Supporting Information

Restricted Boltzmann Machines Implemented by Spin-Orbit Torque Magnetic Tunnel Junctions

Xiaohan Li[1,3,†], Caihua Wan[1,2,†*], Ran Zhang[1,†], Mingkun Zhao[1], Shilong Xiong[1], Dehao Kong[1], Xuming Luo[1], Bin He[1]. Shiqiang Liu[1], Jihao Xia[1], Guoqiang Yu[1,2], Xiufeng Han[1,2,3*]

[1]Beijing National Laboratory for Condensed Matter Physics, Institute of Physics, University of Chinese Academy of Sciences, Chinese Academy of Sciences, Beijing 100190, China

[2]Songshan Lake Materials Laboratory, Dongguan, Guangdong 523808, China

[3]Center of Materials Science and Optoelectronics Engineering, University of Chinese Academy of Sciences, Beijing 100049, China

†These authors contributed equally to this work.

Corresponding to: xfhan@iphy.ac.cn; wancaihua@iphy.ac.cn

Supplementary note S1. MNIST dataset image recognition.

The MNIST dataset comprises of 70000 hand-written images of digits ranging from 0 to 9. Out of the 70000 images, 60000 are used for training and 10,000 for testing. In this study, we demonstrated the "0" and "1" image recognition. Firstly, we reduced the original 28×28 images to 5×5 images using mean pooling. Then, we trained in an unsupervised way the RBM in Figure 2 which consisted of 25 visible nodes and 2 hidden nodes. We trained the RBM network using 100 images of "0" and "1" within an epoch and a learning rate of 0.3. After training, the hidden layer was classified into two categories. Here we used supervised logistic regression to map the two categories to the corresponding labels, using 40 test images within an epoch. Finally, we used a set of 40 test images that were completely different from the used ones to evaluate the performance of the trained RBM with the already trained weights $\mathbf{W}_1$ and $\mathbf{W}_2$. Note that we have used different images for the RBM training, logistic regression and testing to avoid overfitting. The specific algorithm used in the experiment is detailed in Figure S1-S3. We employed SOT-MTJs for all the Gibbs sampling tasks here and then retrieved sampling outcomes to a computer for matrix multiplications.

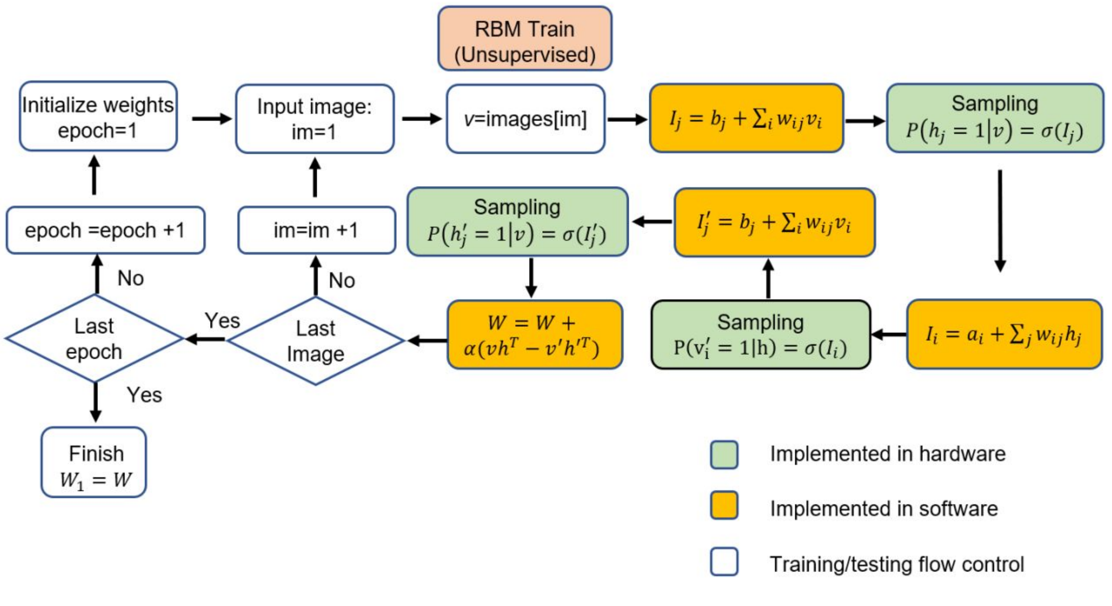

RBM Train (Unsupervised)
Initialize weights epoch=1
Input image: im=1
v=images[im]
$I_j = b_j + \sum_i w_{ij} v_i$
Sampling $P(h_j = 1|v) = \sigma(I_j)$
epoch =epoch +1
im=im +1
Sampling $P(h_j' = 1|v) = \sigma(I_j')$
$I_j' = b_j + \sum_i w_{ij} v_i$
No
No
Last epoch
Yes
Last Image
$W = W + \alpha(vh^T - v'h'^T)$
Sampling $P(v_i' = 1|h) = \sigma(I_i)$
$I_i = a_i + \sum_j w_{ij} h_j$
Yes
Finish $W_1 = W$
Implemented in hardware
Implemented in software
Training/testing flow control

Figure S1. Utilized RBM networks for unsupervised training to obtain weights ($\mathbf{W}_1$).

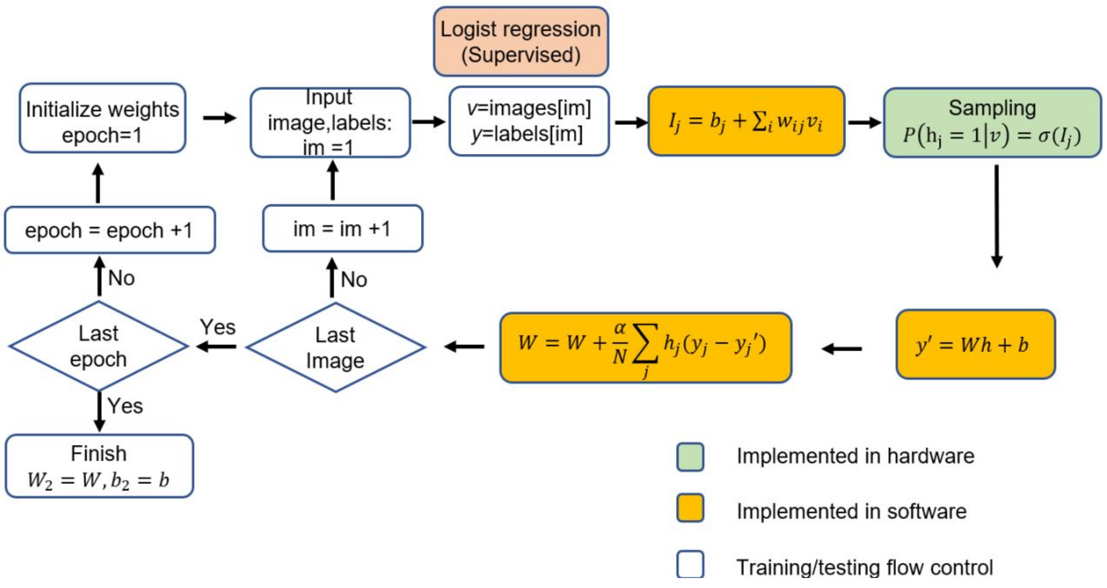

Logist regression (Supervised)
Initialize weights epoch=1
Input image,labels: im =1
v=images[im] y=labels[im]
$I_j = b_j + \sum_i w_{ij} v_i$
Sampling $P(h_j = 1|v) = \sigma(I_j)$
$y' = Wh + b$
$W = W + \frac{\alpha}{N}\sum_j h_j(y_j - y_j')$
Last Image
No
im = im +1
Yes
Last epoch
No
epoch = epoch +1
Yes
Finish $W_2 = W, b_2 = b$
Implemented in hardware
Implemented in software
Training/testing flow control

Figure S2. Using logistic regression for supervised training to obtain weights ($\mathbf{W}_2$).

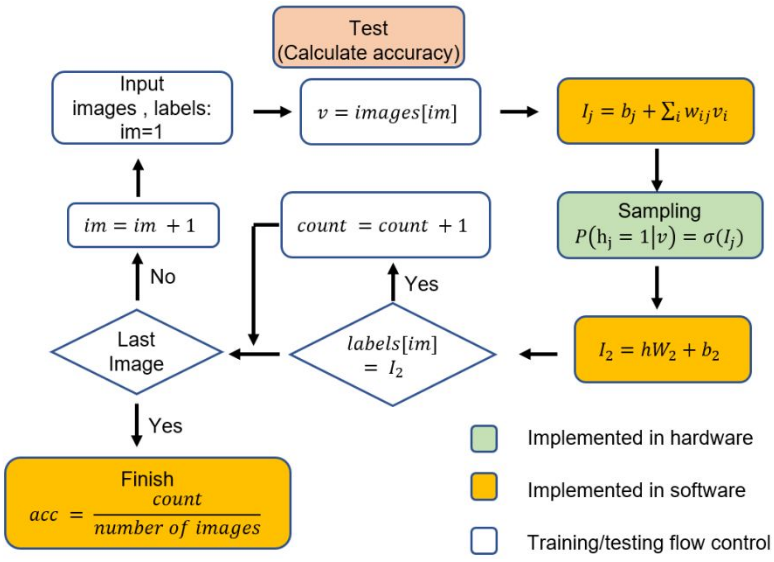

Test (Calculate accuracy)
Input images , labels: im=1
$v = images[im]$
$I_j = b_j + \sum_i w_{ij} v_i$
Sampling $P(h_j = 1|v) = \sigma(I_j)$
$I_2 = hW_2 + b_2$
$labels[im] = I_2$
Yes
$count = count + 1$
Last Image
No
$im = im + 1$
Yes
Finish $acc = \frac{count}{number\ of\ images}$
Implemented in hardware
Implemented in software
Training/testing flow control

Figure S3. Test the weights ($\mathbf{W}_1$ and $\mathbf{W}_2$) obtained from the RBM and logistic regression training to calculate the recognition accuracy.

