nature electronics

Article

https://doi.org/10.1038/s41928-022-00886-9

A memristor-based Bayesian machine

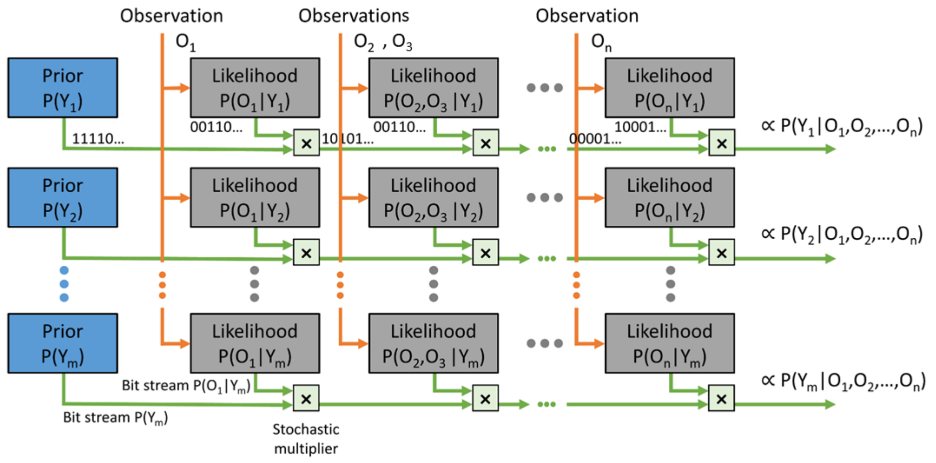
Contents

Supplementary Note 1: Adaptation of the Bayesian machine for situations where some observations are not conditionally independent

All results in the main paper are presented in cases where all observations may be considered conditionally independent given the inferred variable Y (naive Bayesian inference). This is not always the case. In particular, in a situation where two redundant sensors measure the same phenomenon, they may never be considered independent in a good model, even conditionally to the inferred variable. For example, if observations O_2 and O_3 are not conditionally-independent, the $p(O_2, O_3|Y = y)$ likelihood may not be factorized, and an appropriate Bayesian inference model may read

$$p(Y = y|O_1, O_2, ..., O_n) \propto p(O_1|Y = y) \times p(O_2, O_3|Y = y) \times = ... \times p(O_n|Y = y) \times p(Y = y). \quad (1)$$

This model can still be implemented by a memristor-based Bayesian machine. In that case, observations O_2 and O_3 should be pooled into a single column, and their joint value should be used to address the likelihood arrays of this column (see Supplementary Fig. 1). Nevertheless, it should be highlighted that the memory cost of the Bayesian machine grows with the number of non-conditionally independent variables.



Supplementary Figure 1. Architecture of the Bayesian machine with non-conditionally independent observations. This architecture performs non-naive Bayesian inference following eq. 1, by pooling observations O_2 and O_3 into the same column.

Supplementary Note 2: Programming methodology of the Bayesian machine

This note details the programing methodology of the memristor arrays in the fabricated Bayesian machine. Memristors are programmed with voltages higher than the nominal voltage used for digital circuitry. The higher-than-nominal programming voltage requirement of memristors is a minor concern in systems that separate memory from computing, as a single dedicated high-voltage circuitry can be associated with the memory array. The Bayesian machine, by contrast, features multiple small memory arrays fully embedded within logic. The programming of memristors, therefore, requires the distribution of higher-than-nominal programming voltages locally and an appropriate programming strategy. Managing this complexity is the largest design challenge for obtaining a functional Bayesian machine.

The Bayesian machine stores bits using a 2T2R structure (see Suppl. Fig. 2a): the bit cell is composed of a “bit line” and a “bit line bar” memristor (R and R_b), positioned on the same row, and each associated with a selection nMOSFET. The two memristors are connected to two different bit lines (BL and BLb) on their bottom electrode side. Conversely, the top electrode of the two memristors is connected to the same source line, to limit wiring and programming circuitry. This shared source line requires careful attention when programming the memristors.

In addition to the digital power supply (VDD), two higher-than-nominal supply voltages (VDDR and VDDC) are distributed to each memory array. Programming operation is controlled by nominal-voltage signals (CBL, CSL, and CWL) and an address, all provided by the digital control unit of the Bayesian machine. Decoders select the addressed row and column. Then, local level shifters apply either ground or higher-than-nominal voltages to the crossbar arrays, where needed:

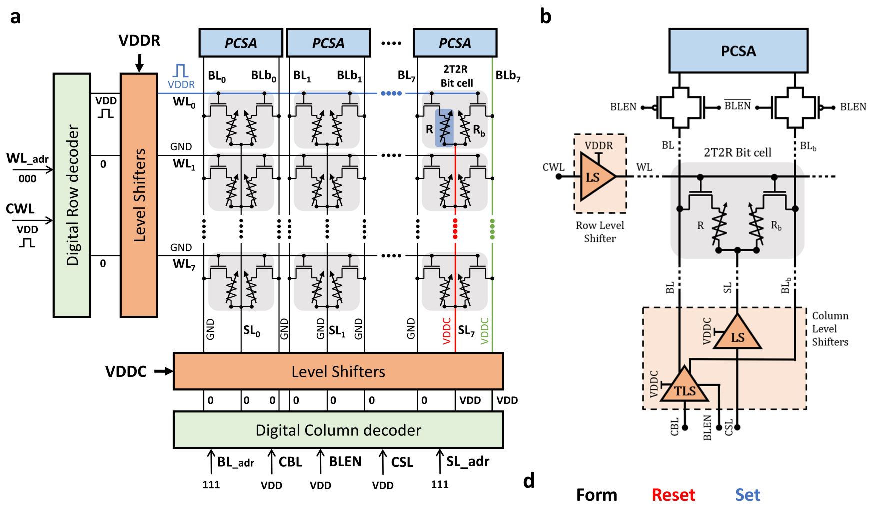
- Each memory row features one level shifter (LS, commanded by CWL), which controls the word line that feeds the gates of nMOS selection transistors of the memory rows. Depending on the value of CWL, the level shifter of the addressed row connects its world line either to ground or to the VDDR power supply (see Suppl. Figs. 2b and e).
- Correspondingly each memory column features two level shifters (regular LS, commanded by CSL and tri-state TLS, commanded by CBL), which control the source lines and the bit lines. These level shifters connect the source line and the two bit lines either to the ground or to the VDDC power supply. Notice that the two-bit lines BL and BLb are connected to the same level shifter, and therefore always receive complementary voltage (see Suppl. Figs. 2b and f).

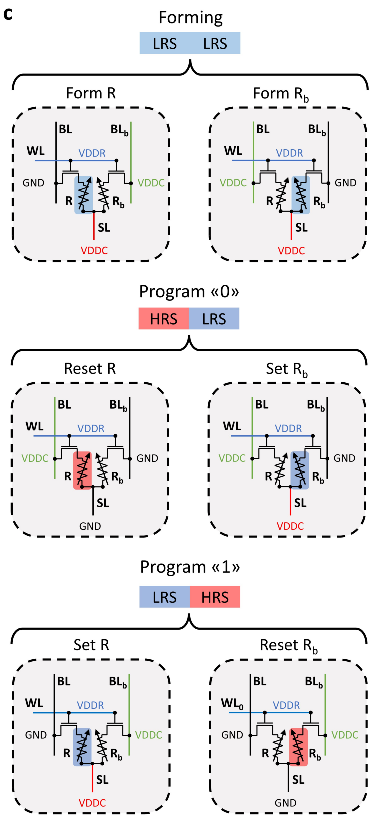
When the system is first characterized, the memristors need to be formed. Suppl. Fig. 2c shows the voltages that need to be applied on the memristors during the forming step. The Table in Suppl. Fig. 2g lists the associated CSL, and CBL values.

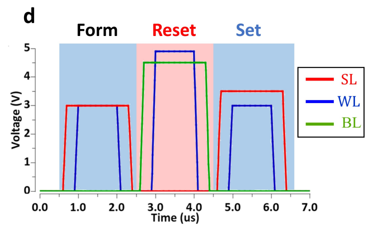
Note that when one memristor is being formed, the other memristor is unaffected, due to the fact that the bit line and bit line bar always see complementary voltages (as they are connected to the complementary outputs of the BL level shifter). Once the proper address, CSL, and CBL values have been set, CWL is raised to one, and forming occurs during the CWL pulse (of one-microsecond duration, see Suppl. Fig. 2d).

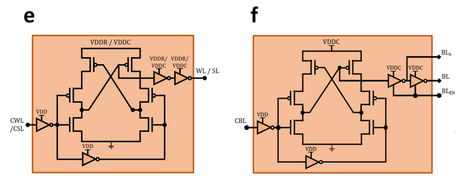
Once the memristors are formed, they are all in a low resistance state. The memristors can then be programmed and reprogrammed at will, either in low resistance state (LRS) by a SET operation, or to a high resistance state (HRS) by a RESET operation. In our 2T2R bit cells, the two memristors are always programmed in a complementary fashion: this complementary technique allows high robustness to memristor variability (see Supplementary Note 8) and read disturb effects (see Supplementary Note 3).

To program a zero value in the bit cell, the bit line memristor is programmed to high resistance (RESET) and the bit line bar memristor to low resistance (SET). The opposite is done to program a one value. Suppl. Fig. 2c shows the required voltages to program one and zero values, and the Table in Suppl. Fig. 2g lists the corresponding CSL and CBL value. The voltage levels of VDDR and VDDC used for the SET and RESET operations are given in the Methods section of the main article and reminded in Suppl. Fig. 2d.







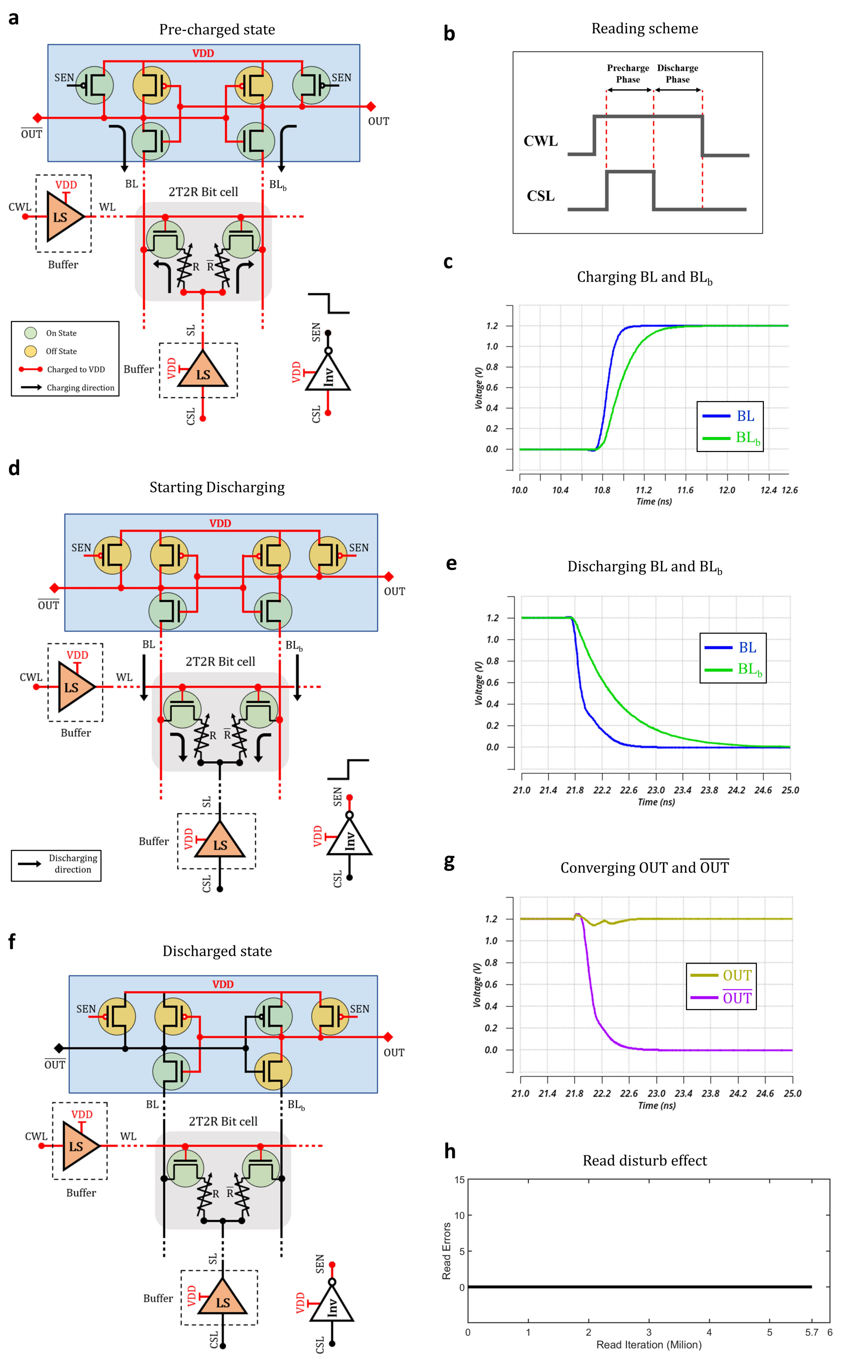


g

	Program «0»		Program «1»		Forming	
Device	R	R_b	R	R_b	R	R_b
Operation	Reset	Set	Set	Reset	Form	Form
State	HRS	LRS	LRS	HRS	LRS	LRS
CBL	VDD	VDD	GND	GND	GND	VDD
CSL	GND	VDD	VDD	GND	VDD	VDD
CWL	Pulse	Pulse	Pulse	Pulse	Pulse	Pulse
BL	VDDC	VDDC	GND	GND	GND	VDDC
BL_b	GND	GND	VDDC	VDDC	VDDC	GND
SL	GND	VDDC	VDDC	GND	VDDC	VDDC

