nature machine intelligence

Article

https://doi.org/10.1038/s42256-023-00680-y

Uncertainty quantification via a memristor Bayesian deep neural network for risk-sensitive reinforcement learning

In the format provided by the authors and unedited

This Supplementary Information contains:

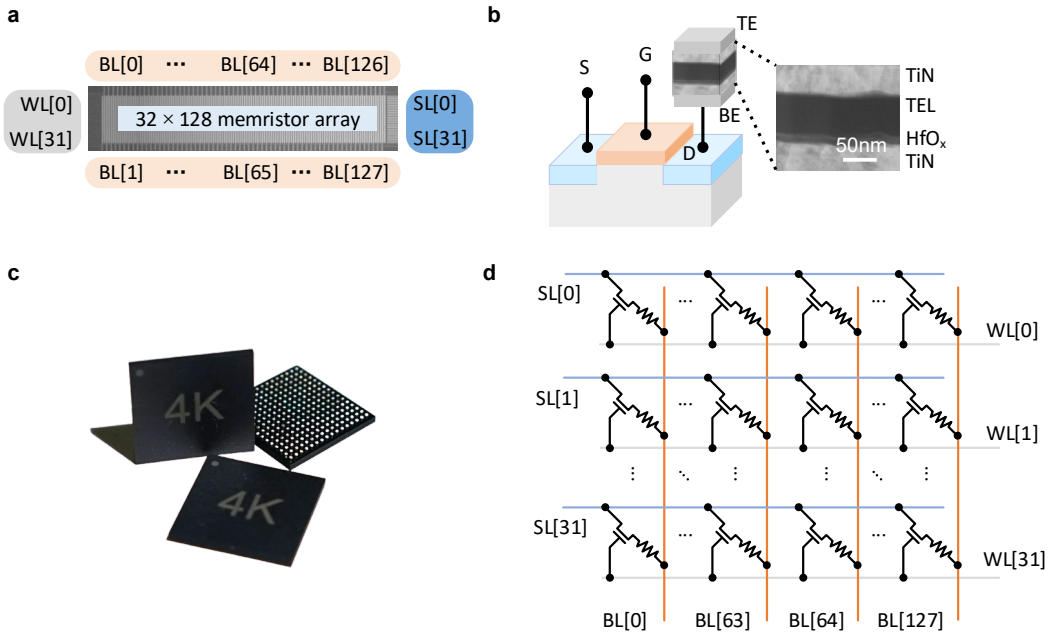
Supplementary figures 1 – 16.

Supplementary tables 1 – 8.

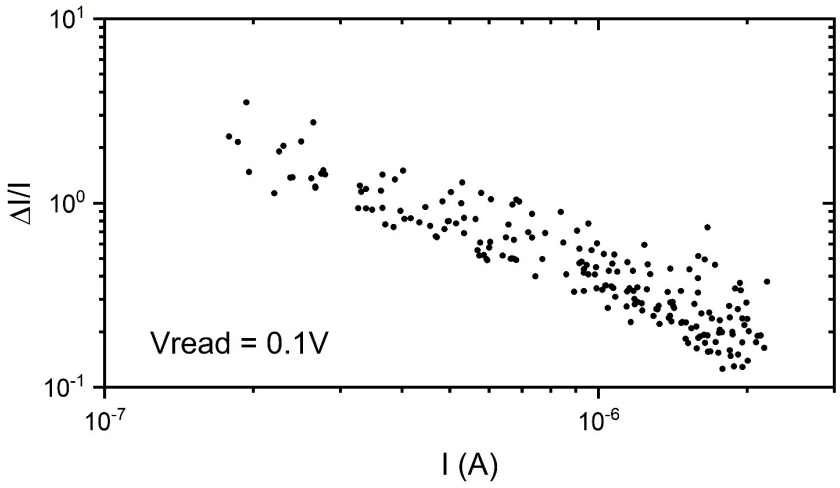
Supplementary notes 1 – 5.

Captions for supplementary video 1.

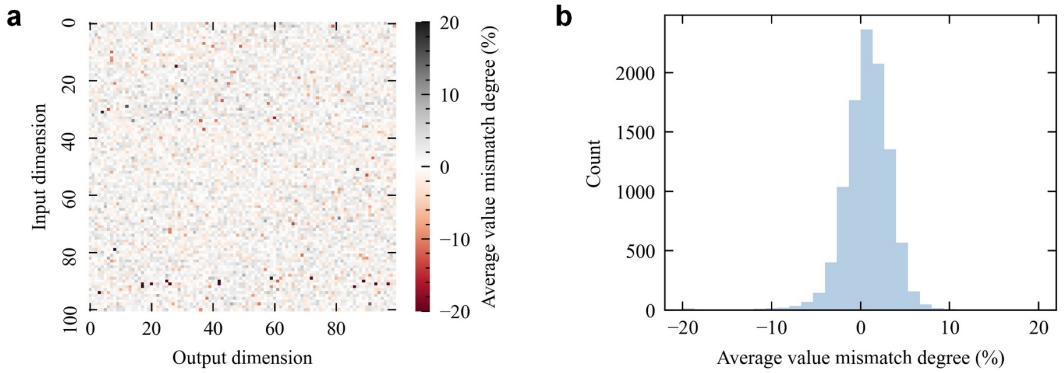
Supplementary Figures



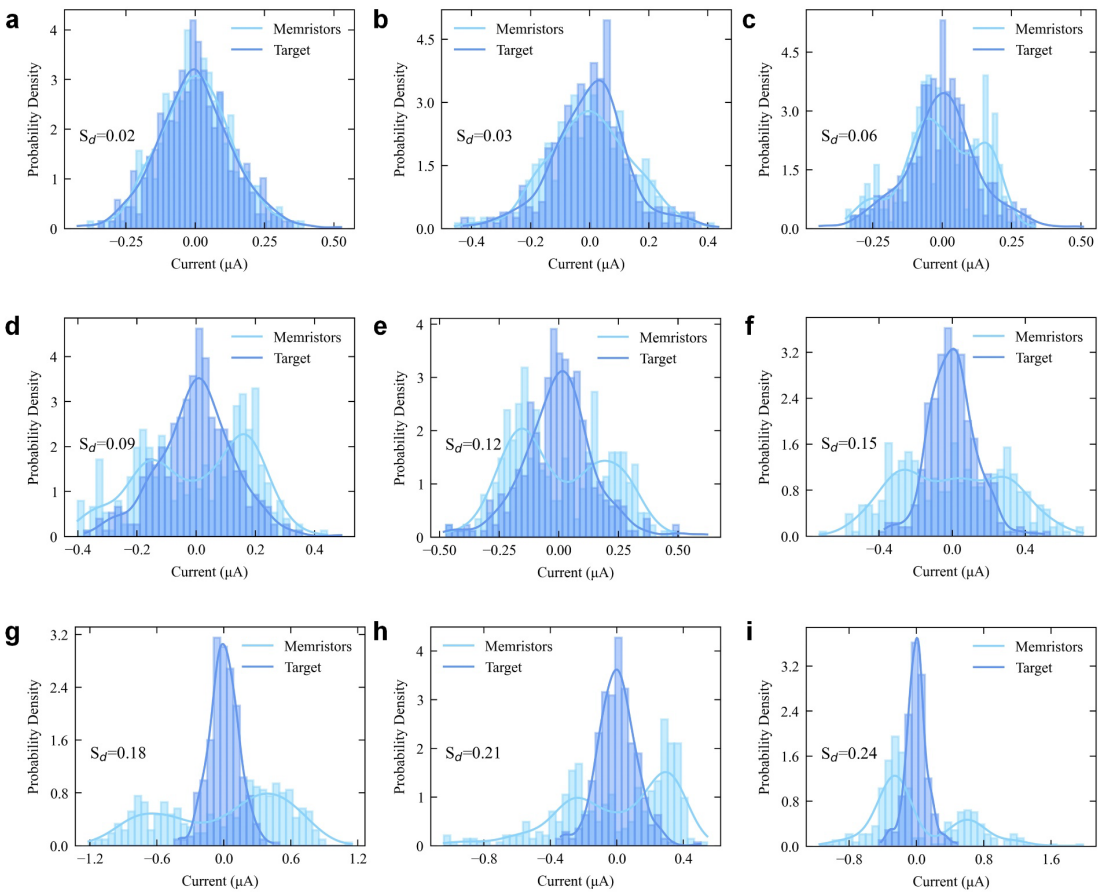
Supplementary Figure 1 | 32×128 1T1R memristor array chip and on-chip circuit. **a**, Layout of the 32×128 (4K) memristor crossbar array. **b**, 1T1R structure and TiN/TEL/HfO_x/TiN material stack of the array. **c**, Photograph of the 4K memristor array chips. **d**, On-chip circuit. To fully utilize the parallelism of the array, the 32 source lines (SLs) occur perpendicular to the 128 bit lines (BLs) and parallel to 32 word lines (WLs).



