Supplementary Material: Hardware-Adaptive and Superlinear-Capacity Memristor-based Associative Memory

Chengping He[1], Mingrui Jiang[1], Keyi Shan[1], Szu-Hao Yang[1], Zefan Li[1], Shengbo Wang[1], Giacomo Pedretti[2], Jim Ignowski[2], and Can Li[1,3,*]

[1]Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong SAR, China
[2]Hewlett Packard Labs, Hewlett Packard Enterprise, Milpitas, CA, USA
[3]Center for Advanced Semiconductor and Integrated Circuit, The University of Hong Kong, Hong Kong SAR, China
[*]Email: canl@hku.hk

Learning Algorithm of Hopfield Neural Network for Associate Memory

The Hopfield neural network is a type of neural network designed for associative memory. To store patterns in the system, the learning algorithm plays a critical role. The most notable learning algorithm for Hopfield neural networks is Hebb's learning rule, introduced by Donald Hebb in 1949[1]. This rule states that when two nodes are simultaneously active, the connection between them is strengthened. The learning algorithm can be expressed mathematically as:

$$\mathbf{W}_{ij} = \sum_m \xi_i^m \xi_j^m \tag{1}$$

Here, ξ^m represents the mth pattern that we aim to store in the network.

Hebb's learning rule has a limited storage capacity, several previous works have been introduced to enhance the system's capacity, such as the Storkey learning rule[2]. Unlike Hebb's learning rule, Storkey's rule incorporates an additional mechanism that improves the learning process. Storkey's rule updates the weights pattern by pattern, ensuring a more efficient and effective storage of patterns in the system. The learning rule can be expressed as follows:

$$\mathbf{W}_{ij} = \mathbf{W}_{ij} + \frac{1}{n}\xi_i^m \xi_j^m - \frac{1}{n}\xi_i^m h_j^m - \frac{1}{n}\xi_j^m h_i^m \tag{2}$$

where $\mathbf{C}^{-1}$ is the inverse matrix of correlation matrix $\mathbf{C}$:

$$h_i^m = \sum_k \mathbf{W}_{ik}\xi_k^m \tag{3}$$

Another advanced method for improving the capacity and reliability of pattern storage in Hopfield networks is the pseudo-inverse learning algorithm[3]. This method minimizes the interference between stored patterns, offering better performance than Hebb's or Storkey's rules, especially in systems requiring higher storage capacity. The pseudo-inverse learning rule is given by:

$$\mathbf{W}_{ij} = \frac{1}{n}\sum_{m,k} \xi_i^m (\mathbf{C}^{-1})_{m,k}\xi_j^k \tag{4}$$

where:

$$C_{mk} = \frac{1}{n}\sum_i \xi_i^m \xi_i^k \tag{5}$$

All the learning algorithms discussed above—Hebb's rule, Storkey's rule, and the pseudo-inverse learning algorithm—focus exclusively on learning weights based on the stored patterns. These methods rely solely on the information contained in the patterns themselves and do not account for variations or imperfections in the physical components of the system, such as memristors. Consequently, they are unable to adapt the weights dynamically to address issues like hardware variability, noise, or non-idealities in the memristor-based implementation.

Equilibrium propagation is another learning algorithm that operates in two distinct phases [4, 5]. The first is the free phase, during which the system propagates through the layers without any external influence. In this phase, the system evolves naturally and settles into a steady state, which is recorded as the free state $\mathbf{A}_{free}$. The second is the nudged phase, where a perturbation is applied to the system. This perturbation causes the system to evolve into a new steady state, recorded as $\mathbf{A}_{nudged}$. The change in weights is then calculated based on the difference between the two steady states, following the rule:

$$d\mathbf{W} = \beta(\mathbf{A}_{nudged}\mathbf{A}_{nudged}^T - \mathbf{A}_{free}\mathbf{A}_{free}^T) \tag{6}$$

Here, β is the learning rate that determines the magnitude of the weight updates. This algorithm allows the system to refine its weights dynamically by leveraging the difference in the steady states induced by the perturbation.