Supplementary Figure 2. Programming circuitry for the likelihood memory arrays. a Detailed schematics of the likelihood memory array, with its programming and reading periphery circuitry, displaying the voltages needed to perform a SET operation on the first row, last column left memristor R. **b** Schematics of the 2T2R bit cell connections to the reading and programming circuitry. Two level shifters (conventional level shifter LS and three-state level shifter TLS) and one sense amplifier (PCSA) are implemented in each column. One level shifter is implemented in each row. The digital signal BLEN allows choosing between the reading or programming mode. **c** Voltages that need to be applied on the bit line BL, bit line bar BLb, and source line SL for forming, programming a zero, and programming a one in a 2T2R Bit cell. **d** Programming voltage levels and timings used for the Forming, RESET, and SET operations (mentioned in the Methods section of the main article). **e** Transistor-level schematics of the level shifter (LS) and **f** the three-state level shifter (TLS) circuits. **g** Table summarizing the configuration of programming signals (level shifters) for the different programming operations supported by the memory array (forming, programming a zero, and programming a one).

Supplementary Note 3: Reading strategy and impact of read disturb on the Bayesian machine



Supplementary Figure 3. Read circuitry for the likelihood memory arrays. This Figure presents circuit schematics (**a**, **d**, and **f**) and simulations (**c**, **e**, **g**, and **h**) that explain the reading operation of the memory array. **b** The read scheme involves a precharge and a discharge phase. **a** Schematic and **c** circuit simulation of the precharge phase. BL and BLb are charged to VDD. **d** Schematic and **e** circuit simulation of the discharge phase, BL and BLb are discharged to GND with different speed. **f** Schematics and **g** circuit simulation of the outputs of the PCSA converging to a stable state, while BL and BLb are completely discharged to GND. **h** Experimental measurement of the read disturb on a likelihood memory arrays, with a VDD value of 1.2 volts. Even after 5.7M read operations of the whole array, no error is seen.

This note explains the reading operation of the memristor states in the fabricated Bayesian machine. During the read mode, the bit lines are disconnected from the programming level shifter circuit (using the tri-state level shifter TLS, see Supplementary Fig. 2b) and connected to the precharge sense amplifier (PCSA) circuit. The source line and word line level shifters then act as buffers supplied by nominal voltage VDD (see Suppl Fig. 3a). The read operation functions in two phases: precharge and discharge (see Suppl Fig. 3b).

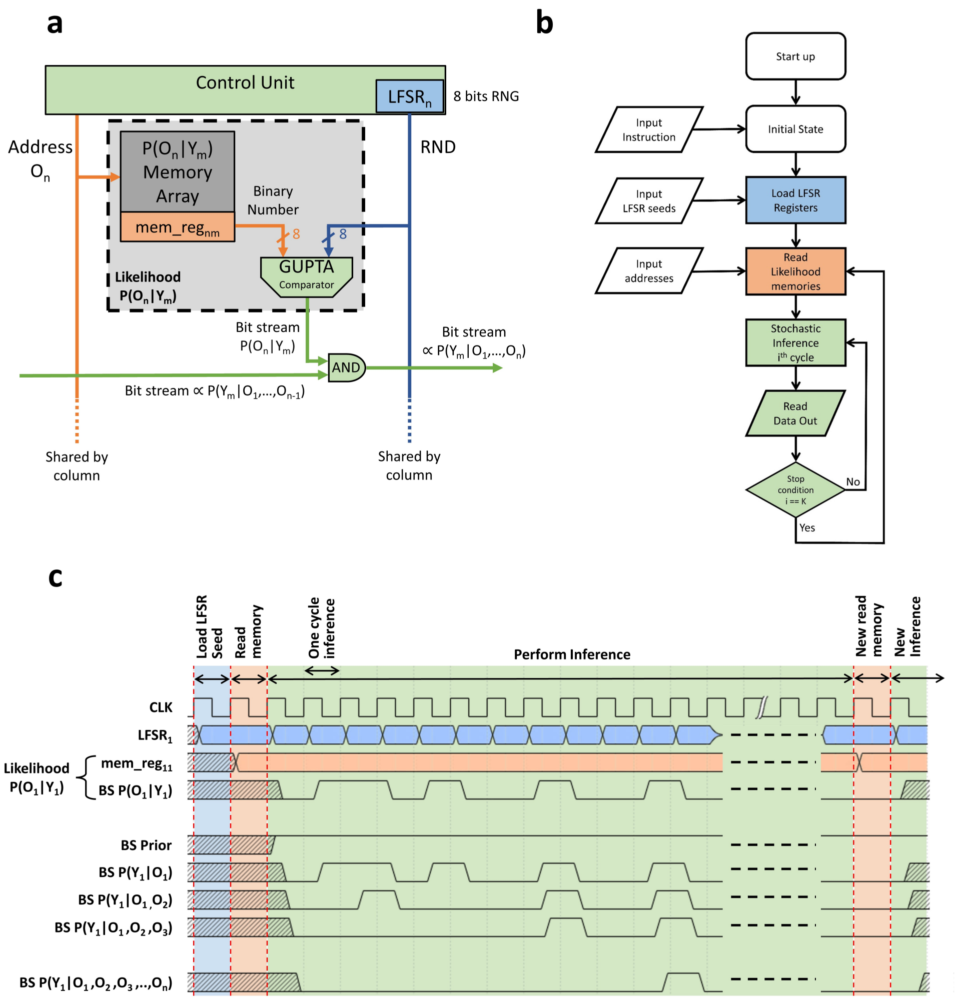
The PCSA is an energy-efficient sense amplifier that operates without any direct current path between the ground and the power supply, thanks to the initial precharge phase. The details of the operation of this circuit are reported in[1,2], and we summarize them here. The precharge pulls up all voltages of the sense amplifier and the bit lines to the digital power supply VDD (Supplementary Figs. 3a and c). In the actual read operation, the two bit lines discharge (Supplementary Figs. 3d,e). The bit line with the lowest-resistance discharges faster unit its associated inverter output discharges to ground, which latches the complementary inverter output to the supply voltage (Supplementary Fig. 3f,g). The two output voltages, therefore, represent the comparison of the two complementary resistance values, which gives the bit stored in the 2T2R bit cell.

The typical operation of a Bayesian machine requires frequent read operation on the likelihood arrays, with rare reprogramming: the memristors need to be reprogrammed only when the model is changed (e.g., after a recalibration based on new training data). It is therefore critical that read operations cannot accidentally change the state of the memory devices (read disturb effect). To evaluate the existence of read disturb, we performed repeated read operations on one likelihood memory block of the fabricated memristor machine. The read operations are performed by the on-chip precharge sense amplifiers. We used a digital power supply voltage of 1.2 volts, as it is the highest digital supply voltage supported by our system (see Figure 3 of the main article), and the more at-risk of read disturb effects. Our experimental setup allows reading the likelihood array approximately one million times per day. We observed that after five days of continuous read operations, i.e. a total of 5.7 million read operations of a complete likelihood memory array, no likelihood bit had changed (see Supplementary Fig. 3h), suggesting a high immunity to read disturb effects.

The immunity to read disturb of our machine can be explained by three main reasons:

- Hafnium oxide memristors are naturally resilient to read disturb effects due to the highly nonlinear nature of their switching process[3].
- The complementary 2T2R approach not only reduces the impact of device variability (see Supplementary. Note 8), but also read disturb effects. Even if the read disturb effect increased the resistance of a low resistance state device, the stored likelihood would be affected only if the disturbed device ends up in a resistance higher than its complementary high resistance device (see Supplementary Note 8).
- The precharge sense amplifier used to read the devices naturally mitigates read disturb effects. When the sense amplifier has identified the stored bit, the nodes connecting the sense amplifier to the memristor array are rapidly pulled down to the ground, and the read memristors therefore see zero voltage. Therefore, current is applied to the memristors only during the time needed for the sense amplifier to differentiate between the two possible memory states. This mode of operation contrasts with conventional current-mode sense amplifiers where current is applied during a fixed time that has to be chosen in a worst-case scenario[4].

Supplementary Note 4: Architectural details of the Bayesian machine



Supplementary Figure 4. Detailed operation of the Bayesian machine. a Schematic illustrating the detailed architecture of a likelihood elementary block. **b** Flowchart of the different operations to perform a Bayesian inference computation in the Bayesian machine. **c** Time diagram illustrating the operation of the Bayesian machine.

Operation of the Bayesian machine

This note details the operation of the Bayesian machine once it has been programmed. To understand this operation, Suppl. Fig. 4a shows a more detailed schematic of the likelihood elementary block (at row m and column n) than presented in the main paper. The function of this block is to output a bit stream with a probability proportional to the likelihood $P(O_n|Y_m)$. In addition, Suppl. Fig. 4b shows a flow chart of the stochastic Bayesian inference in the Bayesian machine, and Suppl. Fig. 4c shows a time diagram of the machine operation The color code throughout Suppl. Fig. 4 is as follows: the blue color corresponds to random number generation, the orange color to memory read, and the green color to the actual stochastic inference.

Before starting Bayesian inference operations, we first need to load the LFSR seeds. In our design, the input seeds are loaded from input pads and routed to the four LFSRs of the Bayesian machine by the Bayesian machine digital control unit. Being able to choose the LFSR seed is important for our study (Figure. 3, main article). In a final design, optimal LFSR seed values could be loaded automatically by the control unit (see Results section of the main article). As the LFSRs have a periodical output, the seed initialization needs to be performed only once as long as the digital power supply VDD remains on.