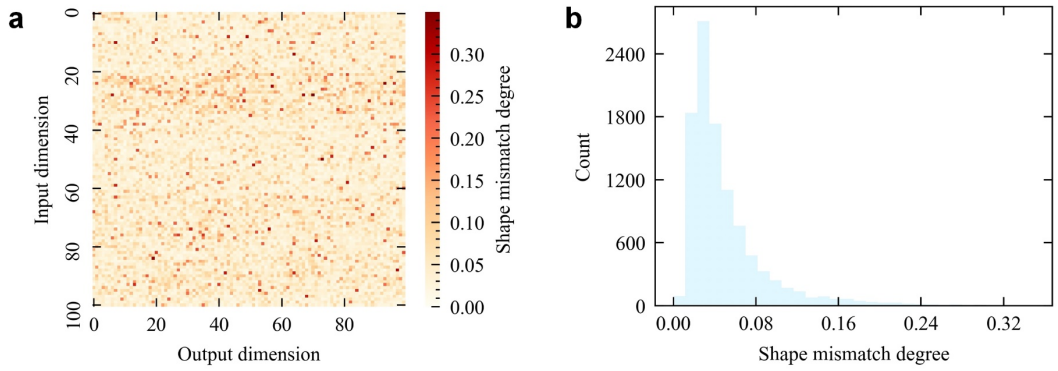
Supplementary Figure 2 | Relationship between $\frac{\Delta I}{I}$ and I. We measured the abrupt amplitude of RTN as ΔI and calculated the values of $\Delta I/I$. We selected 325 devices in the current window ranges from 2×10^{-7} to 2×10^{-6} A at V_{read} = 0.1 V. Each dot indicates an individual memristor device.



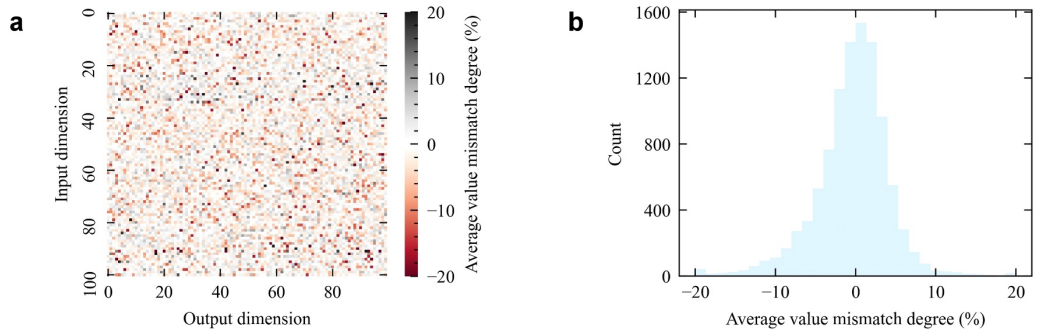
Supplementary Figure 3 | Distribution of average value mismatch degree (A_d). a, Measured distribution of 100×100 weight matrix's A_d. **b,** Histogram plot of A_d.



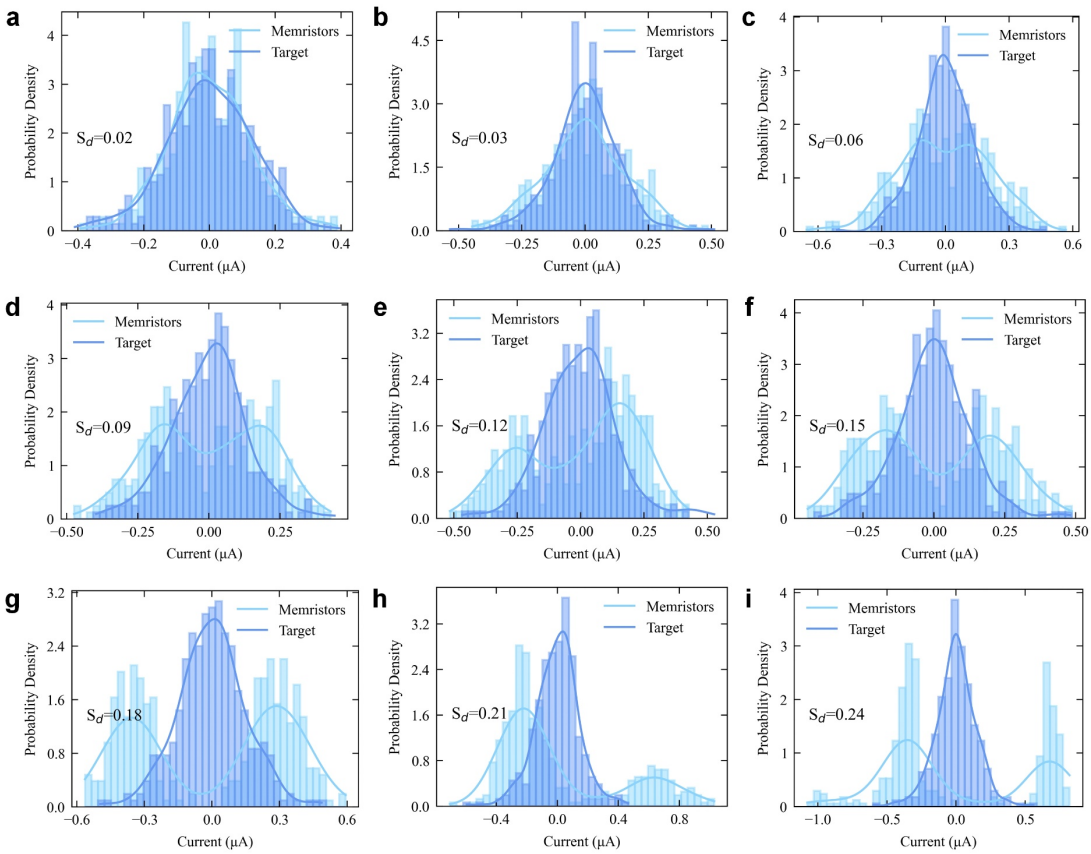
Supplementary Figure 4 | Typical single-weight distribution shapes with different mismatch degree (S_d) in the experiment. a-i, S_d is 0.02, 0.03, 0.06, 0.09, 0.12, 0.15, 0.18, 0.21, and 0.24, respectively. Each distribution shape is obtained from 360-times read results.



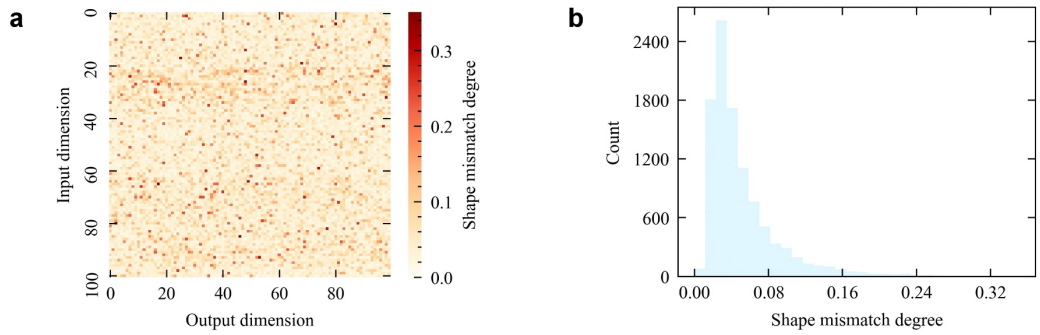
Supplementary Figure 5 | Shape mismatch degree (S_d) distribution. a, Distribution of 100 ×100 weight matrix's S_d in the experiment. **b,** Histogram plot of S_d.