Supplementary note S2. Free spoken speech recognition.

The Free-Spoken-Digit-Dataset comprises 3000 audio recordings of the spoken numbers from 0-9. Initially, we transformed the speech data into spectra by applying Fast Fourier Transformation (FFT) (see Figure S4). Then, we selected the main frequency range of human voices (<1 kHz) and performed mean pooling to compress the selected spectra into 4×4=16 dimensional data (similar with images). Furthermore, we used the 16 pixels as visible nodes for the speech recognition RBM. We trained the

RBM using 25 audio samples with an epoch (learning rate=0.5); 40 audio samples were used for the subsequent logistic regression. After training and regression, the weight matrix $\mathbf{W}_2$ and $\mathbf{W}_3$ were well-determined. Finally, we reserved 40 audio samples for testing. All the Gibbs sampling operations here were also conducted by SOT-MTJs.

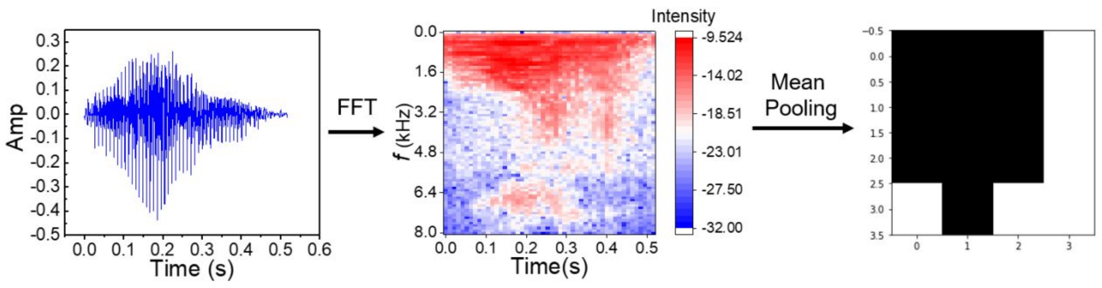


Figure S4. The free spoken digit dataset speech processing process.

Supplementary note S3. Schematic diagram of pooling.

The specific implementation process is detailed in Figure S5. Given that the MNIST dataset frequently exhibits blank peripheral regions on its four edges, we initially perform a cropping operation around the 4 peripheral regions. Subsequently, we apply a pooling operation on the resulting 20×20 region to derive the final 5×5 image.

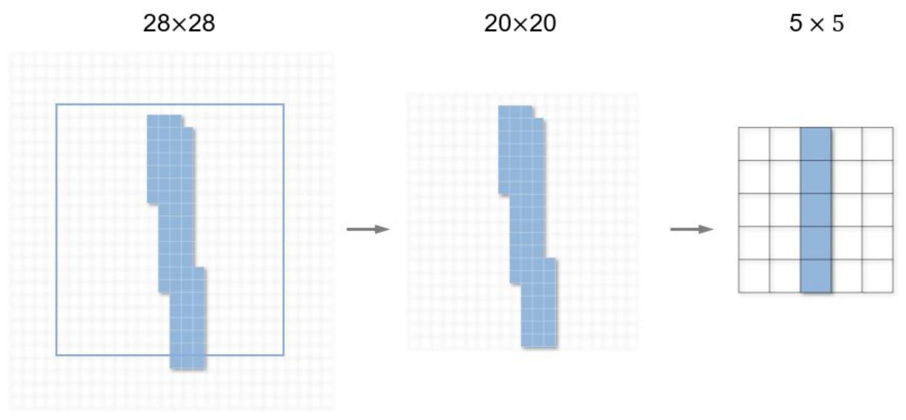


Figure S5. Schematic diagram of pooling. First, crop the 28×28 image to remove the surrounding blank area, turn it into a 20×20 image, and then pool it into a 5×5 image.

Supplementary note S4. The simulation and experiment results about recognition rate, convergence speed.

In Fig.S6**a&b**, we have incorporated the results of both simulations and experiments pertaining to speech (a) and image (b) recognition accuracy. The simulation results represent the average accuracy rate obtained after ten rounds of simulations. These data illustrate that SOT-MTJ's performance is on par with the simulations or even better possibly owing to the higher quality of true random numbers. Furthermore, in Fig.R2**c**, we present a comparative analysis of the change in the loss function over the process of iterations, encompassing both the simulation and experimental results of image recognition. Once again, the close alignment between these two sets of data, indicating a similar convergence speed, underscores the promise of SOT-MTJs for implementing neural network training. Additionally, the endurance of SOT-MTJs during this training process was approximately 4.5×10^3 iterations.

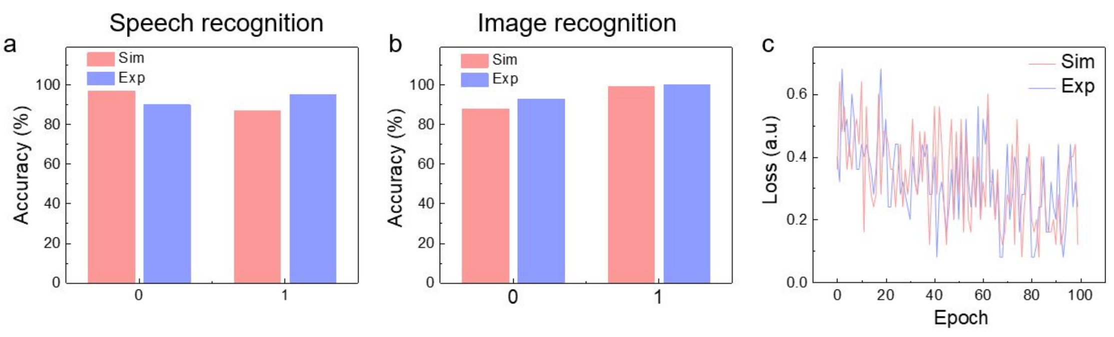


Figure S6. Simulation and experimental results about speech recognition rate (**a**) ,image recognition rate (**b**) and image recognition convergence speed (**c**).

Supplementary note S5. XOR gate inverted logic.

The RBM network to implement the inverted XOR logic is shown in Figure S7. The input (AB) and output (C) of an XOR logic gate are all locate at the visible layer. This RBM is trained to memorize (in the network weights) the 4 modes that accord with the truth table of an XOR gate. To enhance the recognition ability of the RBM, we have tripled the logic nodes as (AAABBBCCC) for the visible layer. The RBM has additional four hidden nodes for storing latent information. We stochastically constructed a training dataset of 76 “samples” where, for example, “AAA” represents “1” only if at least two in the three A are “1”. Otherwise, “AAA” represents “0”. In this case, we can introduce stochastics into the RBM, extend its phase space and enhance its robustness. We trained the RBM network with a batch of 76 samples for 500 epochs with a learning rate of 0.02. Here the network training is completed on a computer.

After training, we used SOT-MTJs as Gibbs samplers in the experiment to conduct both derivation and inference operations for the XOR RBM.

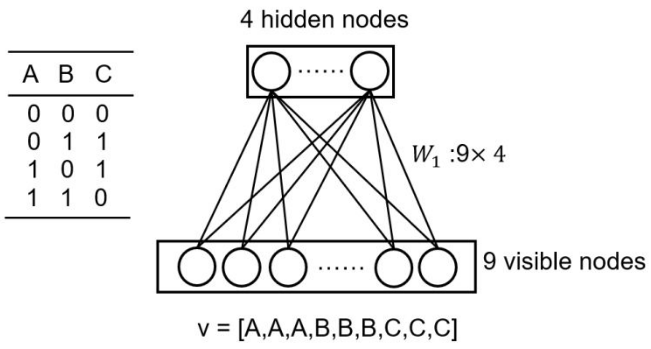


Figure S7. The RBM network for an XOR gate inverted logic.