The Bayesian machine performs inference in two main phases:

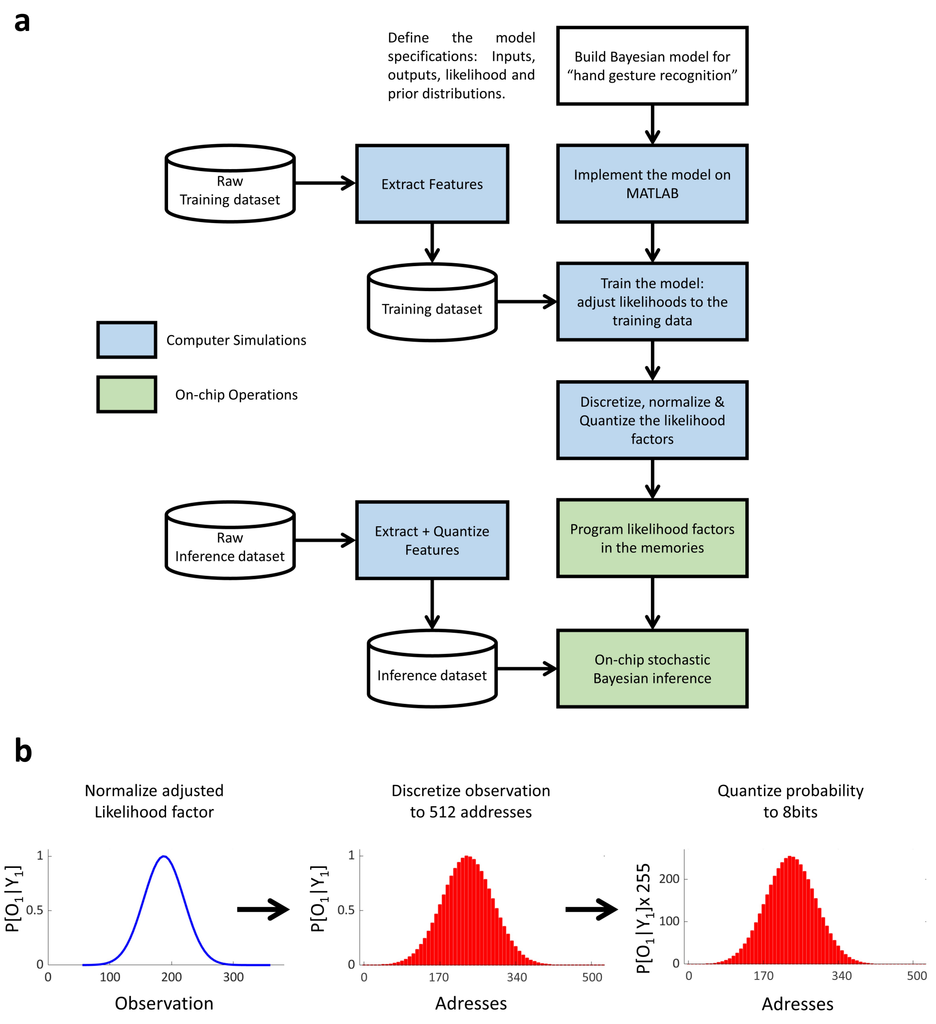
- **Memory read.** Likelihood values are read from the memristor arrays, based on the input observations (acting as row addresses). Observations $O_1, ..., O_n$ are off-chip inputs loaded from dedicated input pads, then addressed to the likelihood memory arrays by the Bayesian machine digital control unit (one observation for each column, see Figs. 1 and 2 of the main paper). All likelihood memory arrays are read simultaneously.
- **Iterative stochastic inference phase.** At each clock cycle, LFSRs generate new eight-bits pseudo-random numbers. These numbers feed the Gupta comparator circuits to compute the stochastic bits based on the read likelihood values. Outputs of Gupta comparator circuits from the same row are fed to a chain of AND gates, to perform the stochastic multiplications; the results of those multiplications represent the Bayesian machine outputs. All these operations are performed by purely combinational circuits in one clock cycle. As the periodicity of the LFSRs is 255 cycles, the maximum number of iterations is 255 cycles.

The Bayesian machine can operate in two modes. In the conventional stochastic inference mode, computation is performed for a pre-chosen number of cycles. The machine decision is based on bit streams counting of outputs (the number of generated one values). In the“power-conscious mode”, the Bayesian machine stops the stochastic inference iterations as soon as one of the outputs produces a bit value of one; the decision is made based on that result (see main paper, Results section).

Steps of a project involving the Bayesian machine

As a summary, Suppl. Fig. 5 also recapitulates the different steps of a project using the Bayesian machine, from training to on-chip operation. (The technical details corresponding to these different steps are reported in the Methods section of the main article):

- **Collect training data.** Any project starts by collecting training data.
- **Implement the likelihood model.** The likelihoods of the Bayesian models are computed. In our sample gesture recognition task, likelihoods are modeled by fitting Gaussian laws on the training data (see Methods of the main paper and Supplementary Note 11). In other situations, likelihood models may also be obtained based on expert knowledge or prior information[5].
- **Normalize and quantize likelihoods.** For improving the efficiency of stochastic computing, likelihoods are normalized per column by the maximum likelihood value of the column (see main paper). Likelihoods are then discretized as eight-bit integers, and models are quantized to the number of observation values supported by the Bayesian machine.
- **Program the Bayesian machine.** Likelihood values are programmed to the memristors of the Bayesian machine following the methodology described in Supplementary Note 2.
- **Use the Bayesian machine to do inference.** The Bayesian machine can finally be used to infer variables based on observations, using the methodology described in the first part of this Supplementary Note.


a
Define the model specifications: Inputs, outputs, likelihood and prior distributions.
Build Bayesian model for "hand gesture recognition"
Raw Training dataset
Extract Features
Implement the model on MATLAB
Training dataset
Train the model: adjust likelihoods to the training data
Computer Simulations
On-chip Operations
Discretize, normalize & Quantize the likelihood factors
Raw Inference dataset
Extract + Quantize Features
Program likelihood factors in the memories
Inference dataset
On-chip stochastic Bayesian inference
b
Normalize adjusted Likelihood factor
Discretize observation to 512 addresses
Quantize probability to 8bits
P[O1|Y1]
Observation
Adresses
P[O1|Y1]x 255

Supplementary Figure 5. **The different steps of a project with the Bayesian machine, from training to on-chip inference.** **a** Diagram summarizing the main steps of a project involving the Bayesian machine, from Bayesian model building to using the Bayesian machine to perform on-chip inference. **b** Illustration of the different steps of normalization, discretization, and quantization of the likelihood factors, necessary before programming the likelihood memory arrays on chip.

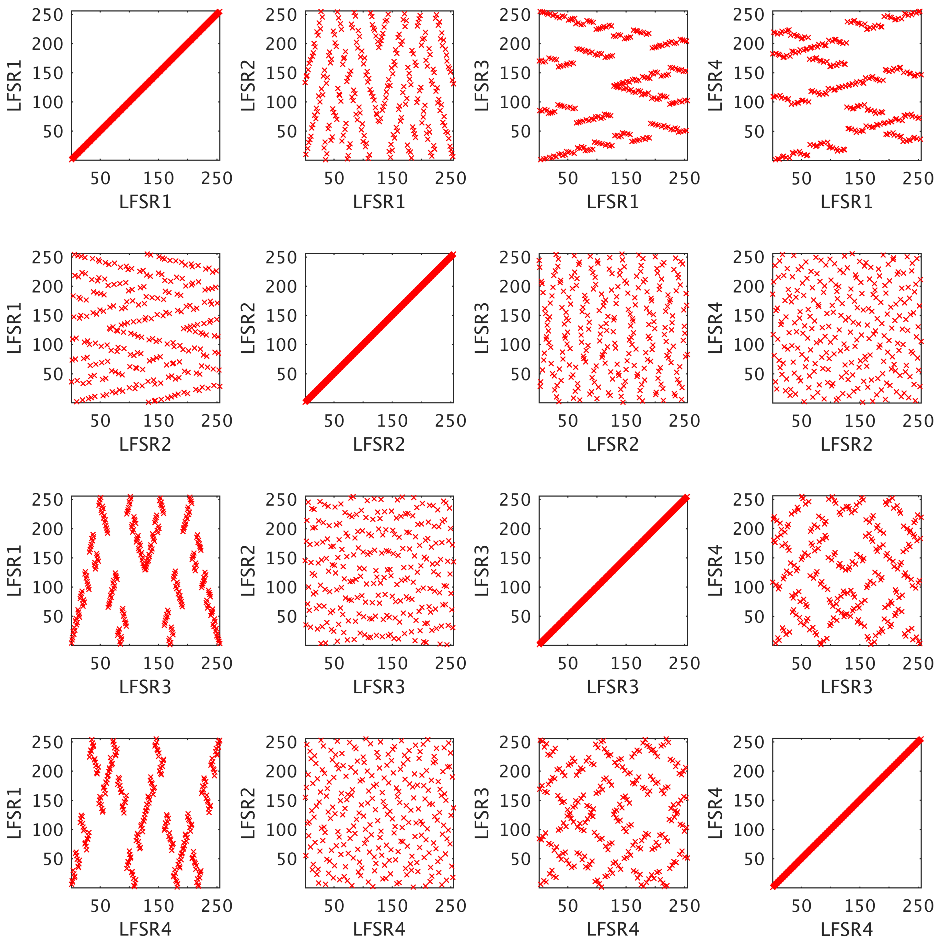
Supplementary Note 5: Optimal choice of the LFSR seeds of the Bayesian machine

In its original form, stochastic computing relies on high-quality and non-correlated random numbers. Generating high-quality random bits and maintaining the independence of stochastic bits after they have been processed by stochastic computing circuits is a major challenge, leading to costly strategies like randomness isolation and regeneration[6]. However, in practice, multiple works have shown that pseudorandomness and correlations are not necessarily a fundamental issue for stochastic computing, if they are properly considered in the system design[7]. In the particular case of our Bayesian machine, we found that we could rely on low-quality, low-cost LFSR-generated random numbers, and still get accurate Bayesian inference, under the condition of initializing the LFSRs with well-chosen seeds.

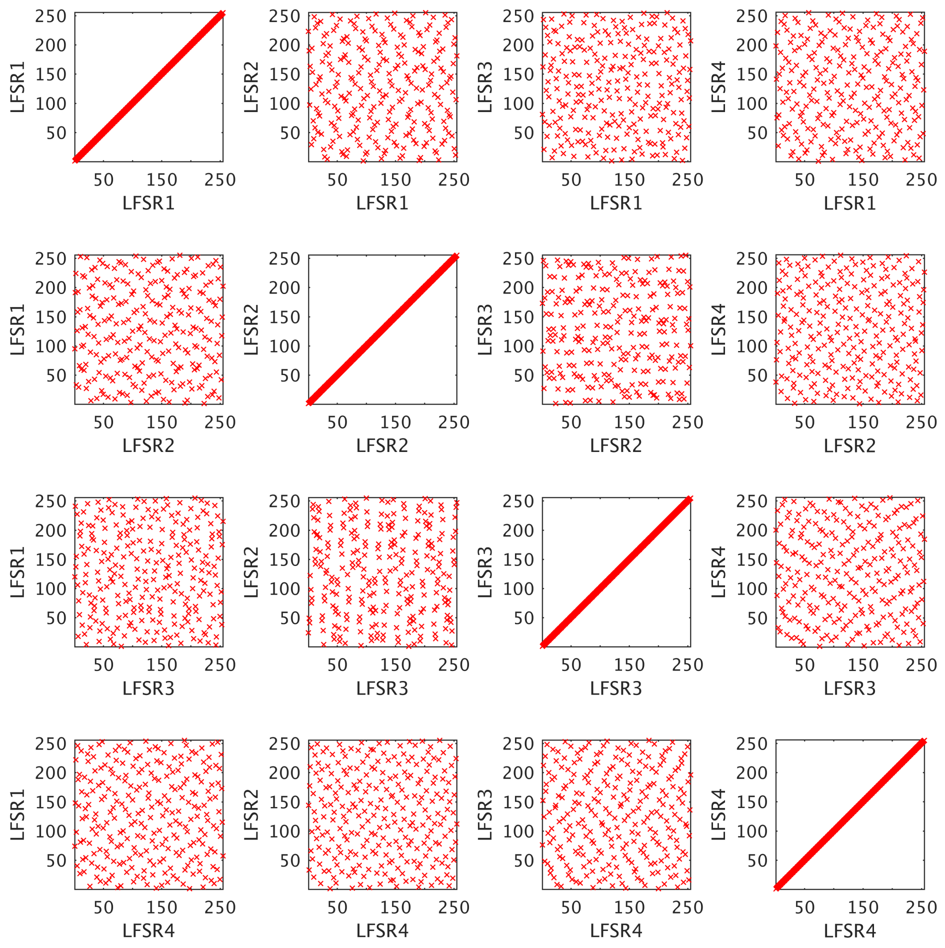
Our design exploits one eight-bit LFSR per column, which generates pseudorandom numbers with a periodicity of only 255 clock cycles. Supplementary Fig. 6 highlights the correlation between the random numbers generated by the LFSRs, by plotting the 255 values generated by each LFSR as a function of the 255 values generated by each other LFSR at the same time. This Figure is plotted with initially random-chosen LFSR seeds, as in Fig. 3c of the main article. Obvious correlations are observed between some LFSR. These correlations cause some inputs to the stochastic computing AND gates of the Bayesian machine to be correlated, leading to the deviations between Bayes' law and measurements in Fig. 3c.