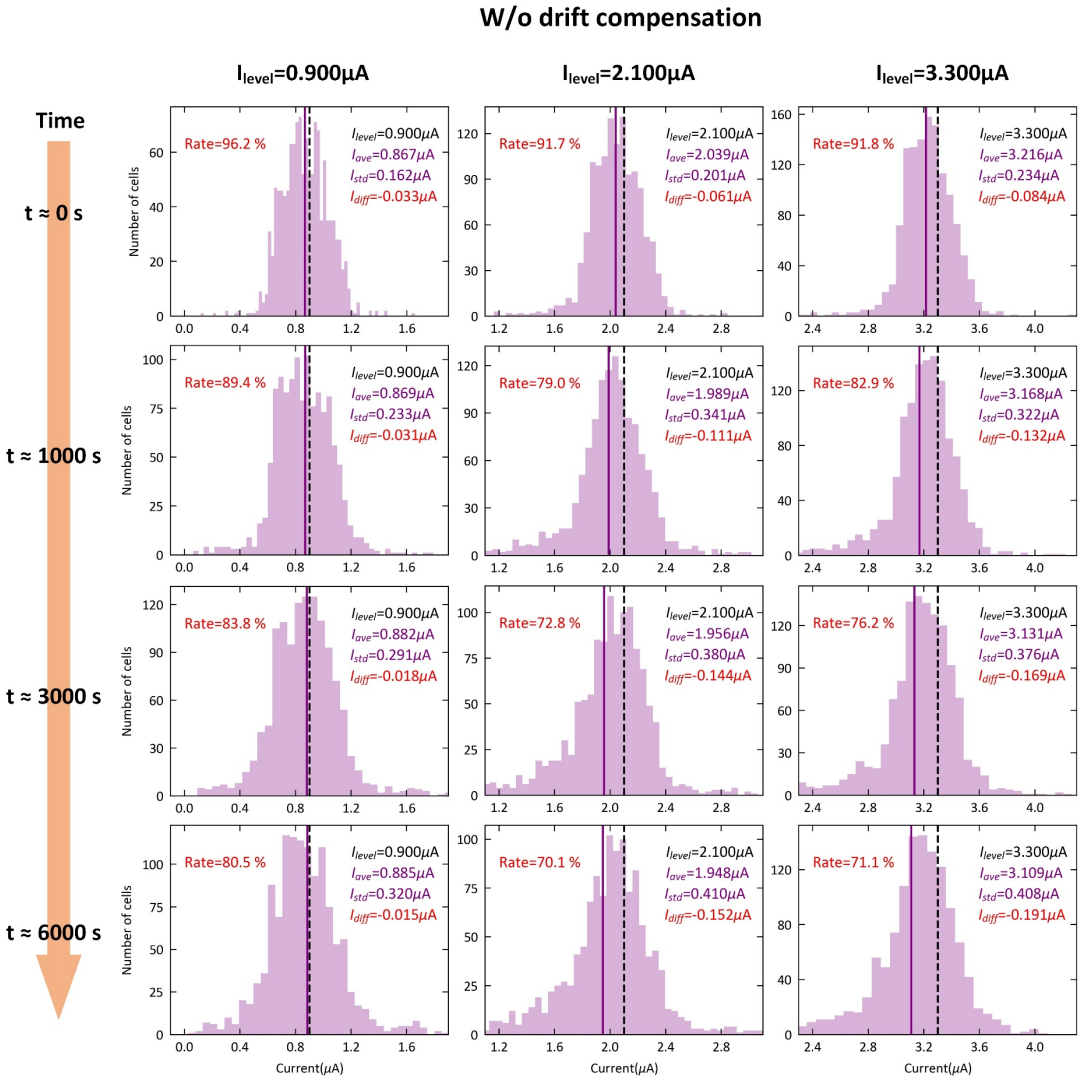
Supplementary Figure 6 | Average value mismatch degree (A_d) distribution after relaxation. a, Distribution of 100×100 weight matrix's A_d in the experiment. **b,** Histogram plot of A_d.



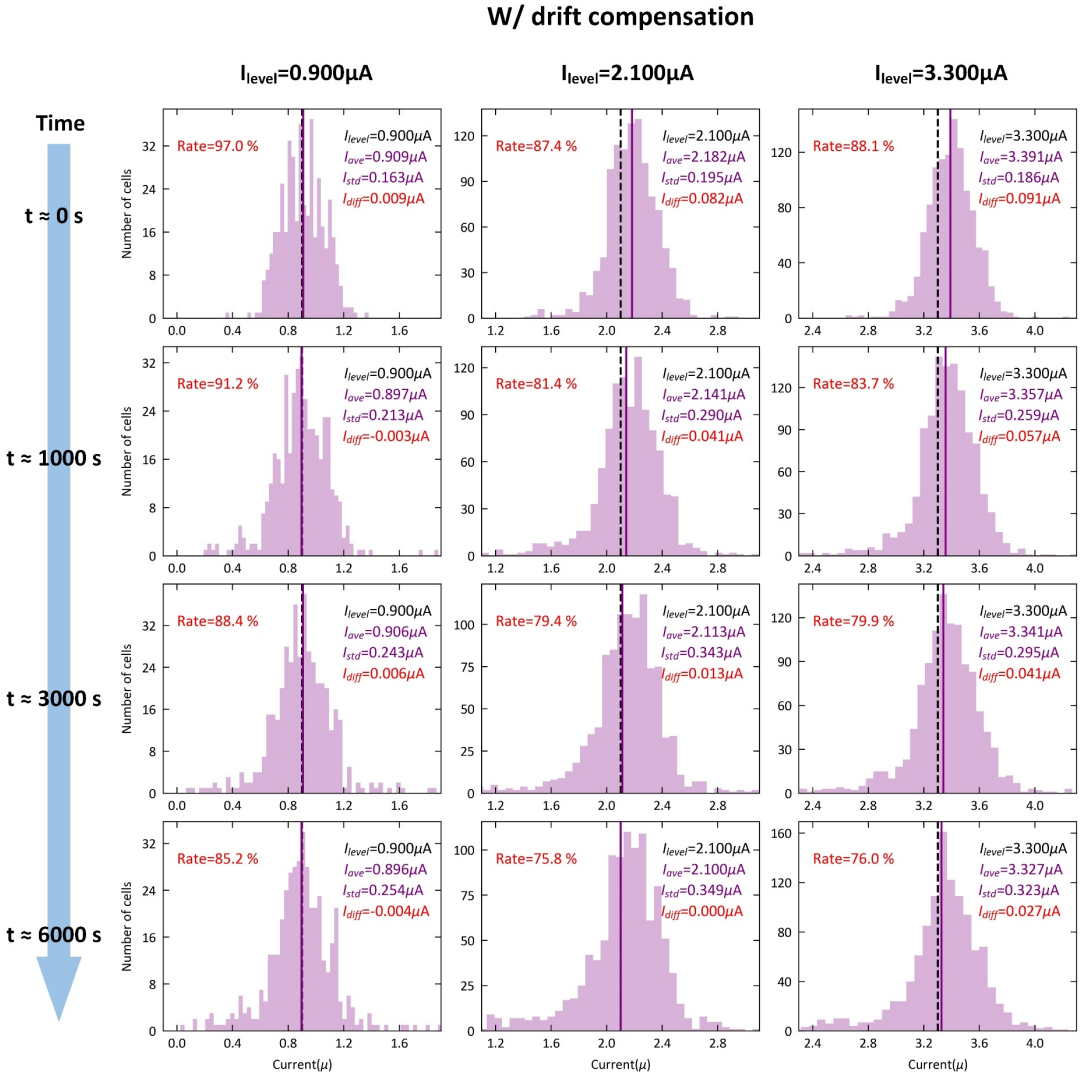
Supplementary Figure 7 | Typical single-weight distribution shapes with different mismatch degree (S_d) after relaxation in the experiment. a-i, S_d is 0.02, 0.03, 0.06, 0.09, 0.12, 0.15, 0.18, 0.21, and 0.24, respectively. Each distribution shape is obtained from 360-times read results.



