. Pi, S. *et al.* Memristor crossbar arrays with 6-nm half-pitch and 2-nm critical dimension. *Nature nanotechnology* **14**, 35–39 (2019).
30. Cai, F. *et al.* A fully integrated reprogrammable memristor–cmos system for efficient multiply–accumulate operations. *Nature Electronics* **2**, 290–299 (2019).
31. Yao, P. *et al.* Fully hardware-implemented memristor convolutional neural network. *Nature* **577**, 641–646 (2020).
32. Shafiee, A. *et al.* Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News* **44**, 14–26 (2016).
33. Kaiser, Ł., Nachum, O., Roy, A. & Bengio, S. Learning to remember rare events. *arXiv preprint arXiv:1703.03129* (2017).
34. Datar, M., Immorlica, N., Indyk, P. & Mirrokni, V. S. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, 253–262 (2004).
35. Sheng, X. *et al.* Low-conductance and multilevel cmos-integrated nanoscale oxide memristors. *Advanced Electronic Materials* **5**, 1800876 (2019).

36. Li, C. *et al.* CMOS-integrated nanoscale memristive crossbars for CNN and optimization acceleration. *2020 IEEE International Memory Workshop, IMW 2020 - Proceedings* **2**, 1–4 (2020).
37. Gionis, A., Indyk, P., Motwani, R. *et al.* Similarity search in high dimensions via hashing. In *Vldb*, vol. 99, 518–529 (1999).
38. Shinde, R., Goel, A., Gupta, P. & Dutta, D. Similarity search and locality sensitive hashing using ternary content addressable memories. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 375–386 (2010).
39. Huang, Q., Feng, J., Zhang, Y., Fang, Q. & Ng, W. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proceedings of the VLDB Endowment* **9**, 1–12 (2015).
40. Kazemi, A. *et al.* A device non-ideality resilient approach for mapping neural networks to crossbar arrays. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 1–6 (IEEE, 2020).
41. Niu, G. *et al.* Geometric conductive filament confinement by nanotips for resistive switching of hfo 2-rram devices with high performance. *Scientific reports* **6**, 1–9 (2016).
42. Li, J., Montoye, R. K., Ishii, M. & Chang, L. 1 mb $0.41 \mu\text{m}^2$ 2t-2r cell nonvolatile tcam with two-bit encoding and clocked self-referenced sensing. *IEEE Journal of Solid-State Circuits* **49**, 896–907 (2013).
43. Lin, C.-C. *et al.* 7.4 a 256b-wordlength rram-based tcam with 1ns search-time and $14\times$ improvement in wordlength-energyefficiency-density product using 2.5 t1r cell. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, 136–137 (IEEE, 2016).
44. Fedorov, V. V., Abusultan, M. & Khatri, S. P. An area-efficient ternary cam design using floating gate transistors. In *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, 55–60 (IEEE, 2014).
45. Apalkov, D., Dieny, B. & Slaughter, J. Magnetoresistive random access memory. *Proceedings of the IEEE* **104**, 1796–1830 (2016).
46. Zhao, M. *et al.* Characterizing endurance degradation of incremental switching in analog rram for neuromorphic systems. In *2018 IEEE International Electron Devices Meeting (IEDM)*, 20–2 (IEEE, 2018).

47. Zhao, M. *et al.* Endurance and retention degradation of intermediate levels in filamentary analog rram. *IEEE Journal of the Electron Devices Society* **7**, 1239–1247 (2019).
48. Ambrogio, S. *et al.* Statistical fluctuations in hfo x resistive-switching memory: part i-set/reset variability. *IEEE Transactions on electron devices* **61**, 2912–2919 (2014).
49. Mao, R., Wen, B., Jiang, M., Chen, J. & Li, C. Experimentally-validated crossbar model for defect-aware training of neural networks. *IEEE Transactions on Circuits and Systems II: Express Briefs* **69**, 2468–2472 (2022).
50. Wang, Y., Chao, W.-L., Weinberger, K. Q. & van der Maaten, L. Simpleshot: Revisiting nearest-neighbor classification for few-shot learning. *arXiv preprint arXiv:1911.04623* (2019).

Acknowledgement This work was supported in part by the Early Career Scheme (Grant No. 27210321) from the Research Grant Council of Hong Kong SAR, NSFC Excellent Young Scientist Fund (HK&Macau) (Grant No. 62122005), Mainland-Hong Kong Joint Funding Scheme (MHKJFS) Project MHP/066/20, ACCESS - AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR, and Semiconductor Research Corporation (SRC) via the ASCENT center – one of six research centers in the Joint University Microelectronics Program (JUMP).

Author contribution C.L., C.G., J.P.S contributed to the conception of the idea. R.M. performed the experiments and analyzed data under the supervision of C.L.. R.M., B.W., Y.H., A.K., A.L, performed simulations. X.S. integrated the memristors. R.M., B.W., R.L., N.W., analyzed the performance compared to the state-of-the-art algorithm. R.M., Y.H, A.K., M.N., X.H., and C.L. wrote the manuscript with input from all authors.

Competing Interests The authors declare that they have no competing interests.

Correspondence Correspondence and requests for materials should be addressed to canl@hku.hk

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- [20220703MANNSt.pdf](#)