This type of correlation is observed with most choices of LFSR seeds. However, the choice of seeds used to generate Supplementary Fig. 7, referred to as "optimal" in the main article, shows dramatically reduced correlations. This allows obtaining the highly accurate results of Fig. 3d, and suggests that these seeds should be used for all computations. Due to the periodicity of the LFSR, LFSR initialization needs to be performed only once, when the system is turned on.

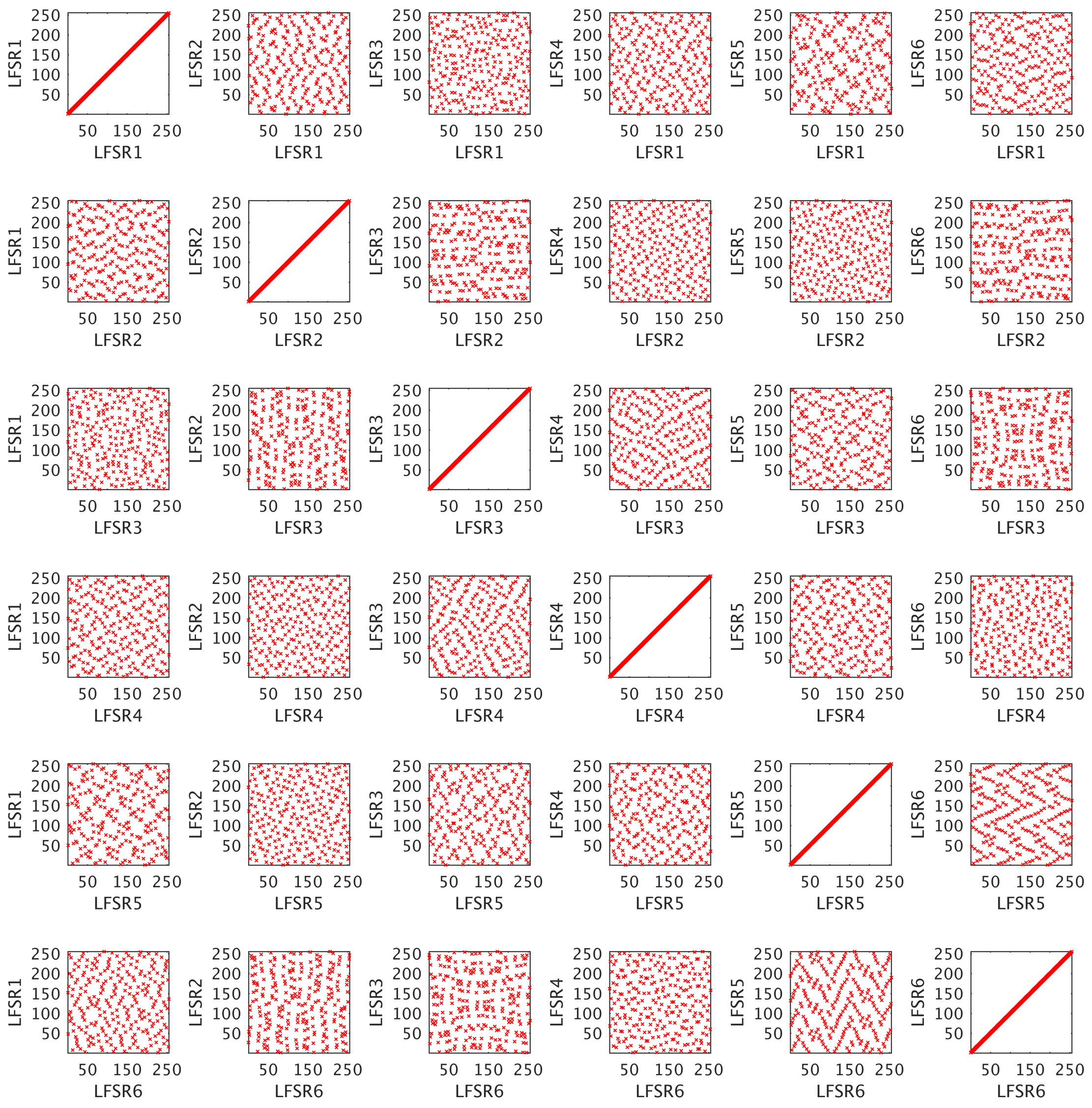
The gesture recognition task of Fig. 4 of the main paper requires six LFSRs. We also used eight-bit LFSRs and optimized the seed choice for this situation. Fig. 8 shows the correlations between the LFSR outputs in the optimized choice used for all the results of Fig. 4.



Supplementary Figure 6. Correlations of the random numbers generated by four LFSRs, with initially randomly chosen seeds used (as in Fig. 3c). Each graph presents the output of one of the four LFSRs of the Bayesian machine, as a function of the output of another LFSR. Each graph contains 255 points corresponding to the 255 cycles of operation of the Bayesian machine. Graphs on the diagonal (LFSR1/LFSR1, LFSR2/LFSR2, LFSR3/LFSR3, LFSR4/LFSR4) appear as x=y lines, by definition. The presence of very discernible patterns in some of the graphs (LFSR1/LFSR3, LFSR1/LFSR4), indicates the existence of a strong correlation between the output of some LFSRs. On the other hand, the outputs of some LFSRs appear largely uncorrelated (LFSR1/LFSR2, LFSR2/LFSR3, LFSR2/LFSR4). The seeds for the four LFSRs are, respectively, in hexadecimal representation: 50, E9, 10, and C6.



Supplementary Figure 7. Correlations of the random numbers generated by four LFSRs, with the optimal seeds used in Fig. 3d. This Figure, plotted with the same conventions as Supplementary Fig. 6, shows the absence of evident correlation between the outputs of the different LFSRs. The seeds for the four LFSRs are, respectively, in hexadecimal representation: EB, FB, 7F, and 5C.

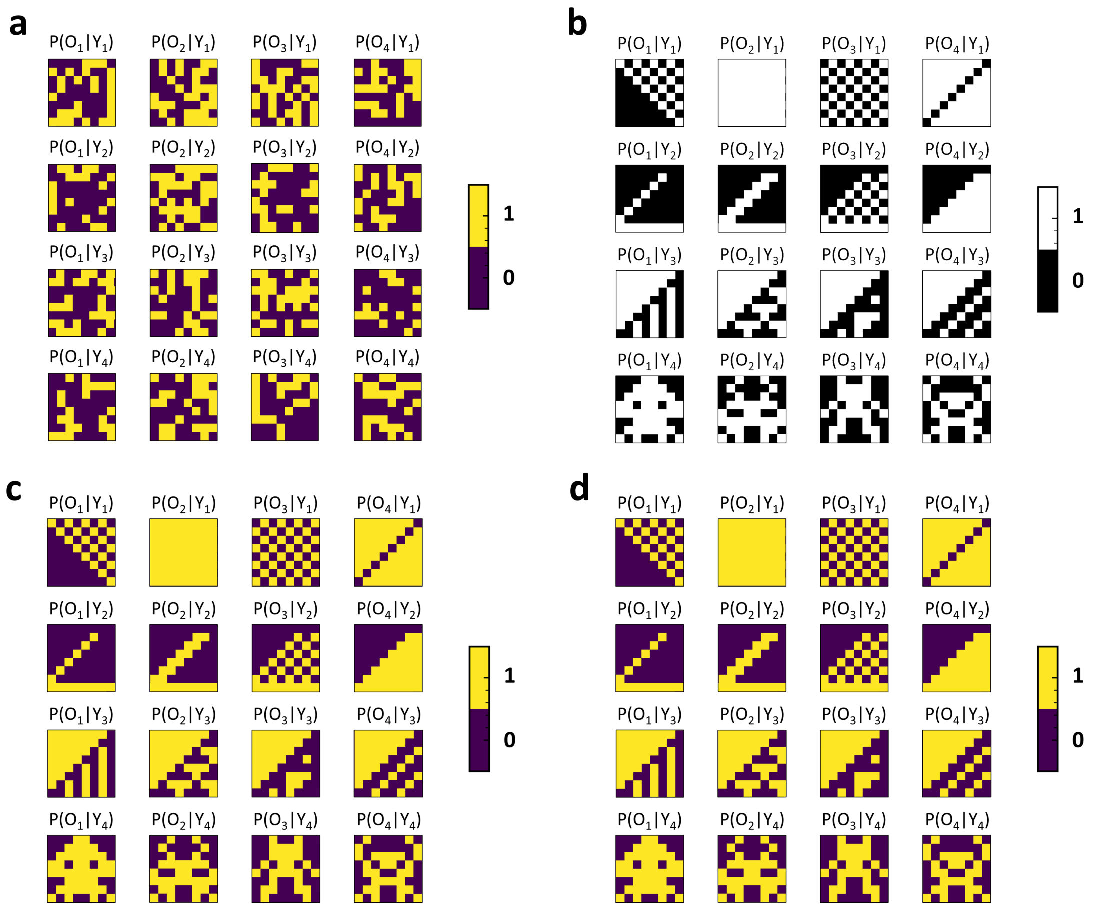


Supplementary Figure 8. Correlations of the random numbers generated by six LFSRs, with the optimal seeds used for all gesture recognition results of the main paper. This Figure, plotted with the same conventions as Supplementary Fig. 6, shows the absence of evident correlation between the outputs of the different LFSRs. The seeds for the four LFSRs are, respectively, in hexadecimal representation: EB, FB, 7F, 5C, F3, and D6.

Supplementary Note 6: Measurements of the likelihoods stored in the demonstrator

For our experiments, we programmed the patterns presented in Suppl. Fig. 9a in the likelihood memory arrays. These artificial patterns were constructed so that performing Bayesian with random inputs allows exploring the whole range of possible output probabilities (see Figs. 3c,d of the main article), allowing to test the functionality of the demonstrator.

Suppl. Fig. 9b shows the likelihoods actually programmed in the demonstrator, as measured right after the programming operation (no program-and-verify operation was performed). No error is seen with regards to the intended pattern, highlighting the performance of our complementary approach to program the memristors. Suppl. Fig. 9c shows measurements of the stored likelihoods five months after programming (the demonstrator was stored at room temperature during these five months). No error is seen.



Supplementary Figure 9. Measurements of the likelihoods stored in the demonstrator. **a** Patterns measured before forming. The results of the measurement appear random. **b** Patterns intended to be stored. **c** Patterns measured immediately after programming. **d** Patterns measured after five months, during which the demonstrator was stored at room temperature. **a**, **c**, **d** are presented in colors to express that they represent on-chip measurements, **b** is presented in black-and-white to express that it represents an intended pattern.

Supplementary Note 7: Features used for the gesture recognition task

Gesture recognition is performed based on ten features (F_0 to F_9) extracted from recordings of the three-axis accelerator of an inertial measurement unit (see Methods of the main method). Before feature extraction, we subtracted the average values of the recording to eliminate gravity.