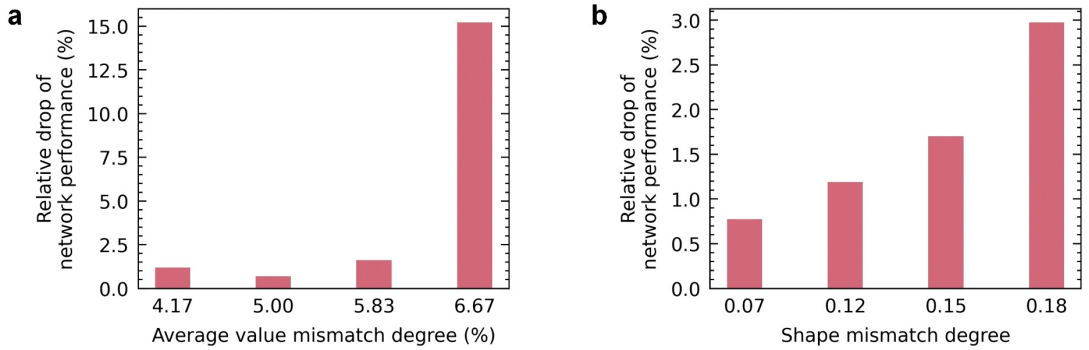
Supplementary Figure 8 | Shape mismatch degree (S_d) distribution after relaxation. a, Measured distribution of 100 ×100 weight matrix's S_d. **b,** Histogram plot of S_d.



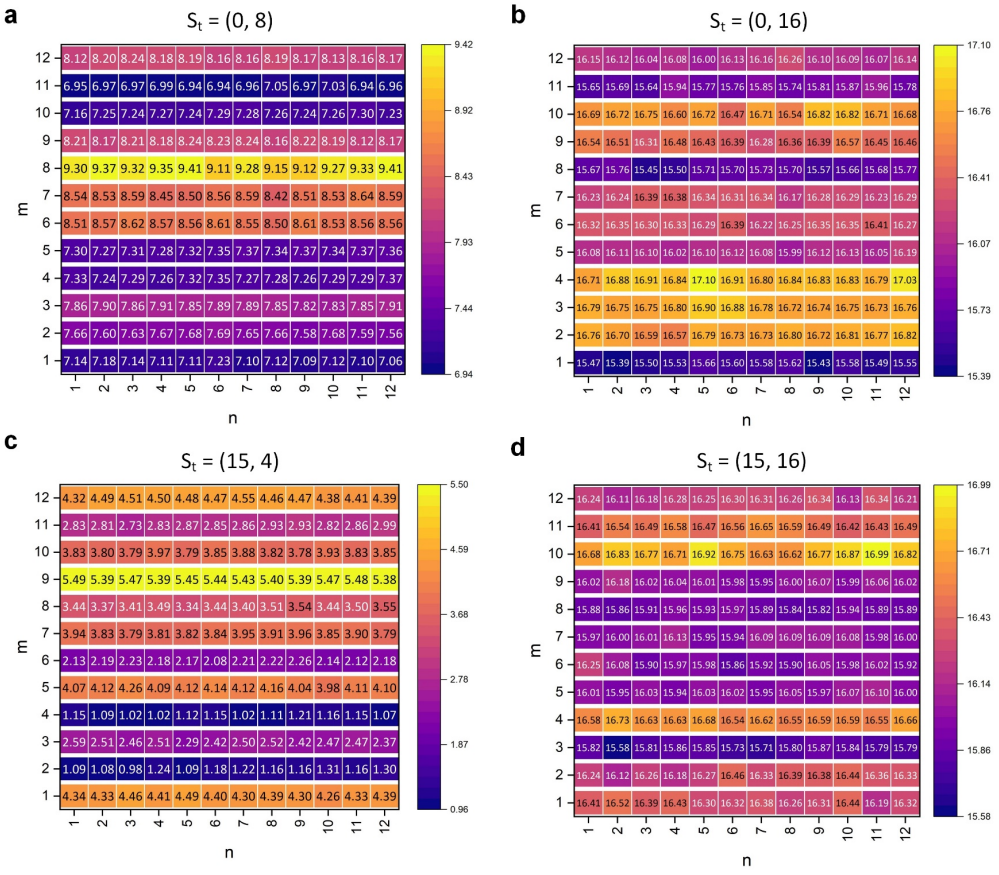
Supplementary Figure 9 | Evolution of devices current distribution without drift compensation. Each histogram shows the current distribution of the group of devices at different escaped time. The average current of devices I_{ave} is represented by a pink line and I_{level} is represented by a black dot line in each histogram.



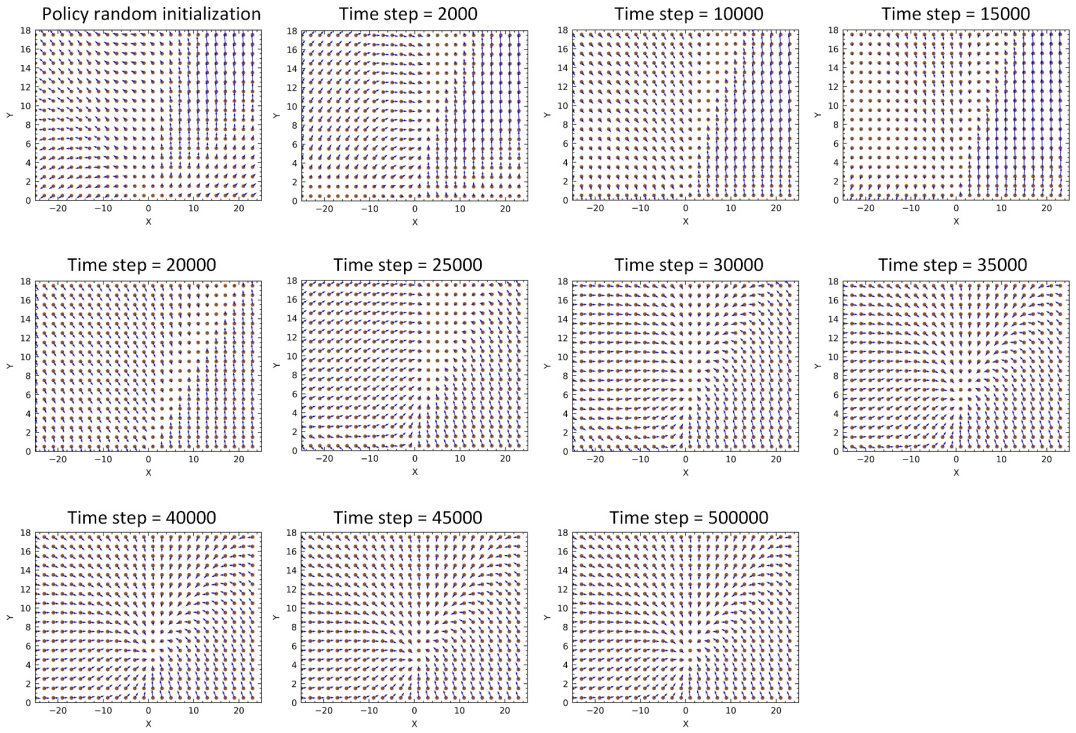
Supplementary Figure 10 | Evolution of devices current distribution with drift compensation. Each histogram shows the current distribution of the group of devices at different escaped time. The average current of devices I_{ave} is represented by a pink line and I_{level} is represented by a black dot line in each histogram.



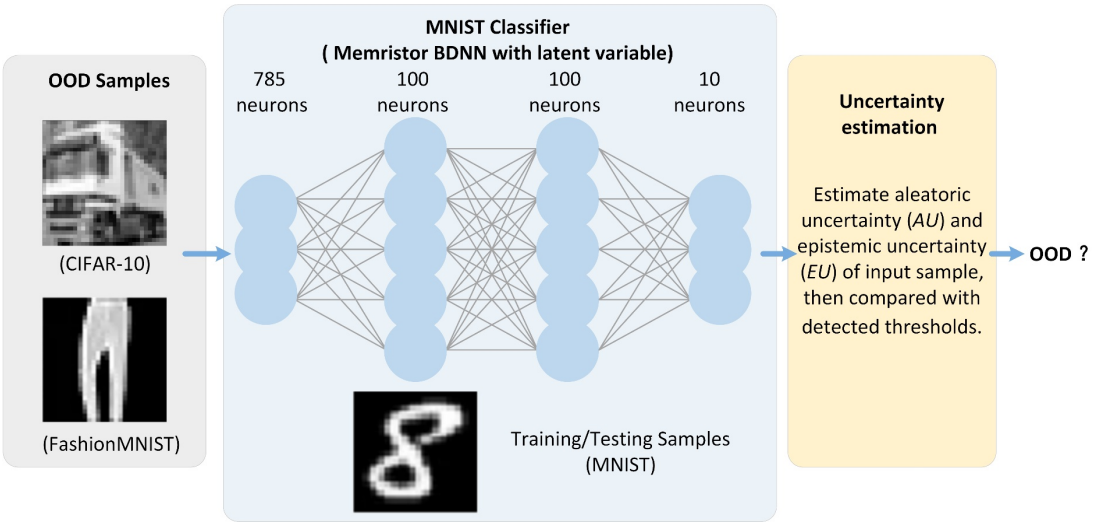
Supplementary Figure 11 | Impacts on network performance under different mismatch degree. a, Different average value mismatch degree (A_d). **b,** Different shape mismatch degree (S_d).