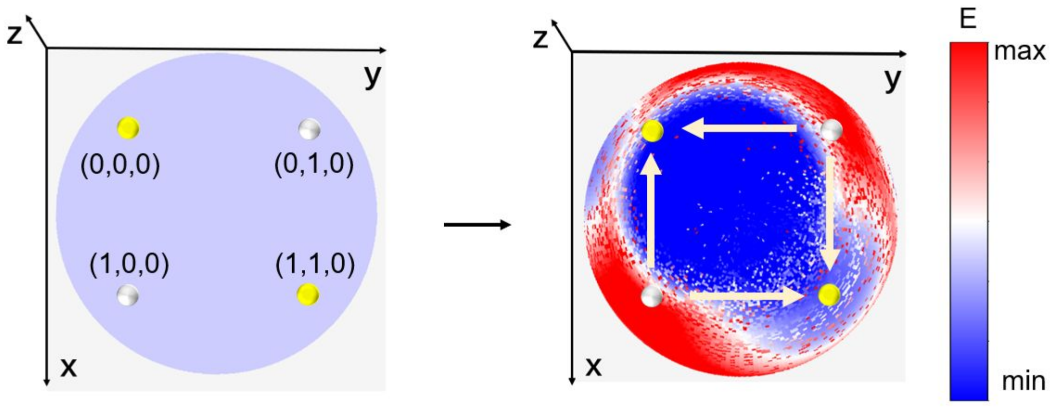


Figure S8. The phase space of the well-trained XOR gate when its output is 0.

Supplementary note S6. Integer factorization.

We define the decomposition of integer N into A and B.

$$N = AB$$

$$A = \sum_i 2^i a_i$$

$$B = \sum_i 2^i b_i$$

Construct an error function:

$$E = (AB - N)^2 + 1 \quad \text{(S1)}$$

The purpose of having a constant 1 in E is only to zero lg(E) when E is at its minimum as conveniently shown in Figure 4f.

The structure of the RBM network is depicted in Figure S9. The weights of the RBM, **W**, can be determined by comparing the generalized expression of $E = \sum_{ij} a_i W_{ij} b_j$ with the above concrete energy equation (S1). The integer factorization is achieved when E equals 1 or lg(E) equals 0. It is noteworthy that only individual a_i and b_i terms, represented by blue nodes in the figure, are independently sampled during the sampling process. Sampling of the other coupling terms, such as $a_i a_j$ or $b_i b_j$ terms, represented by the green nodes in the figure, is unnecessary as they have been already predetermined. Here the sampling process is finished by SOT-MTJs.

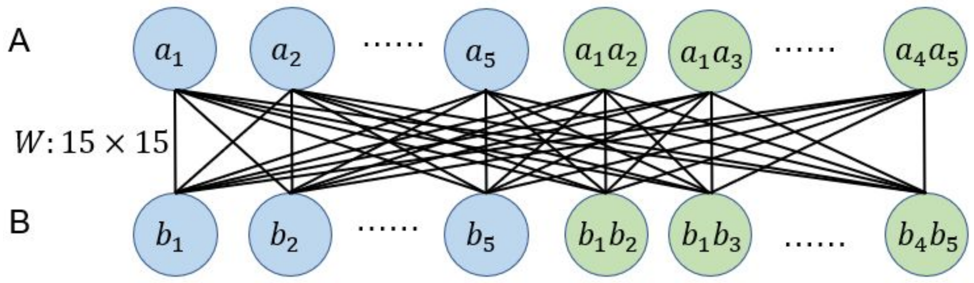


Figure S9. The RBM network that implements integer factorization.

Supplementary note S7. Image and speech recognition based on deep belief network.

We have developed a deep belief network (DBN) to recognize handwritten digits and speeches from 0 to 9 (Figure S10). The DBN comprises three RBMs and one layer of softmax. The hidden layer of the previous RBM serves as the visible layer of the subsequent RBM, and the last hidden layer is connected to the labelling layer encoded by one-hot. The function of first three RBMs is to reduce dimensionality.

For image recognition, the DBN is composed of three RBMs with sizes of 784-500, 500-200, and 200-50, trained on 60000 samples. The test set consisted of 10000 images, and we achieved an accuracy of 99.3% (Figure S11).

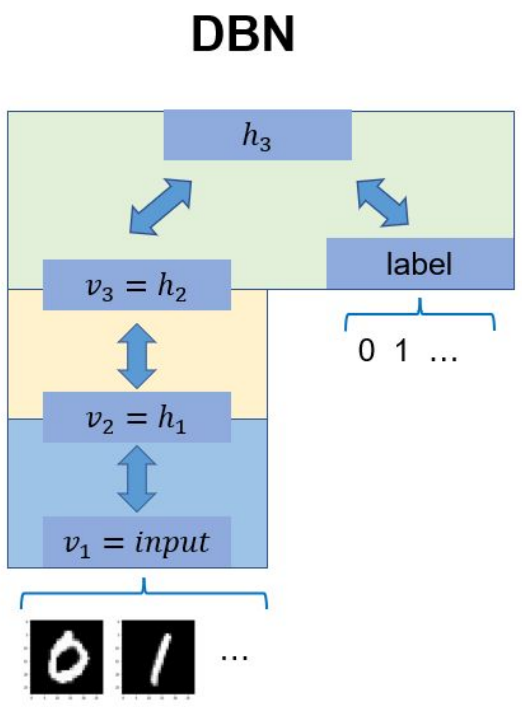
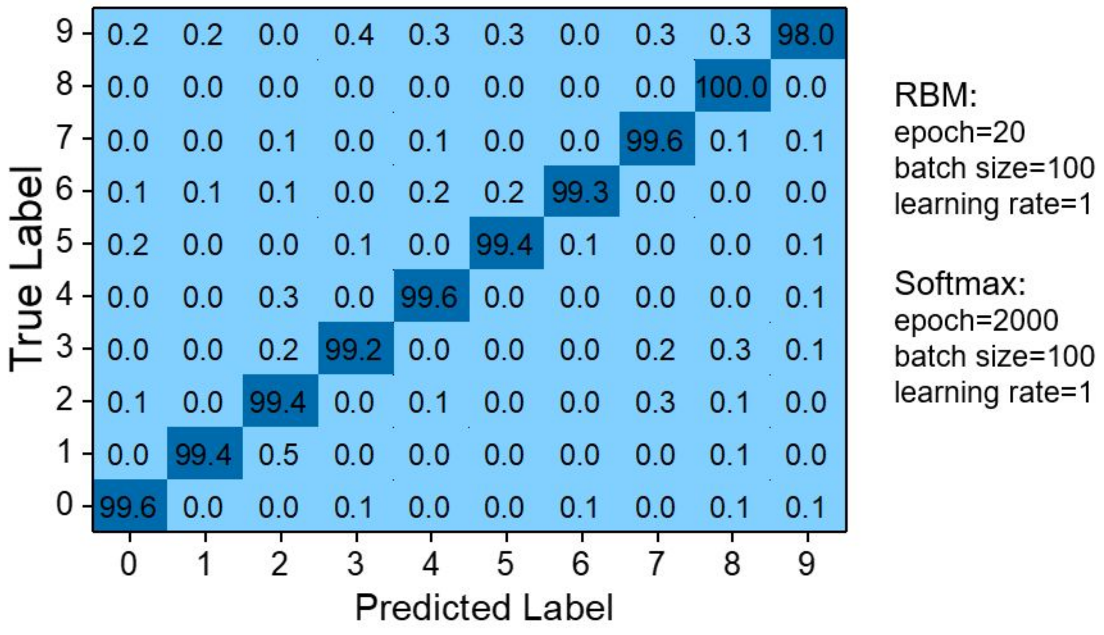


Figure S10. Structure of the DBN consisting of stacked RBMs.



True Label \ Predicted Label	0	1	2	3	4	5	6	7	8	9
9	0.2	0.2	0.0	0.4	0.3	0.3	0.0	0.3	0.3	98.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0
7	0.0	0.0	0.1	0.0	0.1	0.0	0.0	99.6	0.1	0.1
6	0.1	0.1	0.1	0.0	0.2	0.2	99.3	0.0	0.0	0.0
5	0.2	0.0	0.0	0.1	0.0	99.4	0.1	0.0	0.0	0.1
4	0.0	0.0	0.3	0.0	99.6	0.0	0.0	0.0	0.0	0.1
3	0.0	0.0	0.2	99.2	0.0	0.0	0.0	0.2	0.3	0.1
2	0.1	0.0	99.4	0.0	0.1	0.0	0.0	0.3	0.1	0.0
1	0.0	99.4	0.5	0.0	0.0	0.0	0.0	0.0	0.1	0.0
0	99.6	0.0	0.0	0.1	0.0	0.0	0.1	0.0	0.1	0.1

RBM:
epoch=20
batch size=100
learning rate=1

Softmax:
epoch=2000
batch size=100
learning rate=1

Figure S11. Percent distribution of the ten different output digits each input digit is classified into.

For speech recognition, we transformed speech data into spectra through fast Fourier transform (FFT), selecting the frequency of human voice (<1 kHz), and obtaining 10 × 10 images through mean pooling. The remaining process is the same as that for image recognition. Since we only had 3000 data points in total, we used the same data for training and testing. Ultimately, we achieved an accuracy of 83.2% (Figure S12).