The first feature F_0 is the square root of the mean value of the squared acceleration

$$F_0 = \sqrt{\tfrac{1}{T}\textstyle\sum_t \left(\ddot{x}^2(t) + \ddot{y}^2(t) + \ddot{z}^2(t)\right)}. \tag{2}$$

Features F_1, F_2, and F_3 represent the maximum acceleration along each axis:

$$\begin{aligned} F_1 &= \max_t\{|\ddot{x}(t)|\}/F_0 \\ F_2 &= \max_t\{|\ddot{y}(t)|\}/F_0 \\ F_3 &= \max_t\{|\ddot{z}(t)|\}/F_0. \end{aligned} \tag{3}$$

F_4 and F_5 are calculated using the variance of the acceleration on the three axes:

$$\begin{aligned} \mathrm{Var}\,\ddot{x} &= \frac{1}{T}\sum_t \ddot{x}^2(t) \\ \mathrm{Var}\,\ddot{y} &= \frac{1}{T}\sum_t \ddot{y}^2(t) \\ \mathrm{Var}\,\ddot{z} &= \frac{1}{T}\sum_t \ddot{z}^2(t). \end{aligned} \tag{4}$$

We then order those three variances as "min" for the smaller one, "inter" for the middle one, and "max'" for the bigger one and calculate F_4, and F_5 as

$$\begin{aligned} F_4 &= \sqrt{\min\{\mathrm{Var}\,\ddot{x}, T\,\mathrm{Var}\,\ddot{y}, \mathrm{Var}\,\ddot{z}\}/\max\{\mathrm{Var}\,\ddot{x},\ \mathrm{Var}\ \ddot{y}, \mathrm{Var}\,\ddot{z}\}} \\ F_5 &= \sqrt{\ \text{inter}\ \{\mathrm{Var}\,\ddot{x}, \mathrm{Var}\,\ddot{y}, \mathrm{Var}\,\ddot{z}\}/\max\{\mathrm{Var}\,\ddot{x},\ \mathrm{Var}\ \ddot{y}, \mathrm{Var}\,\ddot{z}\}}. \end{aligned} \tag{5}$$

The last features are based on the derivatives series of the acceleration, called "jerk", calculated via finite difference and with a filtering:

$$\begin{aligned} \dddot{x}(t) &= \frac{\ddot{x}(t+1) - \ddot{x}(t-1)}{2} \\ \dddot{y}(t) &= \frac{\ddot{y}(t+1) - \ddot{y}(t-1)}{2} \\ \dddot{z}(t) &= \frac{\ddot{z}(t+1) - \ddot{z}(t-1)}{2} \\ jerk_k(t) &= 0.4 \times \dddot{k}(t) + 0.3 \times (\dddot{k}(t-1) + \dddot{k}(t+1)) \quad \text{with k = x; y; z} \end{aligned} \tag{6}$$

(we take as convention the values before and after the recording as equal to zero). Features F_6 to F_9 are then calculated similarly to F_0 to F_3, with the jerk replacing the acceleration:

$$\begin{aligned} F_6 &= \sqrt{\frac{1}{T}\sum_t \left(jerk_x^2(t) + jerk_y^2(t) + jerk_z^2(t)\right)/F_0} \\ F_7 &= \max_t\{|jerk_x(t)|\}\frac{F_0}{F_6} \\ F_8 &= \max_t\{|jerk_y(t)|\}\frac{F_0}{F_6} \\ F_9 &= \max_t\{|jerk_z(t)|\}\frac{F_0}{F_6}. \end{aligned} \tag{7}$$

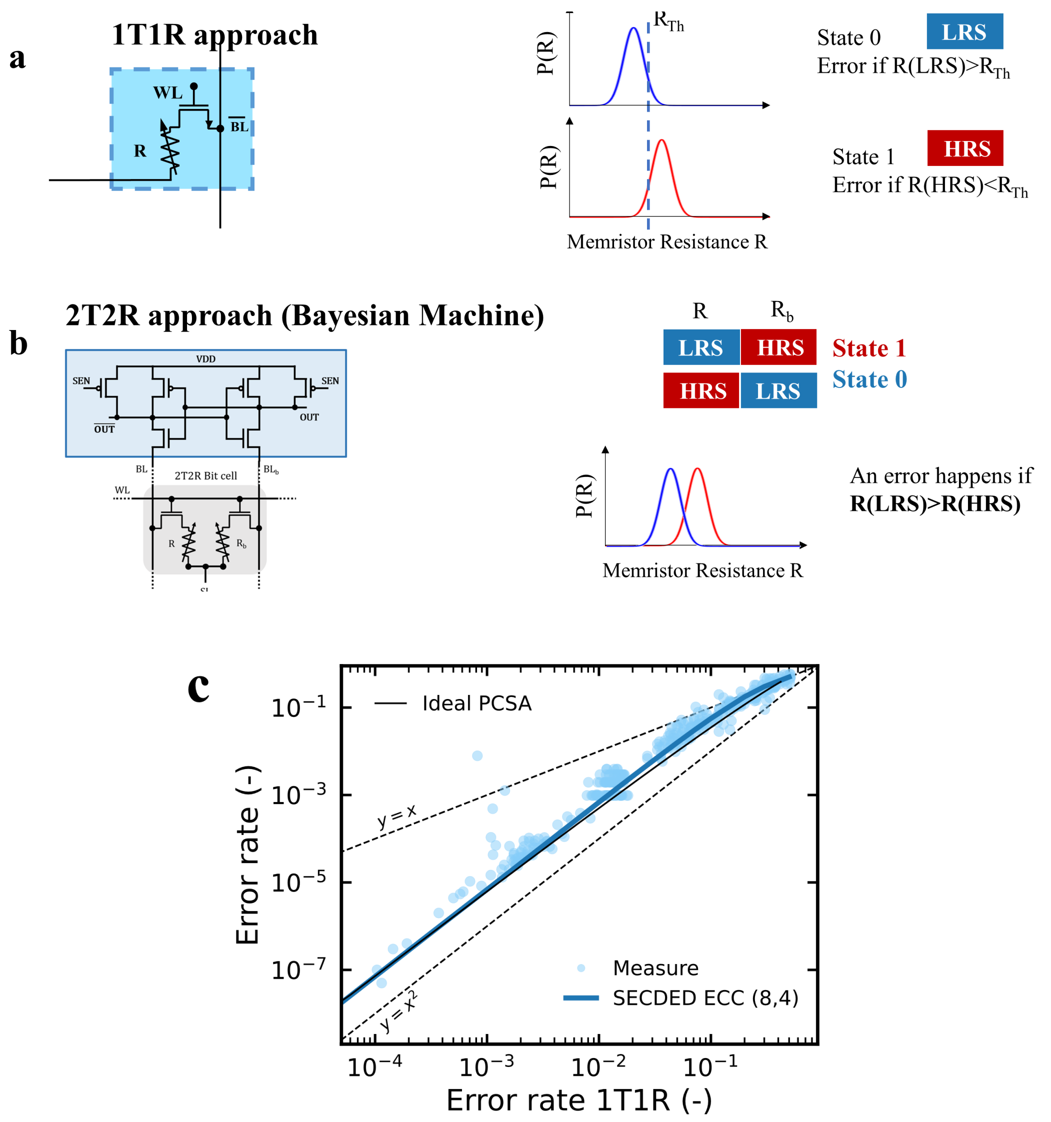
Supplementary Note 8: Impact of device variability on the Bayesian machine

Device variability is a strong limitation of memristor technology, particularly for in-memory and near-memory computing, where strong error-correction codes can not be used. Our design is using a simple strategy to reduce the impact of variability: the bit cell of the memory arrays is composed of two memristors (2T2R structure), and bits are stored in a complementary fashion (see Supplementary Notes 2 and 3, and Supplementary Figs. 2 and 3.

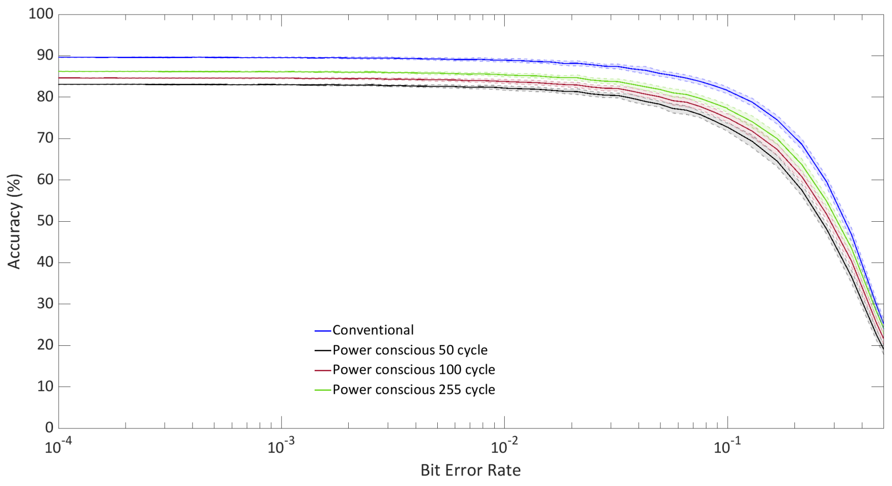
Supplementary Fig. 10a illustrates the impact of device variability in the conventional 1T1R approach. In this case, memristors in LRS mean state zero, and memristors in HRS mean state one. Therefore, a bit error happens whenever a device meant to be in LRS has a resistance higher than the threshold between LRS and HRS, or a device meant to be in HRS has a resistance lower than this threshold. In the 2T2R approach of the Bayesian machine (Supplementary Fig. 10b), conversely, two memristors are always programmed in a complementary fashion. The programmed state is read by comparing the resistance of the two memristors, using a precharge sense amplifier (see Supplementary Note 3). In that case, a bit error happens if a device programmed to LRS has a resistance higher than its complementary device programmed to HRS. Therefore, both devices have to be in a tail of distribution for an error to happen, making bit errors much less likely.

The efficiency of the 2T2R approach to reduce bit errors is confirmed in Supplementary Fig. 10c. This Figure plots the bit error rate of the 2T2R approach as a function of the one of the 1T1R approach, obtained experimentally and theoretically. The theoretical result assumes a perfect sense amplifier. The experimental results are reproduced from[8] and were obtained on a different integrated circuit based on the same technology than the Bayesian machine (the Bayesian machine is entirely dedicated to 2T2R and does not allow estimating the bit error rate of the 1T1R approach). We also plotted the error rate that would be obtained when using a conventional Single Error Correcting/Double Error Detecting correction code (SECDED, or extended Hamming) that doubles the number of memristors, as our approach. We see that our approach reduces the number of bit errors almost equivalently to the SECDED code. However, the SECDED code requires area, delay, and energy-costly error decoding circuits, while, in our approach, error correction happens naturally within the sense amplifier without any additional cost.

Thanks to the power of the 2T2R approach, we observed no bit error in the Bayesian machine when using the programming conditions reported in the Methods section of the main article and in Supplementary Note 2. Still, for fundamental reasons, it is important to understand the impact that bit errors in the memory would have on the operation of the machine. Suppl. Fig. 11 shows the results of simulations of the scaled-up Bayesian machine on the gesture recognition task, where bit errors have been artificially added into the likelihood memory arrays. Results are reported both using the conventional stochastic computing strategy with 255 cycles, and the power-conscious strategy, which stops computation as soon as one row provides a one output (see main text). The results of the power-conscious technique are shown for computations with varying maximum numbers of clock cycles. The accuracy on gesture recognition shows high resilience to these bit errors, both in the cases of conventional and power-conscious stochastic computing. Even for bit error rates as high as 10^{-2}, accuracy is reduced by only 0.7 percentage points with the conventional stochastic computing strategy. This result is similar to what has been observed in hardware neural networks[8] and seems surprising at first: bit errors can affect the most significant bits of stored values, and therefore change the stored likelihoods very significantly. The tolerance can be tied to two reasons. First, the stochastic nature of computing means that a majority of rows give zero value (see also Supplementary Note 10). Memory errors in such rows typically have no impact on the final result. Second, the different features used as inputs of the Bayesian machine carry partially redundant information (see Supplementary Note 7). Therefore, an error in a likelihood can be compensated by the likelihoods concerning the other inputs, visibly providing high resilience to the Bayesian machine.