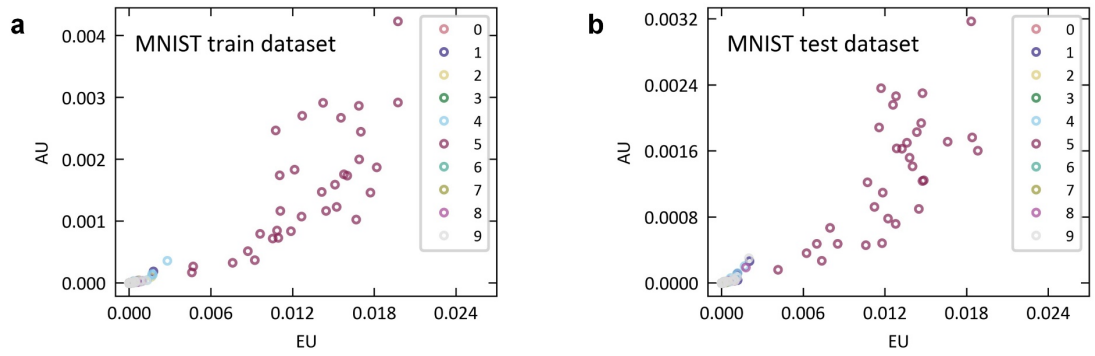
Supplementary Figure 12 | 12×12 distributional prediction matrix for 4 cases of present boat location. a, S_t =(0, 8). **b,** S_t =(0, 16). c, S_t =(15, 4). **d,** S_t =(15,16)



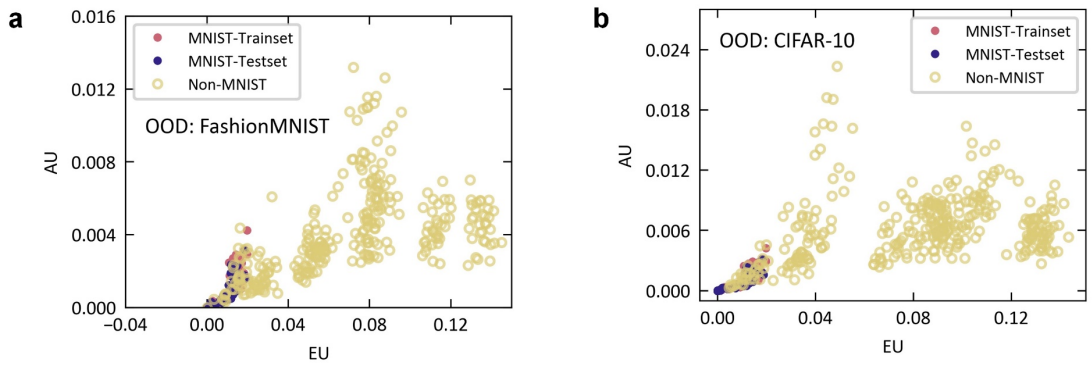
Supplementary Figure 13 | Visualization of actions chosen by the policy network during policy search. The paddling action direction and amplitude chosen by the network are denoted by arrows at the various locations, and locations are indicated by dots.



Supplementary Figure 14 | Illustration of the Out-of-distribution (OOD) detection task.



Supplementary Figure 15 | Aleatoric uncertainty (*AU*) and epistemic uncertainty (*EU*) of MNIST images. a, for MNIST training dataset. **b,** for MNIST testing dataset. Each point represents for an image sample, and point color indicates the class label of the images. Each class is shown with 32 random-pick images in the figures.



Supplementary Figure 16 | Aleatoric uncertainty (*AU*) and epistemic uncertainty (*EU*) of OOD images. a, for Fashion-MNIST dataset. **b,** for CIFAR-10 dataset. It also shows images in MNIST training and testing dataset. Each point represents for an image sample. Each class in Fashion-MNIST or CIFAR-10 dataset is shown with 32 random-pick images in the figures.

Supplementary Tables

Supplementary Table 1 | List of policy search parameters.

Quantity	**Value**
Discount factor	0.99
Timesteps per actor per update	512
Clipping parameter	0.2
Entropy loss weight	0.03
Advantage estimation	0.95
Aleatoric risk-sensitive parameter	0.95
Epistemic risk-sensitive parameter	40

Supplementary Table 2 | Architecture of the BDNN used for the storm coast task.

Weight Layer	**Input neurons of weight layer (n_{in_dim})***	**Output neurons of weight layer (n_{out_dim})**	**No. of random weights (or samples/multiplications/additions per forward pass) (n_{weight})**
1	6	100	600
2	101	100	10100
3	101	2	202

* With bias input

Supplementary Table 3 | Detailed energy cost of each circuitry module at two technology nodes with an 8-bit input per forward process.

Process / Modules	130 nm		28 nm	
	Energy cost/nJ	**Latency/ns**	**Energy cost/nJ**	**Latency/ns**
Array	13.3	/	4.2	/
SL driver	0.6	6.8	0.08	0.006
WL driver	0.2	7.2	0.1	0.03
ADC	202.0	400.0	34.9	240.0
Shift & adder	0.9	60.3	0.04	15.5
ReLu	0.003	0.6	0.03	0.2
Total	**217.0**	**476.9**	**39.3**	**255.7**

Supplementary Table 4 | NVIDIA Tesla A100 performance metrics.

Features	**Values**
TDP [1]	P_{GPU}=400 W
Peak OPS (32-bit floating-point number computation) [1]	OPS_{GPU}=19.5 TOPS
FP32 cores [1]	N_{core}=6912
High bandwidth memory (HBM) latency [2]	T_{GPU}=404 ns
Random number generation throughput [3] (normal distribution, 32-bit floating-point number)	S_{GPU}=236.6 GSamples/sec

Supplementary Table 5 | Energy cost and latency breakdown of the GPU in performing a forward process.

Steps	**Energy cost/μJ**	**Latency/μs**
Sample	5.53	1.26
VMM	0.37	1.21
Total	$\boldsymbol{E_{GPU}}$ **=5.90**	$\boldsymbol{T_{GPU}}$ **=2.47**

Supplementary Table 6 | Comparison of the ESCIM system and current state-of-the-art randomness weight implementation in performing a forward process.

References	**Hardware platform**	**Energy cost/nJ**	**Latency/ns**
GPU-A100*	CMOS (GPU)	5900	2470
Cai, R. *et al.*[8]	CMOS (FPGA)	2400	3110
Dalgaty, T. *et al.* [9]	Memristor	1064	-
Wu, N. *et al.* [10]	Thin-film transistors	1798	1500
Kang, K. *et al.* [11]	CMOS (AISC)	113 *†*	3420
This work	**Memristor**	**39.3**	**255.7**

**Obtained from Supplementary note 4; † Not included DAC and ADC.*

Supplementary Table 7 | Classification accuracy on MNIST and OOD detection accuracy on Fashion-MNIST and CIFAR-10 datasets for two types of networks.

Networks	Classification accuracy	OOD detection accuracy*	
	MNIST	Fashion-MNIST	CIFAR-10
Memristor BDNN	97.76%	78.33%	93.80%
Traditional DNN†	97.63%	Not supported	Not supported

***Percentage of correct detections in the input images; † The network structure is the same as the BDNN and all weights are represented by three memristors; Considering the same device variations as the BDNN.*

Supplementary Table 8 | Comparison of energy cost and latency between GPU and ESCIM system in performing the same OOD detection task with the same BDNN ($N_{prediction} = M \times N = 32 \times 32 = 1024$).