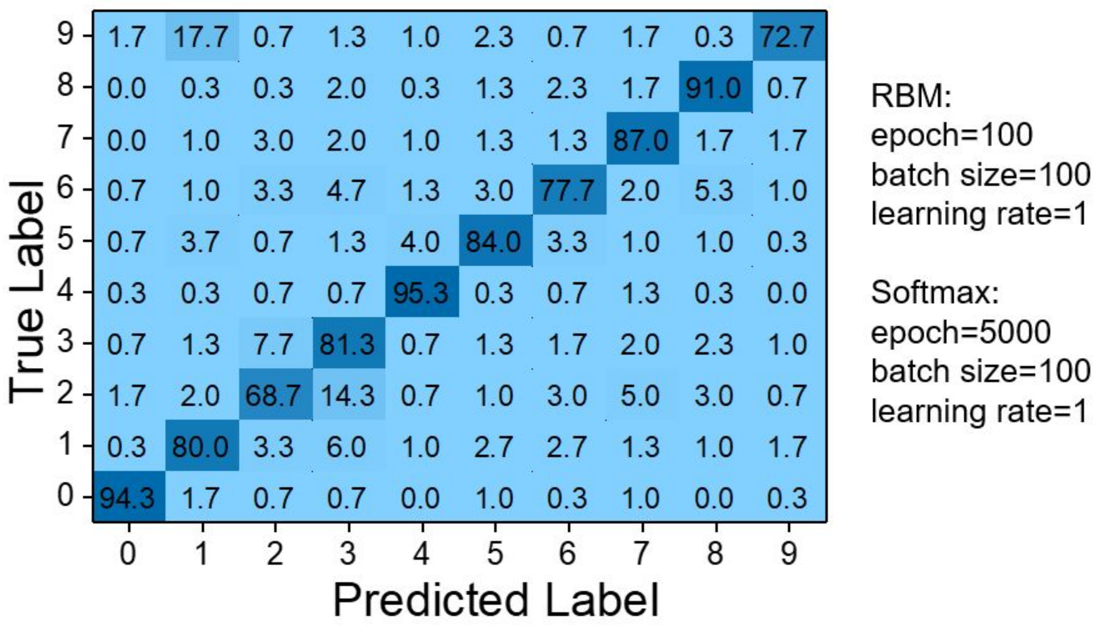


Figure S12. Percent distribution of the ten different output digits each input digit is classified into.

Supplementary note S8. Crossmodal training from images to speeches.

Besides of the crossmodal learning from auditory to visual data in experiment (Figure 3 of the main text), we also simulated crossmodal learning from visual to auditory data, demonstrating the effectiveness in crossmodal perception of our merged RBM method as illustrated in Figure S13. Here because generation of auditory data needs much more numbers of visible nodes than the recognition purpose, the number of our SOT-MTJs is far from enough, which limits us in simulation only. To this end, we utilized a similar network architecture as depicted in Figure 2d. For image training, we employed a dataset of 100 images, each with a size of 28×28 pixels (784 visible nodes) and 2 hidden nodes. The training set was trained for 1 epoch with a learning rate of 0.3. For speech recognition, we used a dataset of 13 audio samples, with 2040 visible nodes (considering only frequencies below 4 kHz) and 2 hidden nodes, trained for 100000 epochs with a learning rate of 0.3. The above two RBMs were interconnected as depicted in Figure 2d. Initially, the input image was trained to derive the corresponding labels, which were then converted into the hidden layer values in the RBM speech recognition. The visible layer was then sampled many times using the stored information in the hidden nodes to reconstruct the speech spectrum (10 times sampled, and the mean value of each pixel calculated). The reconstructed speech spectrum was then transformed back into a time-domain signal using the inverse FFT. The resulting audio files are provided as a supporting material, demonstrating clear identification between the “pronounced” "zero" and "one". Our approach exhibits a powerful solution for achieving crossmodal perception and creating associative ability for AI.

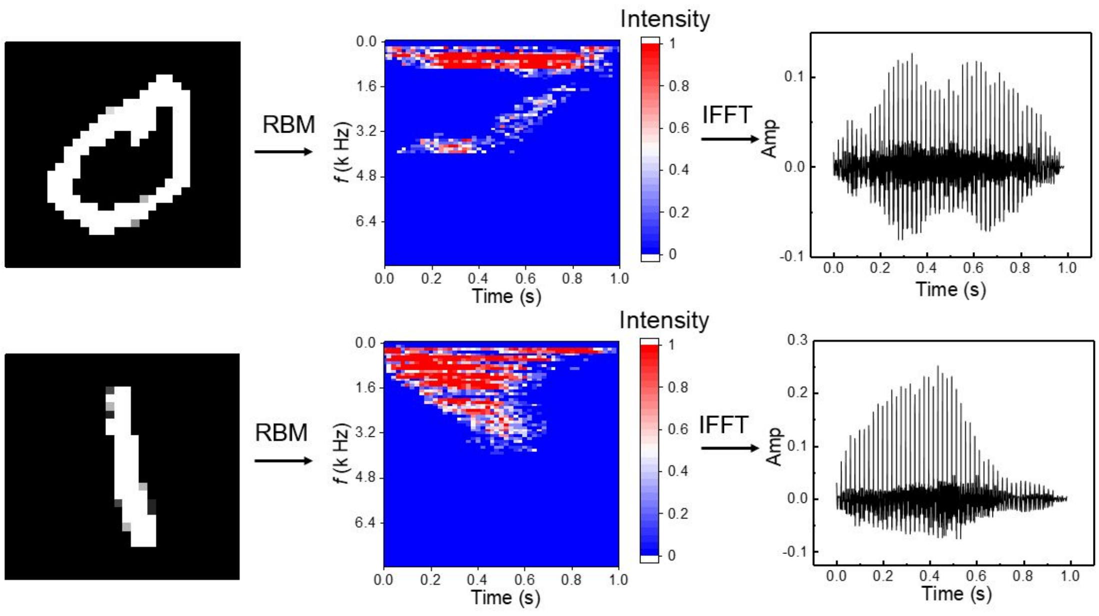


Figure S13. Multimodal training from images (left) to the auditory spectra (middle) and finally to the reconstructed speeches (right).

Supplementary note S9. The verification of the Boltzmann Distribution and forward derivation results of the XOR Gate.

We sampled the hidden layers of the trained inverted XOR logic gate network and calculated the output probability of the hidden nodes. The input for each visible was kept constant. A total of 10000 samples were collected, and the results showed that the output state indeed well satisfied the Boltzmann distribution (Figure S14), with the occurrence probability (*P*) exhibiting a negative exponential relationship with the system energy (*E*).

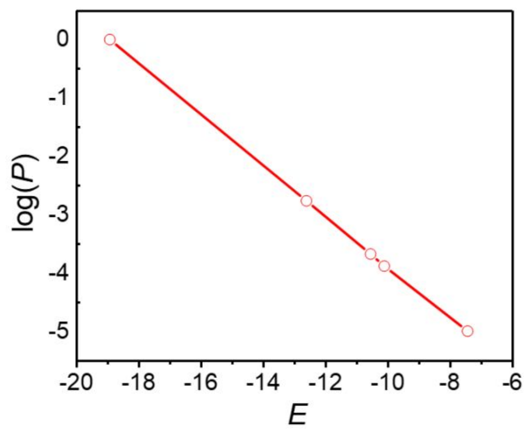


Figure S14. The dependence of output probability P on E for the hidden nodes when the same visible nodes are input.

We then performed backward inference experimentally (Figure 4d) and forward derivation both in experiment and simulation (Figure S15) using the well-trained XOR logic gate. By fixing input AB, we checked accuracy of output C similar to the approach used in Figure 4d. We conducted the forward derivation test in experiment for several times for each A ⊕ B→C combination and 10000 times in simulation. The high accuracy of the XOR gate is obtained in Figure S16.

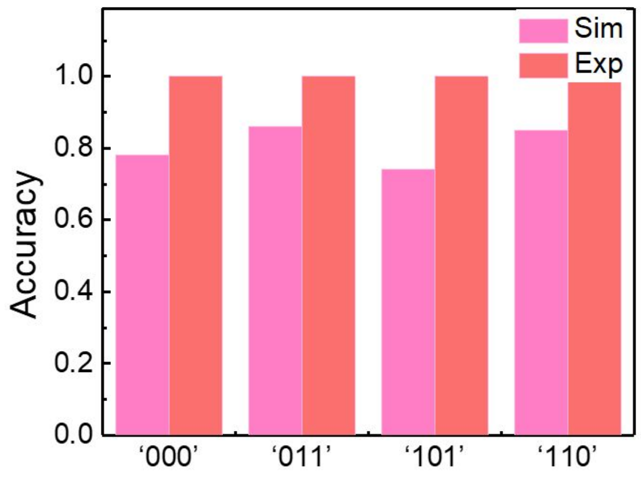


Figure S15. Accuracy of the forward derivation conducted by the trained XOR logic gate.

Supplementary note S10. The energy consumption evaluation.