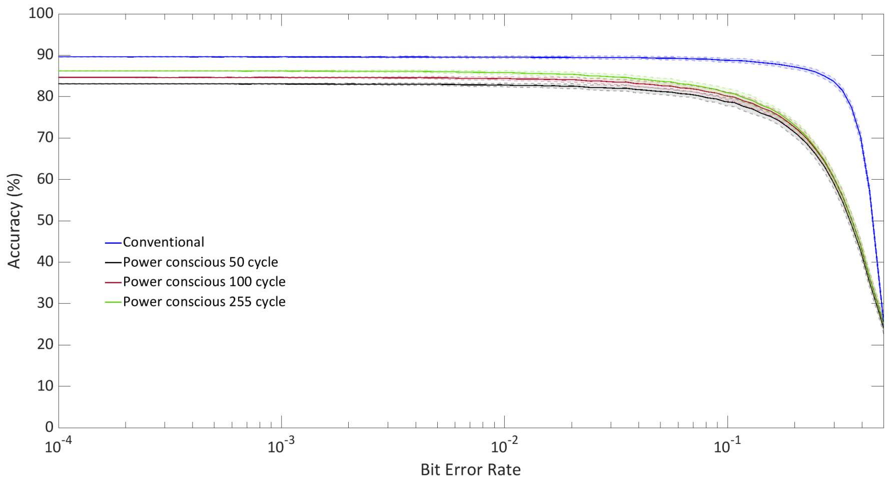


Supplementary Figure 10. Reduction of the effects of device variability with the 2T2R approach. **a** In the conventional approach, an error happens if a device programmed to LRS has a resistance higher than the threshold between LRS and HRS, or if a device programmed to HRS has a resistance lower than this threshold. **b** In the 2T2R approach of the Bayesian machine, an error happens if a device programmed to LRS has a resistance higher than its complementary device programmed to HRS: both devices have to be in a tail of distribution for an error to happen. **c** Error rate of the 2T2R approach as a function of the error rate of the 1T1R approach, in simulations assuming a perfect PCSA (black line) or experimentally measured on the integrated circuit of[8] (light blue points). Blue line: error rate of a SECDED ECC using the same number of devices as our 2T2R approach (see Supplementary Note 8).



Supplementary Figure 11. Impact of errors on memory bits on the accuracy of the task of gesture recognition. Simulated mean accuracy on the task of gesture recognition, when bit errors have been artificially introduced in the memory bits, as a function of memory bit error rate. The simulation was repeated 40 times, the shadows around the graph show the standard deviation of the mean accuracy along with these repetitions. The results are presented for the power-conscious mode with a cutoff at 50, 100, and 255 cycles, and for the conventional stochastic computing mode using all 255 cycles.

Supplementary Note 9: Impact of single-event upsets on the Bayesian machine



Supplementary Figure 12. Impact of single-event upsets on the accuracy on the task of gesture recognition. Simulated mean accuracy on the task of gesture recognition, when single-event upsets have been artificially introduced in the computation, as a function of memory bit error rate. The single-event upsets are introduced at the output of the Gupta circuits (i.e., on the stochastic bit streams created by each likelihood block). The simulation was repeated 100 times, the shadows around the graph show one standard deviation of the mean accuracy along these repetitions. The results are presented for the power-conscious mode with a cutoff at 50, 100, and 255 cycles, and for the conventional stochastic computing mode using all 255 cycles.

The Bayesian machine functions based on the principles of stochastic computing, meaning that the final decision is based on the statistics over several clock cycles. This mode of operation means that bit upsets have typically no catastrophic effects and get naturally averaged out. This feature provides the Bayesian machine with a natural immunity to single-event upsets, which may happen in a radiation-rich environment, especially when operating the machine at low supply voltage.

Suppl. Fig. 12 shows simulations of the gesture recognition task where single-event upsets have been artificially introduced at the output of the likelihood blocks. Simulations are presented using conventional stochastic computing operation with 255 cycles and the power-conscious approach with several maximum numbers of clock cycles. The accuracy shows outstanding immunity to single-event upsets, particularly in the conventional stochastic computing operation. Even with 10% bit errors, the final accuracy is barely affected. Immunity is largely improved with regards to the impact of memory bit errors (Suppl. Fig. 11), as, unlike in the case of memory, a bit error has an impact on a single cycle and may be averaged out by subsequent cycles. The power-conscious mode, which stops computation as soon as a one is produced at any output, is highly resilient to single-event upsets, but less than the conventional mode. This result is natural, as single-event upsets then can produce an incorrect one and stop computation on a wrong answer. In contexts with high levels of single-event upsets (extreme environments), conventional stochastic computing should therefore be preferred.

Supplementary Note 10: Explainability of the gesture recognition task

A very significant benefit of Bayesian approaches is their explainability. In the case of our Bayesian machine, just looking at the standard deviation of the likelihood associated with each input allows understanding which features are important to the decision (low standard deviation), and which ones are accessory (high standard deviation). Explainability is desirable in many critical situations for ethical and regulatory reasons[9]. Additionally, the fact that Bayesian inference takes decisions based on explainable models has very practical consequences, in particular when we use them with inputs that differ from those used for training the model. In this type of situation, neural networks, which function by fitting training data, tend to give sure and unpredictable answers. Due to this issue, neural networks easily give seemingly high-confidence clearly-wrong answers when presented with out-of-distribution data[10,11]. By contrast, Bayesian models excel at recognizing situations where they are not able to provide a reliable answer[12].

To illustrate this capability in the case of our Bayesian machine, we present results about the behavior of Bayesian machines trained on gesture recognition, when they are presented with data from another subject as the one that they were trained to recognize. We show that in a vast majority of these cases, the machine correctly identifies that it cannot provide a reliable answer. More specifically, there can be two signatures that the Bayesian machine cannot provide a confident answer:

- When some observations of the input are particularly unlikely to happen simultaneously in the training data, then all rows of the Bayesian machine may produce only, or almost only, zeros. This situation can arise, for example, in the event of a failing sensor, the output of which becomes incoherent with the other sensor readings.
- If the observations are not the signature of a particular situation, then all rows may produce a balanced number of ones. This situation corresponds to a uniform probability distribution, i.e., a situation where the entropy is maximal, and the model does not show a strong preference for any answer.

Based on these considerations, we can define a metric to determine if the Bayesian machine is confident about its output (Supplementary Fig. 13a illustrates on examples of how this processing is done):

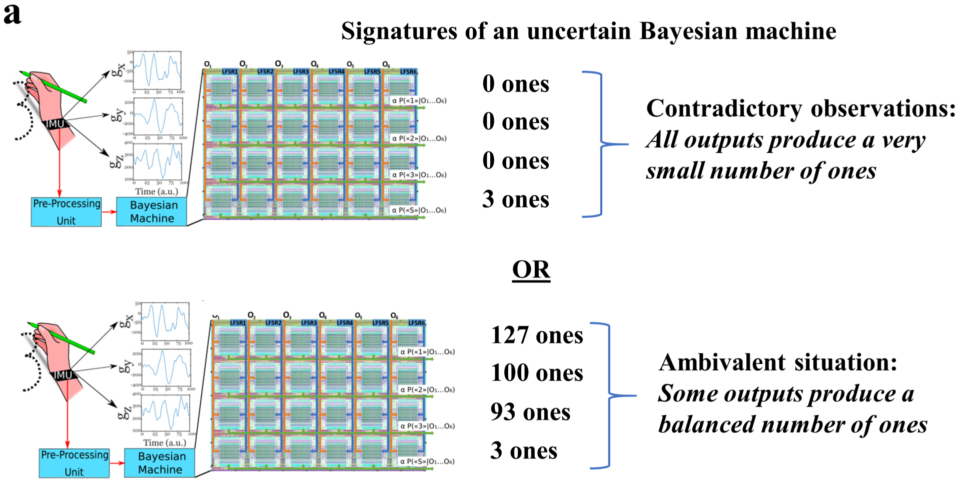
- Concerning the first criterion, we count the number of ones for the four rows. If the number of ones on all rows is smaller than a certain pre-chosen proportion T of the maximal number of cycles (here 255 cycles), which we call certainty threshold, we consider that the Bayesian machine is uncertain.
- Concerning the second criterion, we calculate the sum of ones for each row and we divide the sum on the row with the highest number of ones by the total sum on the four rows. The machine is considered to provide a certain result if this ratio is higher than 50%.

Supplementary Fig. 13b shows graphically the power of this evaluation of uncertainty. This Figure summarizes the results of multiple experiments. We have presented each gesture of our dataset to the Bayesian machines trained for each subject. The x-axis is separated into ten sections corresponding to the ten subjects, and for each subject, the four points are respectively the signature, the digits one, two and three. The y-axis corresponds to the ten Bayesian machines trained for each subject, and the four data points represent its four outputs (the digits one, two, three, and the signature). Each point in the colormap indicates the proportion of gestures corresponding to the x-axis that has been recognized confidently as the output of the y-axis. For this Figure, a certainty threshold of 10% was used.

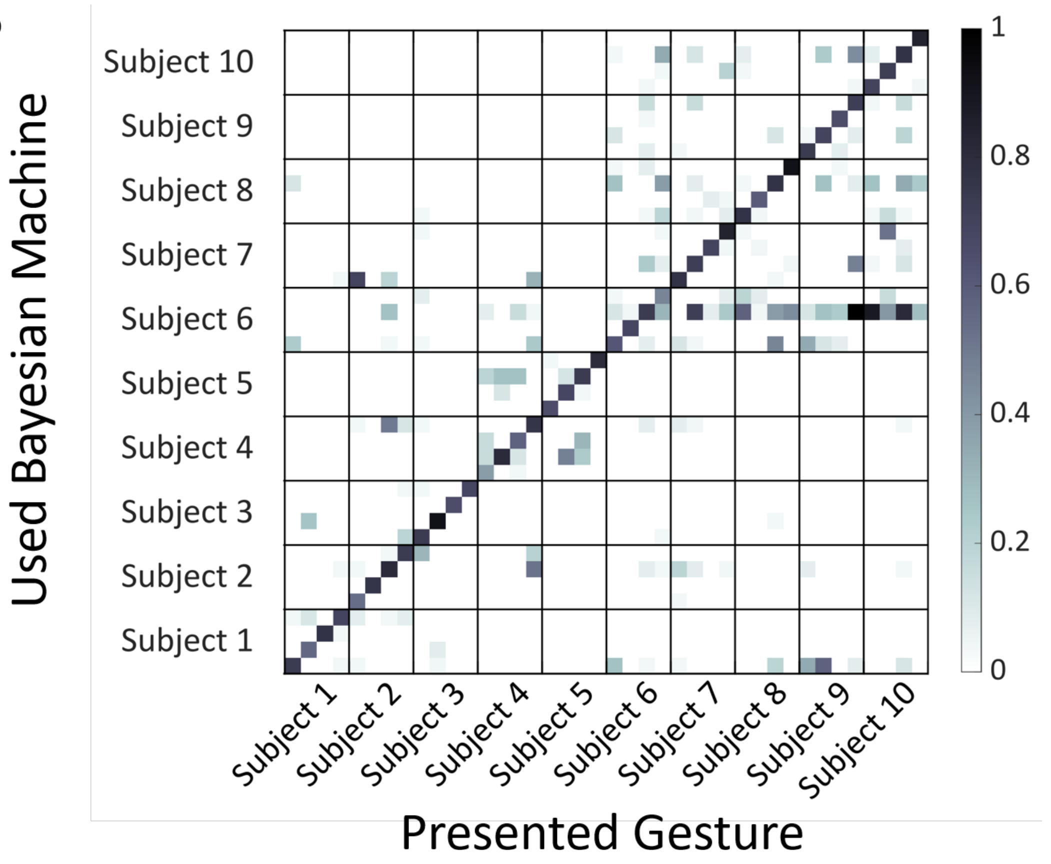
The desirable result is that when input and output subjects are mismatched, the colormap appears all-white (meaning the machine provided no certain results). This is indeed what is largely seen. An exception is the Bayesian machine for subject six, which tends to recognize a gesture for subjects seven, eight, and nine. Conversely, when input subject and Bayesian machine subject match, the desired result is a colormap all-black on the diagonals and white elsewhere. The results are very close to this ideal behavior, with a few gray points indicating the errors of the Bayesian machine.

Supplementary Table 1 details these results quantitatively. It also illustrates how the behavior of the Bayesian machine can be changed by adjusting the certainty threshold. When a gesture corresponding to the wrong subject is presented, the Bayesian correctly declares uncertainty 94% to 98% of the time, depending on the certainty threshold. When gestures from the correct subject are presented, the proportion of time where the machine makes a true mistake (i.e., it identifies the wrong gesture while being certain) is 5% for a certainty threshold of zero, and can be reduced to 3% with a certainty threshold of 10%. However, this comes at the prize or more gestures being flagged uncertain. These results illustrate how an operator can adjust the behavior of the Bayesian machine, by prioritizing the elimination of incorrect answers, or the minimization of uncertain answers. This degree of freedom can be particularly useful for medical tasks where unjustified intervention due to a wrong answer can have more cost than non-intervention based on an uncertain answer.

a


Signatures of an uncertain Bayesian machine
0 ones
0 ones
0 ones
3 ones
Contradictory observations:
All outputs produce a very small number of ones
OR
127 ones
100 ones
93 ones
3 ones
Ambivalent situation:
Some outputs produce a balanced number of ones
IMU
Pre-Processing Unit
Bayesian Machine
Time (a.u.)

b



Supplementary Figure 13. Capability of the Bayesian machine to identify uncertain situations. a Schematic illustration of the two signatures of an uncertain Bayesian machine (see Supplementary Note 10). **b** Simulated mean response of the Bayesian machine, for the gesture recognition task, in situations where a gesture presented to the Bayesian machine may be from a subject different than the one the machine has been trained for. X-axis: presented gesture. For each subject, the four points represent the gestures one, two, three, and the signature. Y-axis: used Bayesian machine. The different points represent the different outputs of the machine with the same order as the X-axis. Color: proportion of cases leading to the output in the Y-axis for gestures presented corresponding to the X-axis, with the Bayesian machine providing a certain output. The results are presented for a certainty threshold T of 10%.