Metrics	A100 GPU (7-nm CMOS technology)	ESCIM system (130 nm)	ESCIM system (28 nm)
Energy cost/ μJ	53495.9	247.3	73.3
Latency/ μs	2872.2	490.1	261.9

Supplementary Notes

Supplementary note 1: Drift current compensation

The relaxation could have two effects on the read current distribution: the average current value drifts, and the shape of the distribution could appear several peaks.

Because the average drift current caused by relaxation is somewhat predictable, the network's weights can be compensated to reduce the influence of current drift. First, we conduct a statistical analysis of the average drift current δI with respect to the initial current under various current states. We program 1890 cells into an expected current state I_{level}. Then, the cells are measured at different escaped time. The drift current δI_{cell} is the difference between the present read current I_{read} and the initial current I_{read_init} at $t = 0\ s$:

$$\delta I_{cell} = I_{read} - I_{read\_init}.$$

Then, we calculate the average drift current δI of all cells:

$$\delta I = AVE(\delta I_{cell})$$

where $AVE(x)$ is the average of data x. The average drift current δI evolution over time of 7 expected current states I_{level} is shown in Extended Data Fig. 4.

Finally, to correct the current value in the memristor BDNN, the weight I^{opt} which is optimized by the training algorithm, is compensated with corresponding drift δI:

$$I_t = I^{opt} - \delta I$$

where I_t is a compensated current. Finally, the compensated current I_t is programmed into device, instead of I^{opt}. Since I_t evolves during time, it is impossible to program the compensated current I_t into the device as a function of time due to the limitation of system overhead. But we reasonably choose the average drift current of about 3000s as the correction value δI. From the Extended Data Fig. 4, we can see that ~90% of the

total drift has occurred when the time escapes 3000s for the current states 2.1μA, 2.7μA, 3.3μA and 3.9μA. Then, the drift current increases at a slower pace over longer time scales. Compared with these current states, the drift of the rest states, 0.4μA, 0.9μA and 1.5μA, almost keep constant over long-time scales. Taking into account both short-term and long-term system performance, we choose the drift current of about 3000s as the correction value δI in our experiment.

For the potential implementation costs of drift compensation, we add a flow chart of the operations as shown in Extended Data Fig. 5 to show the process of ex-situ training more clearly. We can see that the drift compensation technique can be added to the tool chain such as a compiler, and it is a one-time correction after training, which basically does not increase the cost for the integrated circuit.

We evaluate the network performance experimentally with and without drift compensation, respectively. As shown in Extended Data Fig. 8, the time is counted from the moment when the weight-programming process is finished. It reveals that the network performance is almost maintained when the weights are compensated against the current drift.

Regarding the shape of the weight distribution, serval peaks may arise as a result of the three memristor devices' relaxation. However, we discover that the percentage of such multi-peaks weights is very low in our experiment. Besides, as detailed further below (Supplementary note 3), the compensated weight mismatch caused by relaxation have a minor impact on network performance.

Supplementary note 2: Mismatch between memristor weight and target weight caused by device-to-device variability, and the impact on network performance.

We carefully analyze the mismatch between memristor weight w_m and target weight w_t caused by device-to-device variability, and the impact on network performance. Since the weight in BDNN is a distribution, the average value and the distributed shape are two key properties. So, we compare the average value and the distributed shape of memristor weight with target weight, respectively.

The average value of memristor weight ($\overline{w}_m$) is obtained from 360-times read results. The size of hidden layer (100×100 weight matrix) covers most (92.66%) of whole weights in BDNN, so we focus on weights belonging to this hidden layer. To compare the average value mismatch, we use the relative difference percent between $\overline{w}_m$ and average target weight $\overline{w}_t$ as average value mismatch degree indicator A_d:

$$A_d = \frac{\overline{w}_m - \overline{w}_t}{w_{max}}$$

where w_{max} is the maximum weight value, which is limited by the conductance range of the three memristor weights. The calculated average value mismatch degree A_d of different weights is shown in Supplementary Fig. 3. We can see that 85.0% of memristor average values are around their target values (A_d < 4.8%), and only 15.0% weights show relative large variations (A_d > 4.8%).

To compare the distributed shape, the measured memristor weights and target weights are first subtracted their average values, respectively. Then, the difference between them is calculated as the shape mismatch degree indicator S_d:

$$S_d = JS(w_m - \overline{w}_m, w_t - \overline{w}_t)$$

where JS is Jensen–Shannon (JS) divergence function which can measure the difference of two distributions. In this case, the higher JS divergence value is, the greater shape mismatch degree S_d is. Supplementary Fig. 4 shows several typical single-weight (the read current sum of three devices) distribution plots with different

S_d in our experiment. As we can see, when the shape of weight matches with the target shape well, S_d value is small (<0.12). In contrast, when the weight distribution shows multiple peaks or is flattener than target shape, S_d increases up to 0.24.

Then we calculate S_d of each weight in the hidden layer and present the distribution and histogram in Supplementary Fig. 5. It can be found that around 75.3% of S_d is less than 0.06, and only 6.0% of S_d is greater than 0.12. These results show that the majority of the memristor weight shape matches the target shape well, with only a small portion showing some mismatch.

Furthermore, we investigate the impact on network performance through simulation, under different average value mismatch A_d and shape mismatch S_d. The network performance is evaluated by Jensen–Shannon (JS) divergence as discussed in the manuscript. For the average mismatch simulation, we assume 15.0% of whole weights in the network are not matched well with target one. For the shape mismatch simulation, the mismatch weight percent is about 6.0%. The mismatch percent values are consistent with our measured data. The impact on network performance under different mismatch A_d and S_d is shown in Supplementary Fig. 11. It can be found that the network performance shows very small degradation when mismatch A_d and S_d are smaller than 5.83% and 0.15, respectively. And the shape mismatch has less impact on network performance than average value mismatch. This result shows that the BDNN has some tolerance ability to the mismatch of average value and distributed shape. We think such tolerance ability originates from the redundancy of the deep neural network, just like the tolerance ability of the traditional DNNs to some level of weight variations.

Supplementary note 3: Mismatch between memristor weight and target weight caused by relaxation, and the impact on network performance.

We carefully analyze the mismatch between memristor weight w_{mr} after relaxation and target weight w_t caused by device-to-device variability and relaxation, and the impact on network performance. Sine the weight in BDNN is a random variable, the average value and the distributed shape are two key properties. So, we compare the average value and the distributed shape of memristor weight with the target weight, respectively.

The average value of weight ($\overline{w}_{mr}$) based on the total read currents of three memristors is obtained from 360-times read results after relaxation. The size of the hidden layer (100×100 weight matrix) covers most (92.66%) of whole weights in BDNN, so we focus on the weights belonging to this hidden layer. To compare the average value mismatch, we use the relative difference percent between $\overline{w}_{mr}$ and the average target weight $\overline{w}_t$ as the average value mismatch degree indicator (A_d):

$$A_d = \frac{\overline{w}_{mr} - \overline{w}_t}{w_{max}}$$

where w_{max} is the maximum weight value, which is limited by the conductance range of the three memristor weights. The calculated average value mismatch degree A_d of different weights is shown in Supplementary Fig. 6. We can see that 15.0% weights' A_d are greater than 5.8%, and 80.0% of weights' A_d are less than 5.0%.

To compare the distributed shape, the measured memristor weights and target weights are first subtracted their average values, respectively. Then, the difference between them is calculated as the shape mismatch degree indicator (S_d):

$$S_d = JS(w_{mr} - \overline{w}_{mr}, w_t - \overline{w}_t)$$

where JS is Jensen–Shannon (JS) divergence function which can measure the difference of two distributions. In this case, the higher JS divergence value is, the greater shape mismatch degree S_d is. Supplementary Fig. 7 shows several typical

single-weight (the read current sum of three devices) distribution plots with different S_d in our experiment. As we can see, when the shape of weight matches with the target shape well, S_d value is small (<0.12). In contrast, when the weight distribution shows multiple peaks or is flattener than target shape, S_d increases up to 0.24.

Then we calculate S_d of each weight in the hidden layer and shown the distribution and histogram in Supplementary Fig. 8. It can be found that around 73.6% of S_d is less than 0.060, and only 6.0% S_d is greater than 0.124. These results show that the majority of the memristor weight shape matches the target shape well, with only a small portion showing some mismatch.