Regarding the energy consumption per random number, low barrier and high barrier MTJs have different evaluation formula. $E=IV\tau$. I and V are the applied STT (SOT) current and the driving voltage for the low-barrier STT-MTJs (high barrier SOT-MTJs), respectively. $V=IR$ with R being the tunneling resistance of the STT-MTJ (channel resistance of the write line of SOT-MTJ) for the low barrier STT-MTJs (high barrier SOT-MTJs). Most importantly, τ is the writing pulse time for the SOT-MTJs while τ is at least the relaxation time τ_0 of short-time memory of STT-MTJs. To reduce self-correlation between neighboring random numbers for the low-barrier STT-MTJs, τ should be much longer than τ_0, for instance, $3\tau_0$. For the low-barrier STT-MTJs[1], τ_0 =1-100 ms, $I_{50/50}$ = 5-10 μA, R_P = 7-11 kΩ and R_{AP} = 12-19 kΩ, so the E is about 0.18-190 nJ/bit; for the high-barrier SOT-MTJs in this work, τ=50 ns, V = 1.2 V, R_{write_line} = 8 kΩ. The calculated E is thus about 18 pJ. Because of the much shorter τ for high barrier SOT-MTJs, their energy consumption is not necessarily higher than that of low-barrier STT-MTJs. Worth noting, τ_0 of low barrier STT-MTJs can be as short as 8~10 ns in the experimental work,[2] which may further reduce energy consumption to the level of 1 fJ; τ of writing pulse of SOT-MTJs can be as short as 300 ps and the energy consumption may be further reduced to the level of 0.1-1 pJ. [3]

Supplementary note S11. Details of the experiment.

1. Real measurement setups of four probe measurement system for SOT-MTJs

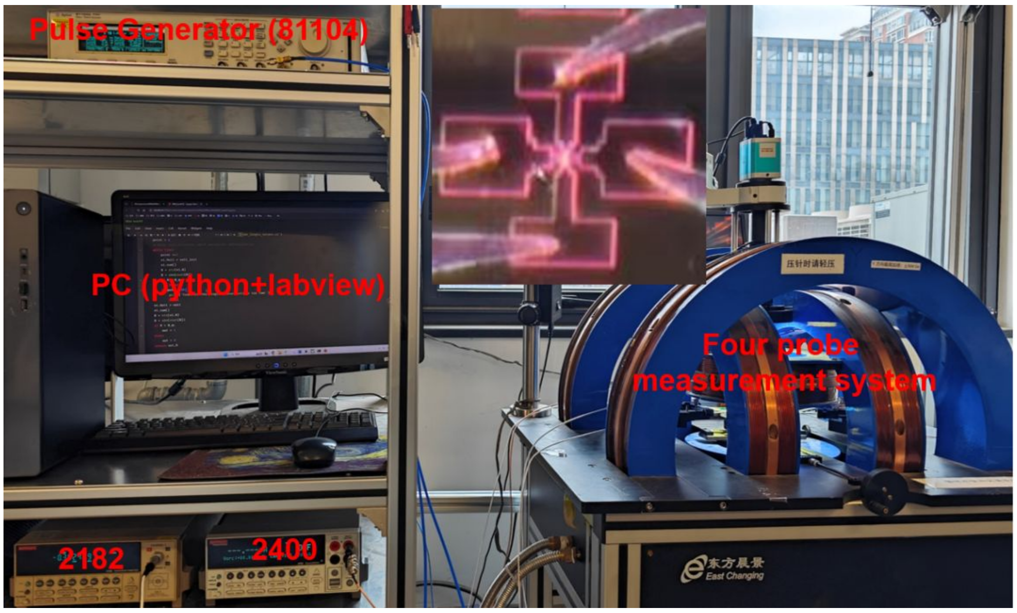


Figure S16. Real measurement setup of our four-probe measurement system.

Our probe system is provided by East Changing Technologies as shown in Fig.S16. The measurement system is sketched in Fig. 3(c) in our prior work.[4] We have employed a pulse generator (Agilent 81104A) to drive MTJ switching for sampling. We have utilized Keithley 2400 SourceMeter and Keithley 2182 Nanovoltmeter to accurately measure the resistance of an MTJ by the four-terminal method. Worth noting, in order to minimize the risk of MTJ breakdown by large pulse voltages during the sampling operations, we have also used a power splitter and a -6dB attenuator (half in magnitude) to balance the voltage vertically imposed on the tunnel barrier to nearly zero. Two bias tees have been used to isolate the pulse (Agilent) and DC (Keithley) measurement parts. We have adopted this mature method in SOT-MTJ studies.[4, 5]

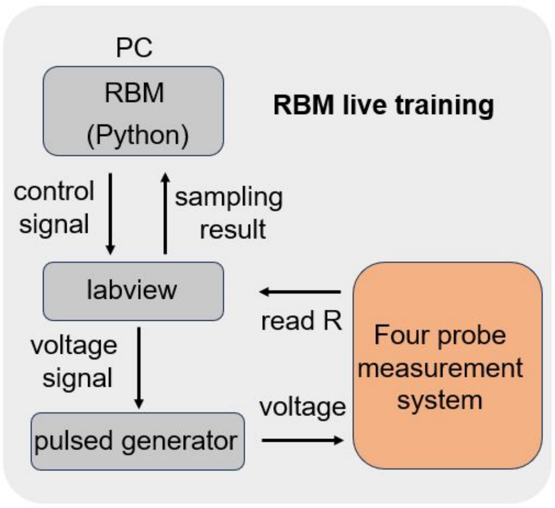


Figure S17. A schematic to show the workflow of the RBM training and test system

Because of nonvolatility of the SOT-MTJ, the sampling result is in-situ stored in the resistive state of the MTJ. Thus, we are given enough time to read the sampled state of the MTJ. Retrieved data are return to the LabVIEW program for analysis, caching and subsequent callings. The workflow is shown in Fig. S17. A PC is used to store RBM network parameters, executing the trivial matrix operations for neuron networks and control the LabVIEW program to interact with hardware. The RBM is programmed in Python. Once upon receiving commands from the PC (mainly the voltage amplitude information), the LabVIEW program activates the pulse generator to switch the MTJ device using the designated voltage amplitude for stochastic sampling. Then the MTJ test system controlled by the LabVIEW reads the resistance state of the MTJ and further returns this resistance information back to the PC. The resistance information is used to update the subsequent sampling voltage of RBM nodes according to the formula $I_j=b_j+\sum_i w_{ij}v_i$ or $I_i=a_i+\sum_j w_{ij}h_j$ by the PC. Worth stressing, here only the multiply-accumulate operation that is trivial for any neuron networks and rescaling operations on input has been computed in PC; no sigmoid function computation and pseudo-random-number is involved in PC. Once new sampling voltages are updated, a new iteration of workflow begins as before. Also worth noting, this workflow enables us to apply one nonvolatile MTJ to denote many RBM nodes in the time-division way since the PC can also be used to store resistance states of RBM nodes conveniently. Another reason why we have to adopt only one MTJ in the time-division way is that our probe system can only support one MTJ to work at the same time. Even in this case, we think, it cannot be denied that our experiment has shown the viable and matching substitution of the lengthy sampling process in software by the simple switching process of SOT-MTJ in hardware.

Fig.S17 illustrates our comprehensive testing methodology encompassing image recognition, speech recognition, and multimodal tasks. Our procedure employs Python to implement the restricted Boltzmann machine (RBM). If the whole workflow is realized in software, during the Gibbs sampling process, the sigmoid function shall be calculated utilizing the classical algorithm, which necessitates intricate calculations involving exponentials and divisions and such libraries as NumPy are utilized for generating pseudo-random numbers. These pseudo-random numbers (r) distribute equally within [0, 1]. We have to further adopt the accept-reject sampling method (accept as $r \geq P$ and otherwise reject) to translate the [0, 1] uniform distribution into the 0 or 1 binary Bernoulli style. This is the reason why this sampling process in software is lengthy and inefficient.

In our approach, the SOT-MTJ is employed to simplify this process. The operations that still need PC to execute are limited to the multiply-accumulate and rescaling the RBM's summation output. The result is then transformed to the updated voltage for sampling the MTJ. The signals from the Python program are interfaced with the LabVIEW signal via the 'autolv' software package (as attached in the "Image_and_speech_recognition_tasks.txt" and "crossmodal_task.txt" file). This interface allows for command from Python to LabVIEW and reading resistance data back from LabVIEW. The LabVIEW program is tasked to handle measurement instruments such as the pulse generator, the DC nanovoltmeter and the sourcemeter.

2. The training and testing process of RBM and the corresponding programs.

As shown in Fig.S18, we summarize our overall strategy to adopt SOT-MTJ as a sampler for RBM training and testing before stepping into details. The whole process includes two parts, the data preparation part and the neural network training process part. In the 1st part, we have to characterize the *P*-*V* relation of the SOT-MTJ according to its *P*-*V* relation and obtain its sigmoid-fitting parameters A and B. These parameters will be used in the 2nd part to rescale inputs of RBM nodes to accord them with the feature of the SOT-MTJ. Another task in the 1st part is the pretreatment of the image and speech dataset. By this pretreatment, we can reduce dimensions of the image and speech data without remarkably influencing their final recognition rates. After the 1st preparation part, we get down to the RBM training and testing.