Case when a gesture is presented to the Bayesian machine **corresponding to a different subject**

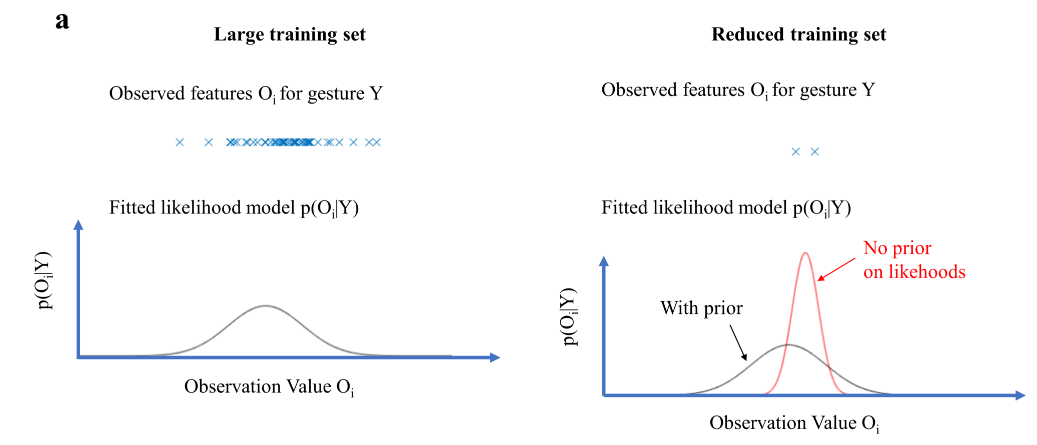
	T=0	T=5%	T=10%
Bayesian machine was uncertain (desired behavior)	93%	97%	98%
Bayesian machine provided a certain output	7%	3%	2%

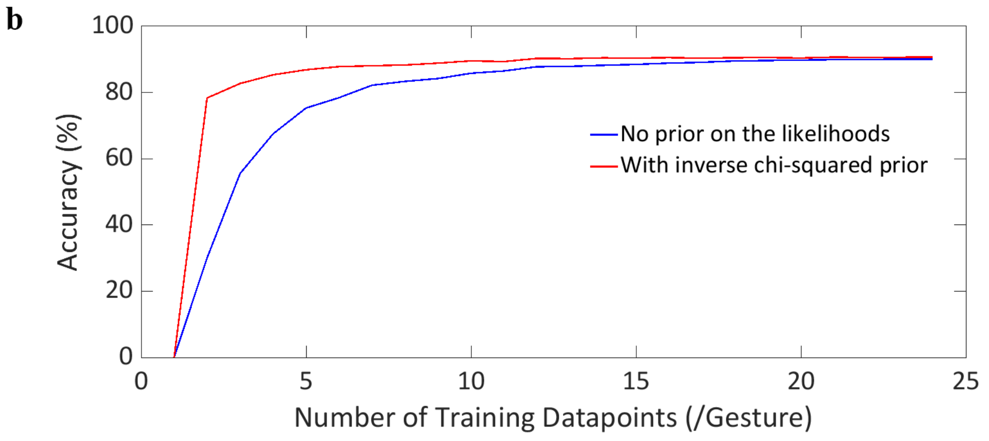
Case when a gesture is presented to the Bayesian machine **corresponding to the appropriate subject**

	T=0	T=5%	T=10%
Bayesian machine was certain about the correct gesture (desired behavior)	88%	80%	71%
Bayesian machine was certain about an incorrect gesture	7%	6%	5%
Bayesian machine was uncertain	5%	14%	24%

Supplementary Table 1. Quantitative behavior of the Bayesian machine when input gestures and recognized gestures may be subject-mismatched. The results are presented for three values of the certainty threshold T.

Supplementary Note 11: Training the gesture recognition task with reduced datasets


a
Large training set
Observed features O_i for gesture Y
Fitted likelihood model $p(O_i|Y)$
$p(O_i|Y)$
Observation Value O_i
Reduced training set
Observed features O_i for gesture Y
Fitted likelihood model $p(O_i|Y)$
No prior on likehoods
With prior
$p(O_i|Y)$
Observation Value O_i



Supplementary Figure 14. Impact of training dataset size on the accuracy of the Bayesian machine. a Illustration of the small-data challenge. When using a large representative training dataset, likelihood models can be fitted directly to the training data. Reduced datasets, on the other hand, may be non-representative, and directly fitting a likelihood model to the data can lead to meaningless results. Incorporating a prior when fitting the likelihood, however, can fix this issue. This possibility is an important benefit of Bayesian approaches. **b** Mean simulated accuracy of Bayesian machines as a function of the number of examples used in the training dataset. The accuracy is plotted in a situation where likelihood models are Gaussian fits of the training data, and when a normal inverse chi-square prior was used. All results are presented using the conventional stochastic computing method with 255 cycles. All results are also cross-validated, with 20 different separations between training and testing sets.

An important benefit of Bayesian approaches is that they adapt well to the quantity of training data. Fitting machine learning models when very little training data is available is highly challenging, as the training set is not representative. In this situation, machine learning models tend to “overfit” the few training examples that are given, and to give meaningless predictions when data that differ from the training examples is given. Fortunately, Bayesian models can deal particularly well with this type of situation, due to their explainability (see Supplementary Note 10), which allows incorporating prior assumptions into the fitting process.

For example, Suppl. Fig. 14a illustrates the challenge of training the Bayesian machine on the gesture recognition task in a very small-data situation. When enough training data is available, Gaussian likelihood models can be directly fitted to this data. When very little data is available, fitting a Gaussian to the data easily gives a meaningless result. A solution is to incorporate prior assumptions on what the mean value and standard deviation are expected to be. In the Bayesian statistics field, an established technique for doing this is to infer likelihood model parameters from the training data using a prior on the model. If we want to fit a Gaussian model with mean μ and variance σ^2 to N data points x_i, Bayesian inference suggests that the probability distribution on the model parameters given the values of the datapoint is

$$p(\mu,\sigma^2|x_1,...,x_N) \propto p(x_1,...,x_N|\mu,\sigma^2) \times p(\mu,\sigma^2) \propto \prod_{i=1}^{N} exp(-(x_i-\mu)^2/2\sigma^2) \times p(\mu,\sigma^2), \tag{8}$$

where $p(\mu,\sigma^2)$ expresses the prior belief on the values of μ and σ^2. A relatively established choice[13] for this prior is the normal inverse chi-squared (or NIX) distribution:

$$p(\mu,\sigma^2) \propto \left(\frac{1}{\sigma^2}\right)^{(\nu_0+3)/2} exp\left(-\frac{\nu_0\sigma_0^2+\kappa_0(\mu-m_0)^2}{2\sigma^2}\right). \tag{9}$$

In this prior, m_0 expresses the believed mean value and κ_0 the degree of confidence about it, σ_0^2 the believed value of the variance, and ν_0 the degree of confidence about it. The normal inverse chi-squared distribution is the conjugate prior for the normal distribution that allows the derivation of closed-form expressions: by combining equations (8) and (9), it can be shown (see[13], p. 136) that the expectation for μ is

$$\bar{\mu} = \frac{\kappa_0 m_0 + \sum_{i=1}^{N} x_i}{\kappa_0+N}. \tag{10}$$

The expectation for σ^2 is

$$\bar{\sigma^2} = \frac{1}{\kappa_0+N}\left(\nu_0\sigma_0^2 + \sum_{i=1}^{N}(x_i-\bar{x})^2 + \frac{N\kappa_0}{\kappa_0+N}(m_0-\bar{x})^2\right), \tag{11}$$

where $\bar{x} = (\sum_{i=1}^{N} x_i)/N$. Equations (10) and (11) can be used to train likelihood models in small data situation. For our work, we normalize each feature with zero being the lowest value in the small-data training dataset and one the highest value. We choose as prior $m_0 = 0.5$, $\sigma_0^2 = 0.25$, with a confidence $\kappa_0 = 1$, $\nu_0 = 1$.

Supplementary Fig. 14b shows the mean accuracy for the gesture recognition task, as a function of the number of examples of gesture that was used for training. The results are cross-validated, meaning that 20 different separations between the training and testing sets are performed for each situation. The results are presented when the likelihoods are just fitted to the data (as prescribed in the main paper), and when this operation is done with a prior using equations (10) and (11). When using a prior, an accuracy of 78% is obtained when using only two samples per gesture, 87% accuracy is obtained using five samples. Without the prior, respectively seven and twelve samples per gesture are needed to obtain the same accuracies. When the number of training examples is increased, the benefits of the prior progressively disappear: for more than 20 examples, using the prior brings no discernible benefits. These results highlight the excellent ability to deal with small-data situations, which may be particularly useful, e.g., for medical applications.

Supplementary Note 12: Discussion of the design choices of the Bayesian machine

Comparison with memristor-based neural networks accelerators

In recent years, several realizations have shown the potential of using memristors or other emerging resistive memories such as phase-change memory (PCM) and spin-torque magnetic random access memory (MRAM) for machine learning accelerators. All these implementations associate closely memory and computing functions to eliminate the energy cost of the von Neumann bottleneck. Besides this fundamental concept, they differ massively in the way that they use the memory devices.

At the end of the main paper, Extended Data Table 1 lists major characteristics of our work and high-profile fabricated machine learning accelerators employing in- or near-memory computing based on emerging memories, published since 2020. All these published works target neural networks: our system is the first fabricated memristor-based Bayesian machine. Interestingly, these neural network realizations show a wide range of design choices, reflecting adaptation to different types of technologies, and different design styles. Still, Bayesian computing differs significantly from neural networks, which led us to choices that stand out.

Some memristive technologies and phase change memories feature analog storage. Some neural network implementations ([14],[15],[16] and[17]) exploit this property. On the other hand, others rely on single-bit basic cells to simplify the periphery circuitry[18] or due to limitations of the device[19]. Note that[19] relies on single-bit-per-cell storage, but still uses analog computation to compute neuronal activation function. Despite the complexity of the associated analog or mixed-signal circuitry (analog-to-digital or time-to-digital-converter), the use of analog storage can be attractive for neural network implementations for two main reasons:

- The basic operation of a neural network, multiply-and-accumulate (MAC), can naturally be realized using Ohm's law and Kirchhoff's current law when memristors are used this way. When performing this in-memory computing operation, several devices can be read simultaneously, using the same periphery circuitry. This parallelism greatly reduces the energy cost of analog and mixed-signal periphery circuitry.
- Only low precision is required for synaptic weight for neural network inference.

The situation is different for Bayesian inference. First, MAC is not needed, greatly limiting the main benefit of analog computation in neural networks and increasing its energy cost due to periphery circuits. Additionally, higher precision is required for storing likelihood than for synaptic weights in neural networks (our machine uses eight-bit precision, which is not achieved with analog memristors). For this reason, our design relies on single-level bit cells read with extremely simple and highly energy-efficient precharge sense amplifiers. These sense amplifiers apply a current to the memristors only as long as necessary to differentiate the resistance between a memristor and a complementarily-programmed one (see Supplementary Note 3), whereas the analog approach needs to apply a current long enough for the voltages to stabilize, which usually involves relatively slow feedback circuit. Also, our approach avoids the need for energy-hungry analog-to-digital or time-to-digital converters, as the sense amplifiers naturally provide a digital output. The reliance on this periphery circuitry has other important benefits that distinguish our work from neural networks implementations.