Furthermore, we investigate the impact on network performance through simulation, under different average value mismatch A_d and shape mismatch S_d. The network performance is evaluated by Jensen–Shannon (JS) divergence as discussed in the manuscript. For the average mismatch simulation, we assume 15.0% of whole weights in the network are not matched well with target one. For the shape mismatch simulation, the mismatch weight percent is about 6.0%. The mismatch percent values are consistent with our measured data.

The impact on network performance under different mismatch A_d and S_d is shown in Supplementary Fig. 11. It can be found that the network performance shows very small degradation when mismatch A_d and S_d are smaller than 5.83% and 0.15, respectively. And the shape mismatch has less impact on network performance than average value mismatch.

Moreover, to show the conductance drift on single cells and provide a thorough explanation, we add an additional comparison between without and with compensation. Compared to the case without compensation, the number of cells after relaxation which are still around expected state is greater for the case with drift compensation (that is, the deviation between average device conductance and the expected state is smaller as the quantitative analysis as follows). Moreover, since the deviation is small, the

distribution of the device conductance around the expected state is more symmetrical. At the system application level of matrix multiplication and addition, the positive and negative conductance fluctuation between multiple cells can better cancel each other out. Blow is our discussion.

For the case without compensation, here we statistically analyze individual devices current which has described in Supplementary note 1 for comparison. We program 1890 cells into an expected current state I_{level}. Then, the cells are measured at different escaped time t. We select $I_{level} = 0.9\ \mu A,\ 2.1\ \mu A$ and $3.3\ \mu A$ and escaped time: $t \approx 0\ s,\ 1000\ s,\ 3000\ s$ and $6000\ s$. The result of the case without compensation is shown in Supplementary Fig. 9. Each histogram shows the distribution of cells current I_{read} at different escaped time t. We calculate the standard deviation and average of I_{read}, noted as I_{std} and I_{ave}. And the difference between I_{ave} and I_{level} as I_{diff} to show the deviation:

$$I_{diff} = I_{ave} - I_{level}\ .$$

I_{ave} is represented by a pink line and I_{level} is represented by a black dot line in each histogram. The percent of devices which are near the current I_{level} is also calculated:

$$Rate = \frac{n_{within}}{n_{total}}$$

where n_{within} is the number of devices which fall within $I_{level} \pm \Delta I$ (ΔI =0.3μA, which is an error margin used in our main text), and n_{total} is the total number of devices. The value of $Rate$ is also shown in each histogram.

As for the case with compensation, we measure the devices in the memristor BDNN with compensation. The expected weight I^{opt}, which is optimized by the training algorithm, is compensated with corresponding drift δI:

$$I_t = I^{opt} - \delta I.$$

where I_t is a compensated current, which is directly programmed into devices. We can

select the devices, whose expected current state $I^{opt} = I_{level} = 0.9\,\mu A,\ 2.1\,\mu A$ and $3.3\,\mu A$, to show what exactly happens for devices in the memristor BDNN. The results of the case with compensation are shown in Supplementary Fig. 10. The value of I_{diff} and $Rate$ are also calculated like Supplementary Fig. 9 for comparison.

As shown in Supplementary Fig. 9 and Supplementary Fig. 10, a spread of the conductance occurs after programming during time for the case with and without compensation. Just after programming, the absolute value of I_{diff} and $Rate$ without compensation is slightly better than the case with compensation. It results in a slightly better network performance without compensation as shown in Extended Data Fig. 8. But both two cases have a good network performance, since the weight mismatch is quite small just after programming. However, as time goes on, the absolute value of I_{diff} and $Rate$ without compensation is getting worse than the case with compensation. Comparing to the case with compensation, the percent of devices which fall within $I_{level} \pm \Delta I$ is ~5% less in the case without compensation when t=6000s. And we can also see that the devices average conductance with compensation deviates from the expected state is smaller. Moreover, since the device conductance could randomly spread to a lower or higher conductance due to relaxation effects, the spreading distribution of the device conductance around the current state I_{level} is more symmetrical in the case with compensation as shown in Supplementary Fig. 10. Comparing to the case without compensation, the positive and negative conductance fluctuation between multiple devices can better cancel each other out when accumulated the devices' current. Hence, the performance of BDNN with compensation is more robust than the case without compensation.

Supplementary note 4: Performance analysis of the ESCIM system compared to the digital NVIDIA Tesla A100 GPU.

We estimate the performance (energy cost and latency) of the ESCIM system and NVIDIA Tesla A100 GPU in uncertainty decomposition in the risk-sensitive RL storm coast task.

To ensure a straightforward and fair comparison, we assume that both systems implemented the same network architecture used for the storm coast task. Each layer of the BDNN comprise n_{weight} random normally distributed weights (please refer to Supplementary Table 2). Since each entry in the weight matrix is a random variable, which need to be sampled during each prediction, the number of samples, multiplications and additions per forward pass are the same, i.e., $n_{weight} = n_{sample} = n_{\text{multiplication}} = n_{\text{addition}}$ (see Supplementary Table 2 for each layer details). The prediction process is executed $M \times N$ times to decompose the prediction uncertainty into aleatoric and epistemic terms in the RL storm coast task. Therefore, the forward passes number of the BDNN is $N_{prediction} = M \times N = 12 \times 12 = 144$. Next, we will discuss and estimate the energy cost and latency of each layer to obtain those of a prediction.

ESCIM system: We assume that three layers of the BDNN are mapped onto the three memristor cores of the ESCIM system. The memristor cores operate in a pipeline fashion. Each memristor core comprise one $n_{in\_dim} \times n_{out\_dim}$ (see Supplementary Table 2 for details) memristor array and all the essential peripheral circuits, including drivers, ADCs, shift & adder components and activation functions (Extended Data Fig. 9). The ADCs could complete the conversation step during the voltage pulse reading period. The read voltage pulse widths are 50 ns@0.2 V and 30 ns@0.2 V at the 130- and 28-nm technology nodes, respectively. At the 130-nm technology node, the parameter of the ADC block is obtained in [4], and the parameters associated with the memristor array and other peripheral circuitry blocks are obtained with the simulator.

At the 28-nm technology node, we designed and evaluated a 28-nm ADC based on [5]. And all the parameters are extracted using the simulated 28-nm technology node circuits, except for the driver circuits which is simulated at the 65-nm technology node. And it could be further reduced upon transistor scaling. The typical energy cost and latency are obtained with the XPEsim simulator [6]. The detailed energy cost of each block is listed in Supplementary Table 3, which indicates the performance of two technology nodes given an input of an 8-bit read pulse. Hence, we could determine that the energy cost and latency of the ESCIM system at the 130-nm technology node are $E_{pred@130}$ =217.0 nJ and $T_{pred@130}$ =476.9 ns, respectively, per forward pass. At the 28-nm technology node, the energy cost and latency of the ESCIM system reached $E_{pred@28}$ =39.3 nJ and $T_{pred@28}$ =255.7 ns, respectively. The total energy cost and latency of prediction uncertainty decomposition could be easily obtained by multiplying the total forward passes $N_{prediction}$ =144. For the 130nm case as an example:

$$\begin{cases} E_{total@130} = E_{pred@130} \times N_{prediction} = 217.0 \times 144 = 31248.0\ (nJ) \\ T_{total@130} = T_{pred@130} \times N_{prediction} = 476.9 \times 144 = 68673.6\ (ns) \end{cases}.$$

NVIDIA A100 GPU-based system: For simplicity, we assume that the A100 GPU operate at the peak random number generation throughput, i.e., the peak operations per second (peak OPS, OPS_{GPU}=19.5 TOPS), and thermal design power (TDP, P_{GPU}=400 W) (please refer to Supplementary Table 4). To accomplish a forward pass of the BDNN, the GPU should first obtain a normally distributed weight sample and then perform VMM operations layer by layer.

Regarding the sample and VMM steps, we consider the GPU relative occupancy O_{sample} and O_{VMM} values, respectively, to carefully estimate the energy cost of a layer. The occupancy in the sample step, O_{sample}, is obtained in [7] according to the number of samples n_{sample} for a layer. Note that the relative occupancy of [7] is based on uniformly distributed pseudorandom number generation, whose occupancy could be lower than that of normally distributed pseudorandom number generation. Hence, the

O_{sample} of three layers are 30%, 30% and 30%, respectively. The average occupancy of the VMM step, O_{VMM}, could be obtained by the number of weights n_{weight} in a layer and cores in the GPU N_{core}:

$$O_{VMM} = \frac{n_{weight}/N_{core}}{\lceil n_{weight}/N_{core} \rceil}$$

where $\lceil x \rceil$ is the ceiling function. Hence, the O_{VMM} of three layers are 17.36%, 73.06% and 5.84%, respectively.