First, we use the RBM architecture with N visible nodes and M hidden nodes. In both visible and hidden layers is there one constant node to encode bias information as usual. All the edge weights between visible and hidden layers are initialized to zero without exception (Step 3). Then we adopt the famous CD (contrastive divergence) algorithm[6] and the pretreated dataset to train the network as well as to adjust the weights. The CD algorithm contains no special mathematics but only three rounds of the Gibbs sampling processes. In the first round of the Gibbs sampling process, we feed the prepared dataset $\mathbf{v}_0$ into the visible layer (for example an image) and sample the values $\mathbf{h}_0$ of the hidden layer accordingly. In the second round, we rebuild (sample the values $\mathbf{v}_1$ of) the visible layer according to the already sampled values $\mathbf{h}_0$ of the hidden layer. In the last round, we resample the values $\mathbf{h}_1$ of the hidden layer again according to the rebuilt visible layer $\mathbf{v}_1$. The difference $\Delta_{ij}=v_{i0}h_{j0}-v_{i1}h_{j1}$ is used to update the corresponding weight w_{ij} via $w_{ij,\mathrm{new}} = w_{ij,old} + \alpha\Delta_{ij}$. The parameter α as the learning rate is chosen as 0.3 here. The training process is iterated as above until convergence or maximum iteration number is reached. Basically, the CD algorithm compares the difference between the modeled and rebuilt dataset and uses this difference to update its network parameters to minimize their difference. After convergence, the rebuilt data should be similar to the original training data, which is the principle how a RBM learns knowledges. In Step 5 (recognition and generation), we fix the image nodes in the visible layer and conduct the Gibbs sampling back and forth until convergence to obtain stochastically stable label nodes also in the visible layer for recognition. Instead, for generation, we fix the label nodes in the visible layer and conduct the Gibbs sampling back and forth to obtain stochastically stable image nodes for generation. This working principle is widely accepted for RBMs.

As shown in Fig.S18, in Step 4 (training) and Step 5 (Recognition and Generation) of the 2nd part, plenty of the Gibbs sampling operations are needed. In our experiment, the sampling operations in these two steps are all conducted by the probabilistic switching events of the SOT-MTJ devices without any involvement of the sigmoid-function calculation and the usage of any pseudo-random-numbers. This point can also be proven by our released program as shown below. Therefore, we confidently argue our work is essentially experimental.

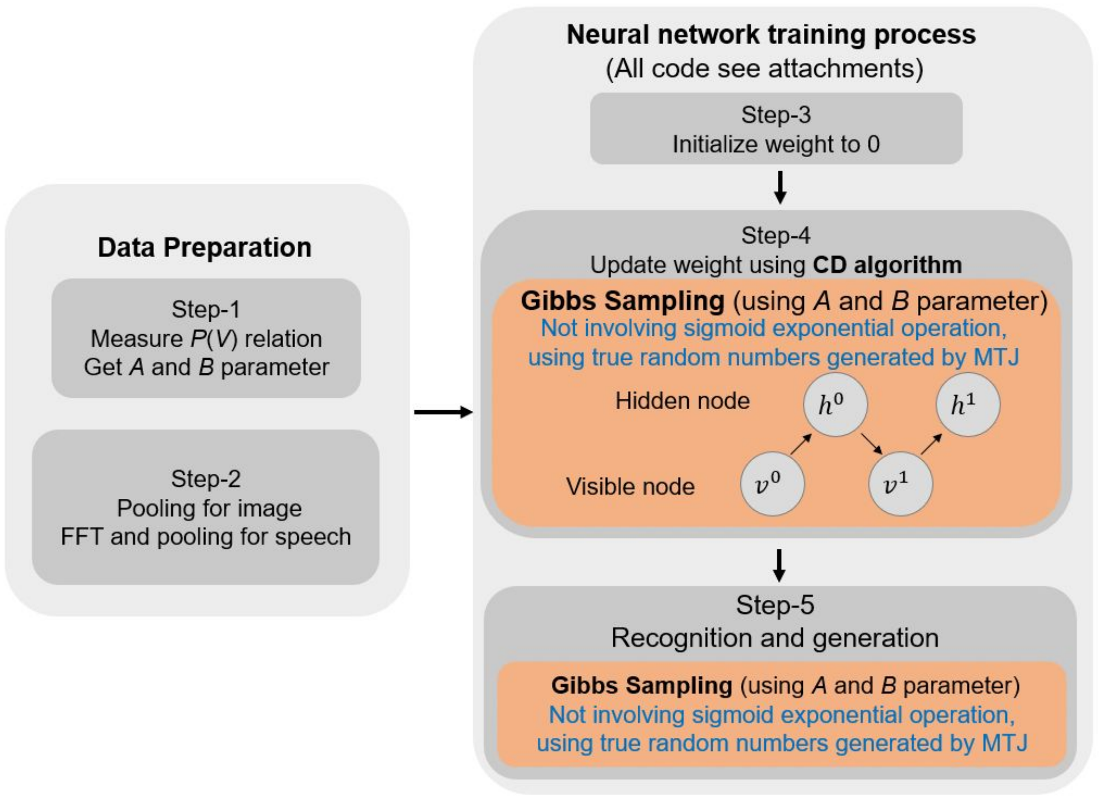

Neural network training process
(All code see attachments)
Step-3
Initialize weight to 0
Step-4
Update weight using CD algorithm
Gibbs Sampling (using A and B parameter)
Not involving sigmoid exponential operation, using true random numbers generated by MTJ
Hidden node
h^0
h^1
Visible node
v^0
v^1
Step-5
Recognition and generation
Gibbs Sampling (using A and B parameter)
Not involving sigmoid exponential operation, using true random numbers generated by MTJ
Data Preparation
Step-1
Measure P(V) relation
Get A and B parameter
Step-2
Pooling for image
FFT and pooling for speech

Figure. S18 The overall strategy to implement RBMs by SOT-MTJ. Here the SOT-MTJ acts as the Gibbs sampler for RBMs to avoid computing the complex sigmoid function and also avoid any usage of pseudo-random-numbers.

Below is a detailed introduction to the entire process using the image recognition application as an example.

2.1 Pooling images.

The specific implementation process is detailed in Fig.S5. Given that pictures in the MNIST dataset frequently exhibit blank peripheral regions on their four edges, we initially perform a cropping operation around their 4 peripheral regions. Subsequently, we apply a pooling operation on the resulting 20×20 region to derive the final 5×5 image. The average pooling is used in our experiment. First, we calculate the average value of an area (4 grids) as the average intensity value of the new grid, and then perform a binarization operation on it. In this case, the dimension of an image is reduced from 400 to 25 and much less nodes are needed in the visible layer of the RBM for the image recognition.

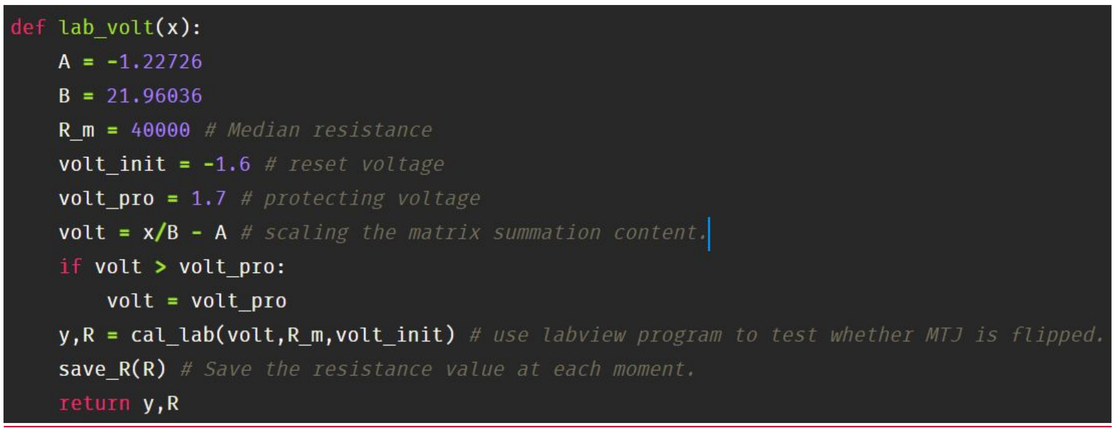
2.2 RBM networks

The program to train the image recognition RBM is shown below. The three green parts correspond to the three rounds of the Gibbs sampling operations in hardware as mentioned above. The detailed program is attached in an independent file.
We trained in an unsupervised way the RBM in Figure 2 which consisted of 25 visible nodes and 2 hidden nodes. We trained the RBM network using 100 images of "0" and "1" within an epoch and a learning rate α of 0.3. After training, the hidden layer was classified into two categories. Here we used supervised logistic regression[7] to map the two categories to the corresponding labels, using 40 test images within an epoch. In logistic regression, the dependent variable is categorical and represents the outcome of interest, typically encoded as 0 or 1. The model works by fitting a logistic curve to the data, which maps the input variables to the probability of the event occurring. This curve is defined by a linear combination of the input variables, each weighted by a coefficient. The coefficients are estimated using maximum likelihood estimation, and the model makes predictions by applying a threshold to the predicted probabilities. Finally, we used a set of 40 test images that were completely different from the used ones to evaluate the performance of the trained RBM with the already trained weights $\mathbf{W}_1$ and $\mathbf{W}_2$. Note that we have used different images for the RBM training, logistic regression and testing to avoid overfitting. The specific algorithm used in the

experiment is detailed in Figure S1-S3. We employed SOT-MTJs to experimentally conduct all the Gibbs sampling tasks here and then retrieved sampling outcomes to a computer for other trivial operations for neuron networks such as matrix multiplications.