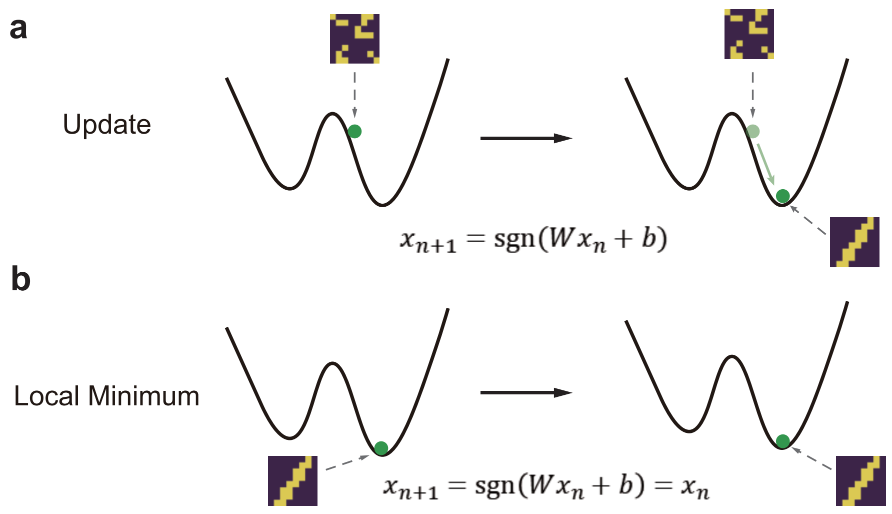

a
Update
$x_{n+1} = \text{sgn}(Wx_n + b)$
b
Local Minimum
$x_{n+1} = \text{sgn}(Wx_n + b) = x_n$

Figure S1: **Update process and local minimum of Hopfield neural network:** (a) When the Hopfield Neural Network is not in a local minimum, the update process causes the system's energy to decrease with each iteration. (b) When the system is already in a local minimum, the state remains unchanged during subsequent iterations.

Figure S1 illustrates the update mechanism of a Hopfield Neural Network (HNN). After each update, the system's energy decreases, and its state changes accordingly. However, if the system is already in a local minimum, further updates will not alter the energy and the state. To store a pattern in the HNN, it must correspond to a local minimum of the energy landscape. In such cases, the update rule:

$$\xi_{n+1} = \text{sgn}(\mathbf{W}\xi_n + b) \tag{7}$$

should result in no change to the state, i.e.,

$$\xi_{n+1} = \xi_n \tag{8}$$

Therefore, for every stored pattern ξ^m, the following condition must hold:

$$\text{sgn}(\mathbf{W}\xi^m + b) = \xi^m \tag{9}$$

Based on this condition, we can define a loss function to guide the learning of the network's weights:

$$\min_{\mathbf{W}} \sum_m \text{Distance}\left(\xi^m, \text{sgn}(\mathbf{W}\xi^m + \mathbf{b})\right) \tag{10}$$

This loss measures the discrepancy between the stored patterns and their corresponding network outputs. Optimization techniques such as Adam, RMSProp, or SGD can be employed to train the weights accordingly.