Then, according to the given sampling throughput with a normal distribution ($S_{GPU} = 236.6$ GSamples/sec), the estimated energy cost and latency of the sample step for a layer are:

$$E_{sample} = (n_{sample}/S_{GPU}) * P_{GPU} * O_{sample}$$

$$T_{sample} = T_{GPU} + n_{sample}/S_{GPU}$$

where T_{GPU}=404 ns refers to access to the HBM memory for weight sample storage.

Based on the given peak OPS (OPS_{GPU}=19.5 TOPS), the estimated energy cost and latency of the VMM operation step for a layer are:

$$E_{VMM} = (n_{\text{multiplication}} + n_{\text{addition}})/OPS_{GPU} * P_{GPU} * O_{VMM}$$

$$T_{VMM} = T_{GPU} + (n_{\text{multiplication}} + n_{\text{addition}})/OPS_{GPU}$$

By accumulating the energy cost and latency of three layer, we can obtain the computational cost in performing a forward pass as listed in Supplementary Table 5. The energy cost and latency of the GPU in decomposing the prediction uncertainty could be easily obtained by multiplying the total forward passes $N_{prediction}$.

Summary of the comparison with GPU: Extended Data Fig. 10 summarizes the energy cost and latency of the GPU and ESCIM system in performing uncertainty decomposition in the risk-sensitive RL storm coast task. Compared to the NVIDIA Tesla A100 GPU, the energy cost of the memristor-based ESCIM system is

approximately 27 times better at 130 nm and 150 times better at 28 nm. In terms of latency, that of the ESCIM system is 5 times better at 130 nm and 10 times better at 28 nm than that of the GPU. It should be noted that the NVIDIA Tesla A100 GPU is based on 7-nm CMOS technology, which notably is more aggressively scaled than are the nodes considered in our ESCIM system.

Comparison with other difference implementation approaches of BDNN: We also compared the computational cost of the ESCIM system with current state-of-the-art randomness weight implementations. We calculate the average energy cost of a weight and the average time cost of a layer, according to the reported data in related works. The total energy cost and the latency are then calculated proportionally based on the number of weights and layers of the BDNN (i.e., 10,902 weights and 3 layers), respectively. The calculated results are shown in Supplementary Table 6.

Supplementary note 5: Anomalous images classification benchmark.

In order to demonstrate how to deal with the uncertainty prediction, we added **an anomalous image classification task based on uncertainty estimation**. In this classification task, a trained BDNN classifier can categorize most of the images from the training and testing datasets correctly. However, when the BDNN is given anomalous images which are not from the training and testing datasets, the prediction uncertainty will rise significantly. Because anomalous images deviate from the training data distribution (i.e., out-of-distribution images), this classification task is also called an out-of-distribution (OOD) detection task.

Our simulation task involves utilizing an MNIST classifier to determine whether the input images are OOD images using the uncertainty estimation method (as shown in Supplementary Fig. 14). The MNIST classifier is a memristor BDNN (785×100×100×10) with latent variable input (as presented in Method 7), which is trained on the MNIST trainset through our memristor variational inference (as presented in Method 4). On the MNIST testset, its final classification accuracy is 97.76%.

To detect OOD images, we must first get appropriate detected thresholds from the MNIST dataset. So, using the proposed uncertainty decomposition method (as presented in Methods 8), we estimate the aleatoric uncertainty (AU) and epistemic uncertainty (EU) values of the MNIST training and testing datasets. The results are shown in Supplementary Fig. 15. The appropriate detected thresholds $AU_{TH} = 0.003$ and $EU_{TH} = 0.02$ are acquired. If the AU or EU of an input image is greater than AU_{TH} or EU_{TH}, the input image is classified as an OOD image.

Then, we use Fashion-MNIST dataset and resized CIFAR-10 dataset as OOD input images. We calculate the AU and EU values for each input image, and compare them with AU_{TH} or EU_{TH} to determinate whether the input is an OOD image. The AU and EU of the images from the two datasets are shown in Supplementary Fig. 16. We

can see that the majority of OOD images have high EU or AU, indicating a lack of knowledge and large aleatoric noise of OOD images.

Next, the accuracy of OOD detection is measured as the percentage of correct detections in the input images. We randomly select 15,000 images from the Fashion-MNIST and CIFAR-10 datasets as input, respectively. The detection accuracy results are shown in Supplementary Table 7. The accuracy of OOD detection in the two datasets is 78.33% and 93.80%, respectively. CIFAR-10 outperforms Fashion-MNIST in terms of accuracy, which is due to the fact that the former has obviously different styles, whereas the latter has a similar background style with MNIST. As shown in Table R1-3, in comparison to traditional DNNs, the memristor BDNN exhibits comparable accuracy in the MNIST classification task. Most notably, it can perform OOD detection while traditional DNNs cannot, owing to its ability to estimate uncertainty.

Finally, as shown in Supplementary Table 8, we calculate the energy cost and latency of the ESCIM system and NVIDIA Tesla A100 GPU in this OOD detection task. The energy cost of the memristor-based ESCIM system is roughly 200 times lower at 130 nm and 700 times lower at 28 nm when compared to the NVIDIA Tesla A100 GPU. In terms of latency, the ESCIM system outperforms the GPU by 6 times at 130 nm and 11 times at 28 nm.

Supplementary Video

Captions for Supplementary Video 1

The movie shows result for the risk-sensitive RL storm coast task. A boat trajectory passes through the low-epistemic uncertainty sea area and low environmental stochasticity. The epistemic and aleatoric uncertainties are warning when boat is in sea areas with a high environmental stochasticity and high epistemic uncertainty, respectively. The warnings guide the boat paddling upwards so that leaving the high uncertainties areas. The stable point occurred at a suitable distance from the coast due to consideration of the uncertainties.

Supplementary Information References

[1] Nvidia ampere architecture whitepaper: Nvidia A100 tensor core GPU architecture. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf , accessed: 2022-03-08

[2] How GPU Computing Works, Stephen Jones, Nvidia. https://www.nvidia.com/en-us/on-demand/session/gtcspring21-s31151/ , accessed: 2022-03-08

[3] Nvidia curand: Random number generation on nvidia gpus. https://developer.nvidia.com/curand , accessed: 2022-03-08

[4] S. M. Louwsma, J. M. van Tuijl, M. Vertregt and B. Nauta, "A 1.35 GS/s, 10b, 175 mW time-interleaved AD converter in 0.13 μm CMOS," 2007 IEEE Symposium on VLSI Circuits, 2007, pp. 62-63, doi: 10.1109/VLSIC.2007.4342766.

[5] B. R. Gregoire and U. Moon, "An Over-60 dB True Rail-to-Rail Performance Using Correlated Level Shifting and an Opamp With Only 30 dB Loop Gain," in IEEE Journal of Solid-State Circuits, vol. 43, no. 12, pp. 2620-2630, Dec. 2008, doi: 10.1109/JSSC.2008.2006312.

[6] Zhang, W. et al. Design guidelines of RRAM-based neural-processing unit: a joint device– circuit–algorithm analysis. In 2019 56th ACM/IEEE Design Automation Conference (DAC) 63.1 (IEEE, 2019).

[7] Pascuzzi, Vincent R. and Mehdi Goli. “Achieving near native runtime performance and cross-platform performance portability for random number generation through SYCL interoperability.” *ArXiv* abs/2109.01329 (2021)

[8] Cai, R. *et al.*, “VIBNN: Hardware Acceleration of Bayesian Neural Networks,” in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2018, pp. 476–488. doi: 10.1145/3173162.3173212.

[9] Dalgaty, T. *et al.* In situ learning using intrinsic memristor variability via Markov

chain Monte Carlo sampling. *Nat Electron* **4**, 151–161 (2021).

[10] Wu, N. *et al.* A Real-Time and Energy-Efficient Implementation of Difference-of-Normal with Flexible Thin-Film Transistors. in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* 455–460 (IEEE, 2016). doi:10.1109/ISVLSI.2016.87.

[11] Kang, K. & Shibata, T. An On-Chip-Trainable Normal-Kernel Analog Support Vector Machine. *IEEE Trans. Circuits Syst. I* **57**, 1513–1524 (2010).