2.3 Detailed core source code

Below displayed are some core codes:



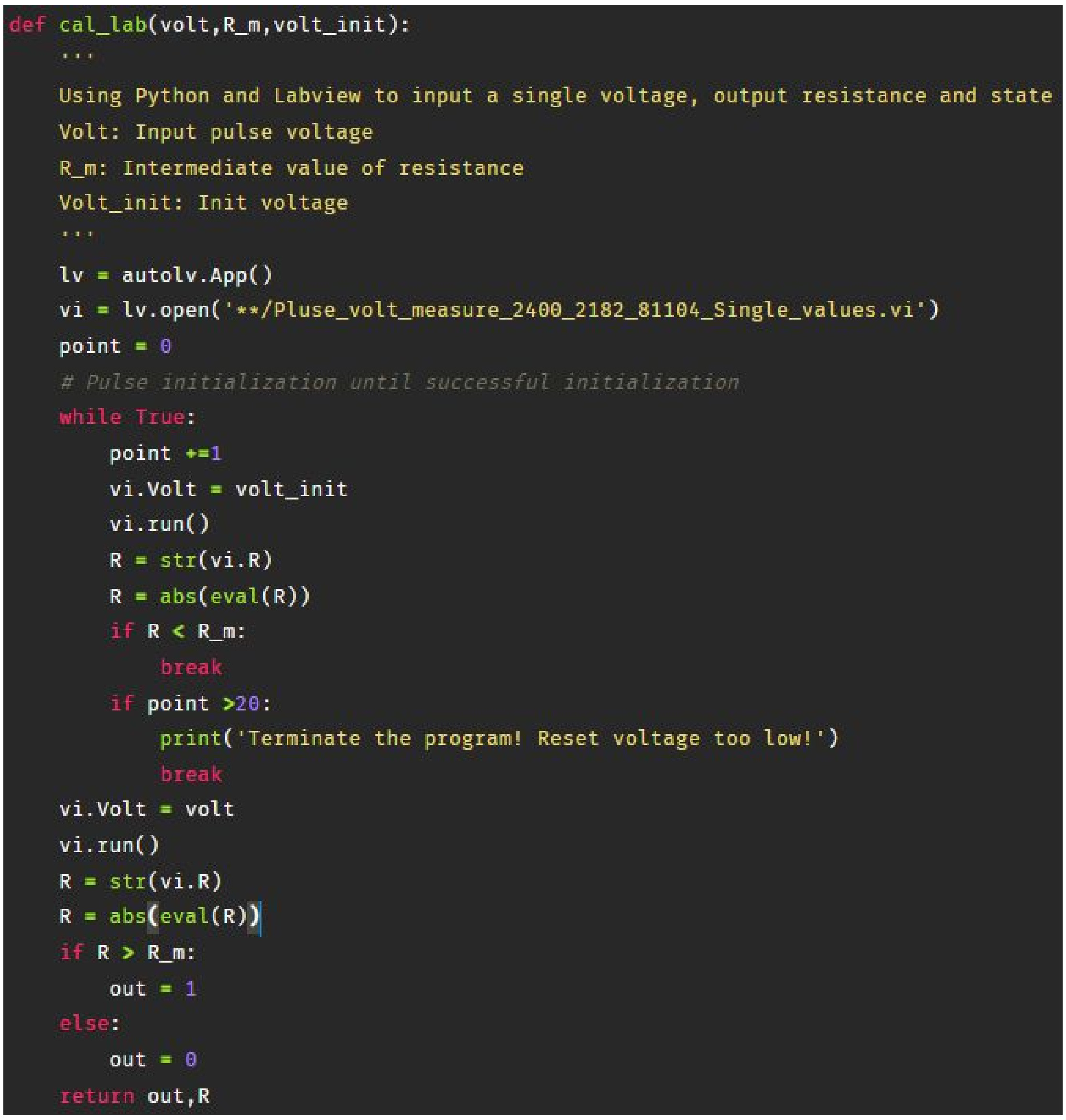
```python
def lab_volt(x):
    A = -1.22726
    B = 21.96036
    R_m = 40000 # Median resistance
    volt_init = -1.6 # reset voltage
    volt_pro = 1.7 # protecting voltage
    volt = x/B - A # scaling the matrix summation content.
    if volt > volt_pro:
        volt = volt_pro
    y,R = cal_lab(volt,R_m,volt_init) # use labview program to test whether MTJ is flipped.
    save_R(R) # Save the resistance value at each moment.
    return y,R
```

Figure S19. This program to calculate the pulse voltage DOES NOT use the sigmoid function. The only budget is division and addition as shown by "volt = x/B -A". Finally, the voltage value is delivered to the LabVIEW program as shown by the sentence "y, R= cal_lab(volt, R_m, volt_init)", and the resistance value of the SOT-MTJ after sampling is returned and recorded.

The "lab_volt" function in Fig.S19 initiates its process by employing parameters from the sigmoid function which are fitted to the calibrated *P-V* curve from the MTJ. It then calculates a voltage value by summing elements within a matrix and applying a rescaling operation from x to volt. This voltage is subsequently sent to the pulse generator, Agilent 81104A, through the "cal_lab" function.

After sourcing a pulse voltage to the writing line of the SOT-MTJ by Agilent 81104A, the resistance of the MTJ after sampling is read by Keithley 2400 SourceMeter and Keithley 2182 Nanovoltmeter. The obtained resistance is then conveyed to the variable 'R'. The resistance value 'R' is used to record the MTJ state-high or low allowing the conversion into a binary signal 1 or 0.

Especially, the particular line of code, “volt = x/B - A”, reveals that our voltage values are not derived by computing the sigmoid function. Instead, they are directly rescaled on the basis of the matrix summation result 'x' considering the characteristics (*A* and *B*) of a real SOT-MTJ device. This approach represents the replacement of the sampling process in software by a real SOT-MTJ switching process in hardware, effectively reducing the need for some complex computations and avoid the calling and usage of any pseudo-random-numbers. The following is the code of the “cal_lab” section.



```python
def cal_lab(volt,R_m,volt_init):
    '''
    Using Python and Labview to input a single voltage, output resistance and state
    Volt: Input pulse voltage
    R_m: Intermediate value of resistance
    Volt_init: Init voltage
    '''
    lv = autolv.App()
    vi = lv.open('**/Pluse_volt_measure_2400_2182_81104_Single_values.vi')
    point = 0
    # Pulse initialization until successful initialization
    while True:
        point +=1
        vi.Volt = volt_init
        vi.run()
        R = str(vi.R)
        R = abs(eval(R))
        if R < R_m:
            break
        if point >20:
            print('Terminate the program! Reset voltage too low!')
            break
    vi.Volt = volt
    vi.run()
    R = str(vi.R)
    R = abs(eval(R))
    if R > R_m:
        out = 1
    else:
        out = 0
    return out,R
```

Figure S20. Using autolv to implement programs that call LabVIEW. Firstly, use the reset pulse to reset the MTJ to its initial R_P state. Then give a pulse of 50 ns with a certain voltage amplitude to the SOT-MTJ device and check whether the MTJ has been

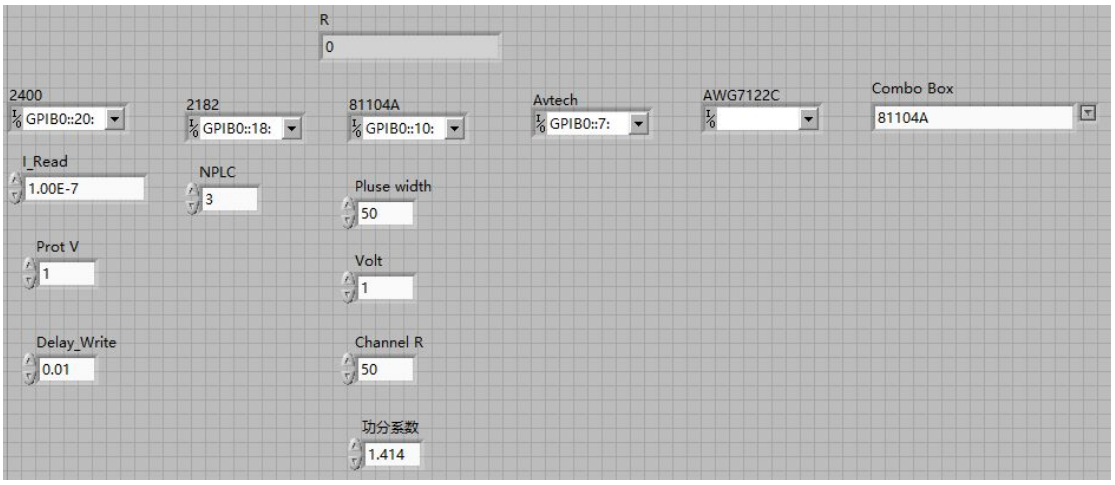
switched to its R_{AP} state or still remained on the R_P state. If SOT-MTJ switching is achieved, output 1; otherwise, output 0.