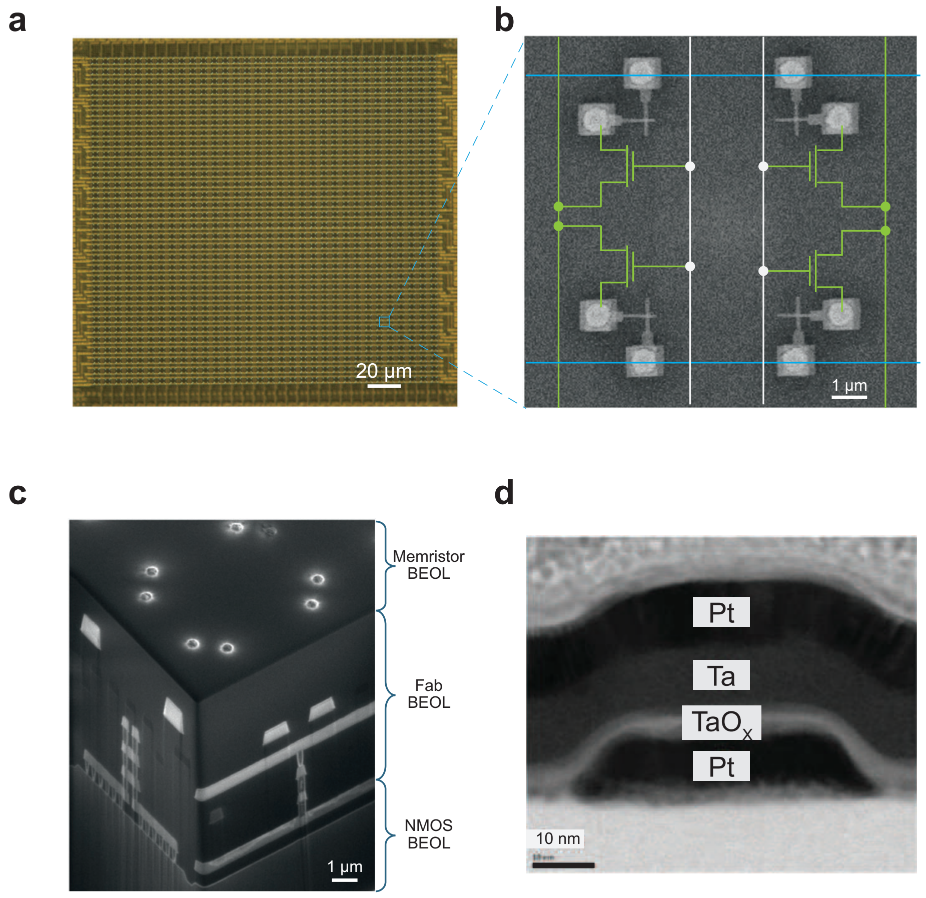
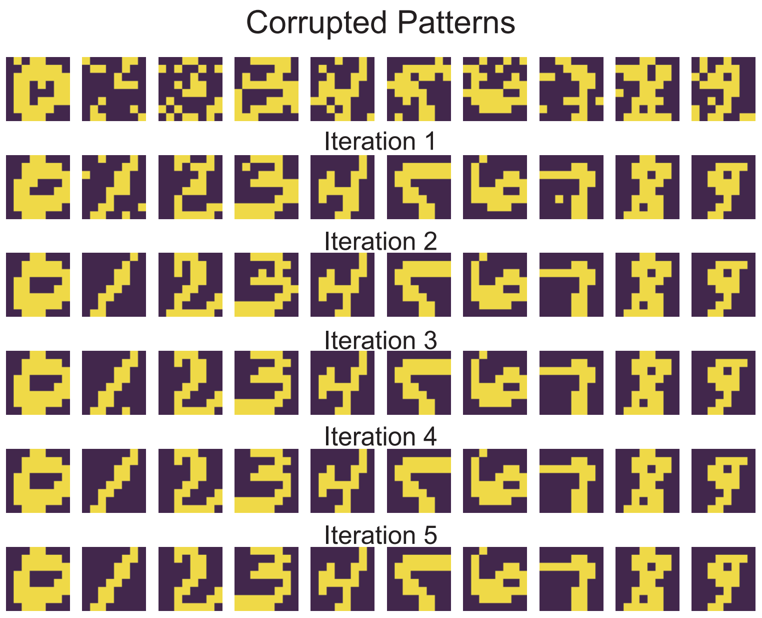