- Our circuit is highly flexible in terms of supply voltage (see main article, Fig. 3). Due to its differential nature, the sense amplifier functions over a wide range of voltage, without any need for recalibration or adjustment of any reference. This feature can be particularly useful in environments with little energy available (e.g., relying on variable energy harvesting), or in conjunction with dynamic voltage scaling. By contrast, neural network accelerators employing analog and/or mixed-signal circuitry are designed to function at a specific supply voltage.
- Analog approaches usually require calibration and compensation mechanisms to eliminate circuit and device imperfections (e.g., voltage offsets due to circuit variability)[16–19]. Our simple read circuit does not need any of this, greatly simplifying circuit operation and its flexibility in all types of conditions.

The memristor-based neural network accelerators have reached better technological maturity than our Bayesian system, with several systems implemented in sub-30-nanometer CMOS[17–19]. The memory array in[18], based on a fully commercial technology, possesses an impressive number of 2M devices. At the same time, state-of-the-art memristor-based network accelerators usually feature a single memory array[15–19], or in the case of[14], three independent memory arrays. Our Bayesian system is composed of 16 tiny memory arrays that function together to perform Bayesian inference. Our system, therefore, validates the possibility of distributing locally to multiple memory arrays the multiple higher-than-nominal voltages to form and program memristors (see Supplementary Note 2).

This discussion allows us to clearly define the contribution of our realization:

- Our integrated circuit is the first fully fabricated memristor-based Bayesian inference system.

- Compared with memristor-based neural network accelerators, our system shows outstanding flexibility and simplicity (possibility to vary supply voltage, absence of calibration or compensation), and high robustness to read disturb (see Supplementary Note 3), device variability (see Supplementary Note 6), and outstanding robustness to single-upset events (see Supplementary Note 7). All these features are due to the use of single-level cells read with particularly simple, robust, and energy-efficient precharge sense amplifiers.
- Our integrated circuit is a full system that features 16 memory blocks that perform Bayesian inference together, therefore solving the challenge of the distribution of the various voltages to form memristors.

Choice of Bayesian paradigm

	This work	Dalgaty et al. 2021[20]	Dalgaty et al. 2021[21]
Current status	**Full system implemented**	Hybrid SW/HW exp.	Hybrid SW/HW exp.
Device	**HfOx memristor**	HfOx memristor	HfOx memristor
Use of the device	**Local digital memory**	Local analog memory	Local analog memory
Concept	**Low-energy Bayesian inference**	Bayesian learning	Bayesian neural network inference

	Gao et al., 2021[22]	Vodenicarevic et al., 2017[23]	Faria et al., 2018[24]
Current status	Simulated	Hybrid SW/HW exp.	Simulated
Device	Memristor	Stochastic MTJ	Stochastic MTJ
Use of the device	Local analog memory	RNG	RNG
Concept	Resilient neural network inference	Low-energy Bayesian inference	Low-energy Bayesian inference

Supplementary Table 2. Comparison of our work with approaches of the literature associating nanoelectronics with Bayesian concepts. Abbreviations. SW: software. HW: hardware. MTJ: magnetic tunnel junction. RNG: random number generation.

In the literature, other works have explored connections between emerging memories and Bayesian inference. Supplementary Table 2 lists several recent works along this idea, and compares them with the Bayesian machine.

Our Bayesian machine is the only fully fabricated nanomemory-based Bayesian system. Other works from the state of the art either relied on computer simulations[22,24], or on hybrid hardware/software realizations, where experimental nanodevices are used, and the rest of the system is simulated on a computer[20,21,23]. Second, and more fundamentally, the Bayesian machine differs conceptually from other proposals of the literature.

The works of[20,21] focus on the implementation of Bayesian neural networks, a special class of Bayesian model that can model uncertainty much better than conventional neural networks, but do not feature the explainability of the more traditional Bayesian inference addressed by our machine. These two works use memristors as main memory, as in our machine, and exploit the variability of memristors as a source of random variable. On the other hand our machine focuses on reliability by eliminating the impact of memristor variability (see Supplementary Note 6).

The work of[22] uses Bayesian inference in a very different way. Unlike all other works reported in Supplementary Table 2, the final goal of this work is to implement non-Bayesian machine learning model (a conventional neural network). By treating these networks as Bayesian neural networks modeling memristor variability, the authors are able to tolerate memristor imperfection better than more conventional approaches. In this work, memristors also implement memory.

References[23,24] differ from all other works of Supplementary Table 2, in that the nanodevices are not used as memory, but as random bit generators (replacing the linear feedback shift registers). These works are not full-system studies, as they do not address the memory question. Also, they focus on random variables with binary values, unlike our Bayesian machine that deals with multiple-valued inputs and outputs. On the other hand, we see in the next section that incorporating some ideas from these works is a natural prospect for our Bayesian machine.

Assessment of stochastic computing and of the reliance on linear-feedback shift registers for random bit generation

Finally, we evaluate the choice of stochastic computing for performing Bayesian inference. A major benefit of stochastic computing is its intrinsic tolerance to single-event upsets (see Supplementary Note 9). It also has favorable properties in terms of energy consumption.

To evaluate stochastic computing, we designed an alternative implementation of the Bayesian machine relying on conventional integer (i.e., fixed point) arithmetic, and using the same 130-nanometer process as the one use for the stochastic-computing-based Bayesian machine. We find that this implementation requires 0.31 nanojoules to perform the arithmetic operations of Bayesian inference. The energy consumed by the stochastic computing version of the Bayesian machine is reported in Fig. 4f of the main article. We see that when using the power-conscious approach, the stochastic computing version is always at least three times more energy-efficient than the conventional arithmetic version. On the other hand, the conventional stochastic computing version is more energy-efficient than conventional arithmetic if we target an accuracy on gesture recognition of 86% or less. This result is consistent with the general idea that stochastic computing is favorable in situations where the highest precision is not required.

The energy efficiency of the stochastic Bayesian machine could significantly improve in the future. Fig. 4d of the main article reveals that in the current design, 75% of the energy is spent for the random number generation by the LFSRs and their distribution to the likelihood blocks ("vertical wires"). Several recent works have shown the possibility of generating random bits at a very low energy cost using stochastic nanodevices such as superparamagnetic tunnel junction[23–26] or random telegraph noise in RRAMs[27]. These proposals rely on natural fluctuations of the devices due to thermal or random telegraph noise and can therefore generate random bits relying only on read operation, at a very low cost. They could be distributed within likelihood arrays, thanks to their small area, and generate random bits at a much lower energy cost than LFSR (in the order of femtojoule/bit) and without requiring vertical wires. In the particular case of "p-bits", the Gupta circuit, which consumes in our design 22% of the inference energy could also be avoided, as this concept provides random bits with easily adjustable probabilities[25,26].

Note that other works have proposed to generate random bits by exploiting the write operations of emerging memories (e.g., in spin-torque MRAM[28] or in RRAM[29]). These proposals can generate high-quality random bits at high speed, but would not necessarily be adapted for the Bayesian machine. Due to the need for a write operation to generate a random bit, their energy consumption is comparable or higher than the LFSRs used in the current version of the machine.

References

1. Zhao, W., Chappert, C., Javerliac, V. & Noziere, J.-P. High speed, high stability and low power sensing amplifier for mtj/cmos hybrid logic circuits. *IEEE Transactions on Magn.* **45**, 3784–3787 (2009).

2. Zhao, W. *et al.* Synchronous non-volatile logic gate design based on resistive switching memories. *IEEE Transactions on Circuits Syst. I: Regul. Pap.* **61**, 443–454 (2014).

3. Grossi, A. *et al.* Electrical characterization and modeling of 1t-1r rram arrays with amorphous and poly-crystalline hfo2. *Solid-State Electron.* **128**, 187–193 (2017).

4. Chang, M.-F. *et al.* A 0.5 v 4mb logic-process compatible embedded resistive ram (reram) in 65nm cmos using low-voltage current-mode sensing scheme with 45ns random read time. In *2012 IEEE International Solid-State Circuits Conference*, 434–436 (IEEE, 2012).

5. Bessière, P., Mazer, E., Ahuactzin, J. M. & Mekhnacha, K. *Bayesian programming* (CRC press, 2013).

6. Winstead, C. Tutorial on stochastic computing. In *Stochastic Computing: Techniques and Applications*, 39–76 (Springer, 2019).

7. Alaghi, A. & Hayes, J. P. Survey of stochastic computing. *ACM TECS* **12**, 92 (2013).

8. Hirtzlin, T. *et al.* Digital biologically plausible implementation of binarized neural networks with differential hafnium oxide resistive memory arrays. *Front. neuroscience* **13**, 1383 (2020).

9. Rai, A. Explainable ai: From black box to glass box. *J. Acad. Mark. Sci.* **48**, 137–141 (2020).

10. Goodfellow, I. J., Shlens, J. & Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

11. Amodei, D. *et al.* Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565* (2016).

12. Letham, B., Rudin, C., McCormick, T. H. & Madigan, D. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *The Annals Appl. Stat.* **9**, 1350–1371 (2015).

13. Murphy, K. P. *Machine learning: a probabilistic perspective* (MIT press, 2012).

14. Li, C. *et al.* Cmos-integrated nanoscale memristive crossbars for cnn and optimization acceleration. In *2020 IEEE International Memory Workshop (IMW)*, 1–4 (IEEE, 2020).

15. Yao, P. *et al.* Fully hardware-implemented memristor convolutional neural network. *Nature* **577**, 641–646 (2020).

16. Wan, W. *et al.* 33.1 a 74 tmacs/w cmos-rram neurosynaptic core with dynamically reconfigurable dataflow and in-situ transposable weights for probabilistic graphical models. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, 498–500 (IEEE, 2020).

17. Khaddam-Aljameh, R. *et al.* Hermes-core—a 1.59-tops/mm 2 pcm on 14-nm cmos in-memory compute core using 300-ps/lsb linearized cco-based adcs. *IEEE J. Solid-State Circuits* **57**, 1027–1038 (2022).

18. Xue, C.-X. *et al.* A cmos-integrated compute-in-memory macro based on resistive random-access memory for ai edge devices. *Nat. Electron.* **4**, 81–90 (2021).

19. Jung, S. *et al.* A crossbar array of magnetoresistive memory devices for in-memory computing. *Nature* **601**, 211–216 (2022).

20. Dalgaty, T. *et al.* In situ learning using intrinsic memristor variability via markov chain monte carlo sampling. *Nat. Electron.* **4**, 151–161 (2021).

21. Dalgaty, T., Esmanhotto, E., Castellani, N., Querlioz, D. & Vianello, E. Ex situ transfer of bayesian neural networks to resistive memory-based inference hardware. *Adv. Intell. Syst.* **3**, 2000103 (2021).

22. Gao, D. *et al.* Bayesian inference based robust computing on memristor crossbar. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 121–126 (IEEE, 2021).

23. Vodenicarevic, D. *et al.* Low-energy truly random number generation with superparamagnetic tunnel junctions for unconventional computing. *Phys. Rev. Appl.* **8**, 054045 (2017).

24. Faria, R., Camsari, K. Y. & Datta, S. Implementing bayesian networks with embedded stochastic mram. *AIP Adv.* **8**, 045101 (2018).

25. Camsari, K. Y., Sutton, B. M. & Datta, S. P-bits for probabilistic spin logic. *Appl. Phys. Rev.* **6**, 011305 (2019).

26. Borders, W. A. *et al.* Integer factorization using stochastic magnetic tunnel junctions. *Nature* **573**, 390–393 (2019).

27. Govindaraj, R., Ghosh, S. & Katkoori, S. Csro-based reconfigurable true random number generator using rram. *IEEE Transactions on Very Large Scale Integration (VLSI) Syst.* **26**, 2661–2670 (2018).

28. Fukushima, A. *et al.* Spin dice: A scalable truly random number generator based on spintronics. *Appl. Phys. Express* **7**, 083001 (2014).

29. Balatti, S., Ambrogio, S., Wang, Z. & Ielmini, D. True random number generation by variability of resistive switching in oxide-based devices. *IEEE J. on Emerg. Sel. Top. Circuits Syst.* **5**, 214–221 (2015).