The primary objective of the function “cal_lab” in Fig.S20 is to configure the LabVIEW program using Python's 'autolv' software package, which includes setting voltage values and other parameters. In a reset-failure event, the system is programmed to persistently attempt resetting until the reset is successful. Once a successful reset is achieved, the system will use the input voltage to drive switching of the SOT-MTJ and compare the resistance result of the MTJ after sampling with the median resistance value and output a binary signal accordingly.

Within the code, the 'vi.run()' function plays a critical role. It transfers the voltage value from the Python script to the LabVIEW and executes the LabVIEW program. The outcome is a sampled binary random number according to the desired probability distribution. This process marks the second part of hardware acceleration, bypassing the need for Python-generated or other software-based pseudo-random numbers. Instead, it directly utilizes a single voltage pulse imposed on the write channel of the MTJ to produce genuine binary random numbers with proper probability distribution for the Gibbs sampling.

Pluse_volt_measure_2400_2182_81104_Single_values.vi

The program is shown in the following figure：

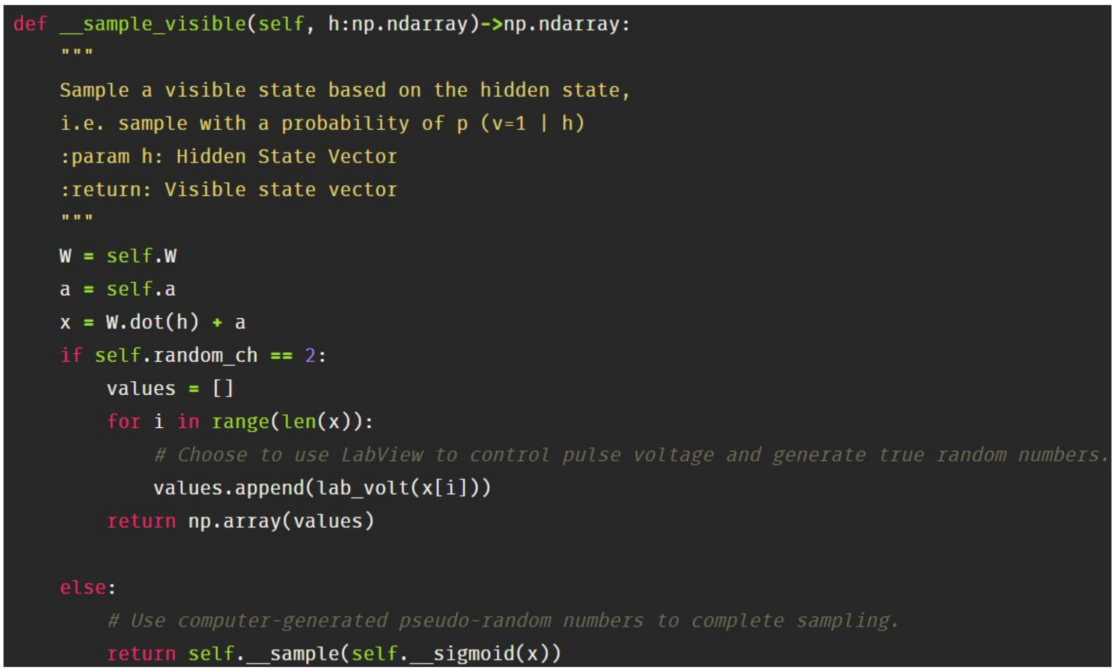


R: 0
2400: GPIB0::20:
I_Read: 1.00E-7
Prot V: 1
Delay_Write: 0.01
2182: GPIB0::18:
NPLC: 3
81104A: GPIB0::10:
Pluse width: 50
Volt: 1
Channel R: 50
功分系数: 1.414
Avtech: GPIB0::7:
AWG7122C:
Combo Box: 81104A

Figure S21. LabVIEW core program. The autolv software package developed in

Python enables the control of the LabVIEW core program. This software enables Agilent 81104A to output the voltage pulse with the amplitude obtained from the Python processing program to drive the probabilistic switching of the SOT-MTJ. Subsequently, after physical sampling, the resistance of the MTJ is read by Keithley 2400 and 2182, and the result is sent back to the Python processing program to complete the Gibbs sampling. Worth mentioning, in the above operation, we do not use any sigmoid function and do not use any pseudo-random numbers as well.

The following code is the Gibbs sampling part of the RBM algorithm:



```
def __sample_visible(self, h:np.ndarray)->np.ndarray:
    """
    Sample a visible state based on the hidden state,
    i.e. sample with a probability of p (v=1 | h)
    :param h: Hidden State Vector
    :return: Visible state vector
    """
    W = self.W
    a = self.a
    x = W.dot(h) + a
    if self.random_ch == 2:
        values = []
        for i in range(len(x)):
            # Choose to use LabView to control pulse voltage and generate true random numbers.
            values.append(lab_volt(x[i]))
        return np.array(values)

    else:
        # Use computer-generated pseudo-random numbers to complete sampling.
        return self.__sample(self.__sigmoid(x))
```

Figure S22. The program for sampling visible layer nodes. When “random_ch == 2”, we use true random numbers generated by SOT-MTJ for the Gibbs sampling, and when “random_ch == 1”, we use computer-generated pseudo-random-numbers. This set of program enables us to implement the Gibbs sampling for RBM in hardware or software.

The key component of the system is the ability to generate true random numbers. If the value of "random_ch" is set to 2, the LabVIEW operation utilizing SOT-MTJ as the sampler will be selected to generate true random numbers. Alternatively, if it is not 2, the NumPy package in Python will be activated and be used to generate pseudo-random numbers for simulation purposes. The former option provides real-time sampling, while the latter is used for simulation only.

All programs required to reproduce data in this study have been included and annotated in details in an independent file (Image_and_speech_recognition_tasks.txt and crossmodal_task.txt). We sincerely welcome other teams to replicate our results. As we have repeated the results many times, we think it can be reproduced by others following the same or similar procedure.

In conclusion, we did not only use simulation to realize the image recognition, speech recognition, or multimodal tasks. Instead, we have conducted real-time testing using the real SOT-MTJ device for the Gibbs sampling operation. Our experimental results demonstrate viability, matchiness and stability of SOT-MTJs in hardware acceleration for RBM neural networks. We are confident these results also provide invaluable support for future development of hardware accelerators based on SOT-MTJ devices.

Reference

(1) Borders, W. A.; Pervaiz, A. Z.; Fukami, S.; Camsari, K. Y.; Ohno, H.; Datta, S. Integer factorization using stochastic magnetic tunnel junctions. *Nature* **2019**, *573* (7774), 390-393. DOI: 10.1038/s41586-019-1557-9.
(2) Hayakawa, K.; Kanai, S.; Funatsu, T.; Igarashi, J.; Jinnai, B.; Borders, W. A.; Ohno, H.; Fukami, S. Nanosecond Random Telegraph Noise in In-Plane Magnetic Tunnel Junctions. *Phys Rev Lett* **2021**, *126* (11), 117202. DOI: 10.1103/PhysRevLett.126.117202.
(3) Cai, K.; Talmelli, G.; Fan, K.; Van Beek, S.; Kateel, V.; Gupta, M.; Monteiro, M. G.; Chroud, M. B.; Jayakumar, G.; Trovato, A.; et al. First demonstration of field-free perpendicular SOT-MRAM for ultrafast and high-density embedded memories. In 2022 International Electron Devices Meeting (IEDM), 2022.
(4) Zhao, M. K.; Zhang, R.; Wan, C. H.; Luo, X. M.; Zhang, Y.; He, W. Q.; Wang, Y. Z.; Yang, W. L.; Yu, G. Q.; Han, X. F. Type-Y magnetic tunnel junctions with CoFeB doped tungsten as spin current source. *Applied Physics Letters* **2022**, *120* (18), 182405. DOI: 10.1063/5.0086860.
(5) Li, X. H.; Zhao, M. K.; Zhang, R.; Wan, C. H.; Wang, Y. Z.; Luo, X. M.; Liu, S. Q.; Xia, J. H.; Yu, G. Q.; Han, X. F. True random number generator based on spin–orbit torque magnetic tunnel junctions. *Applied Physics Letters* **2023**, *123* (14), 142403. DOI: 10.1063/5.0171768.
(6) Hinton, G. E. Training products of experts by minimizing contrastive divergence. *Neural computation* **2002**, *14* (8), 1771-1800.
(7) LaValley, M. P. Logistic regression. *Circulation* **2008**, *117* (18), 2395-2399. DOI: 10.1161/CIRCULATIONAHA.106.682658.