Figure S2: **Integrated memristor chip and devices:** (a) Optical image of a 64 × 64 crossbar array in a fully integrated memristor chip. (b) SEM image of the cross-point four memristor devices. (c) Cross-sectional view of the memristor chip, illustrating the CMOS circuits at the bottom, interconnections in the middle, and metal vias on the surface for memristor integration using back-end processes. (d) Cross-sectional TEM image of the 50 nm×50 nm Pt/Ta/TaOx/Pt memristor device.

a



b

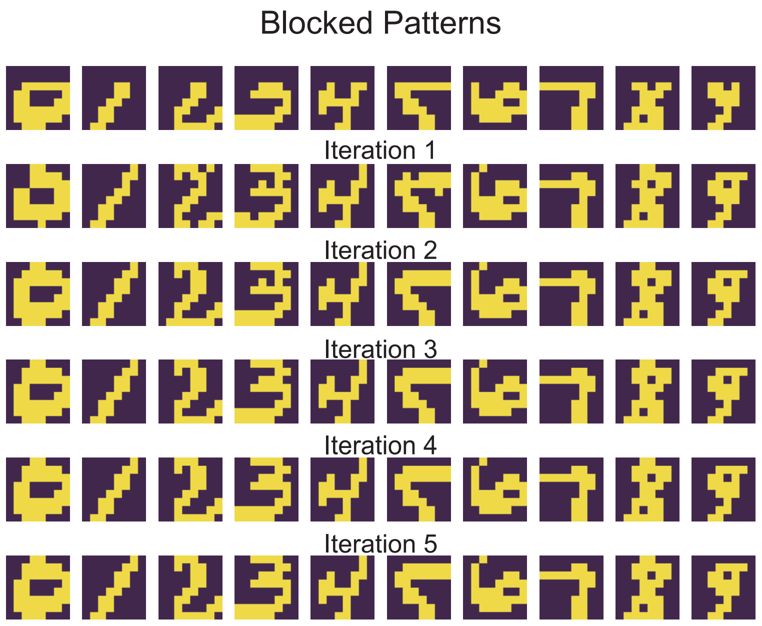


Figure S3: **The retrieve process of the corrupted and blocked patterns:** (a) The experimental retrieval process for corrupted patterns with 10-digit numbers. The corrupted patterns were generated by flipping pixels with a 10% probability. (b) The experimental retrieval process for blocked patterns. The blocked patterns were generated by obscuring the first two rows of the original patterns. Most patterns are successfully retrieved within one or two iterations, and all patterns converge to full retrieval within five iterations.

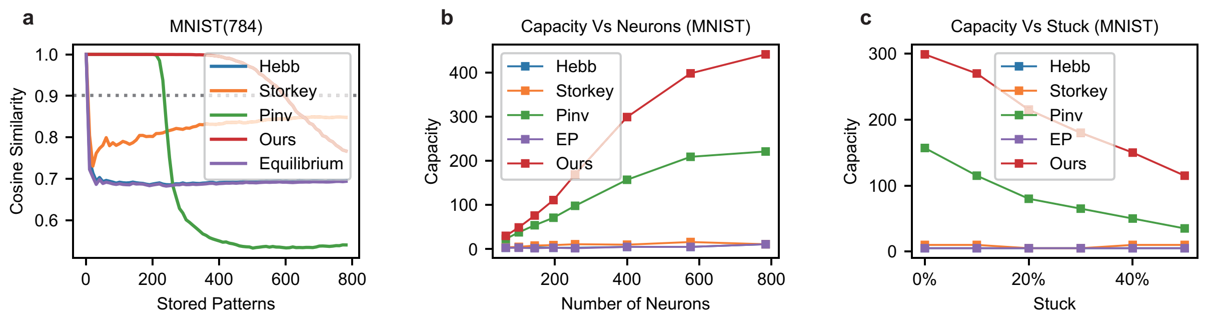


Figure S4: **Performance evaluation on MNIST dataset:** (a) Cosine similarity versus the number of stored pattern int the system with 784 neurons (MNIST). our algorithm retrieves patterns with better quality. (b) System capacity (maximum number of retrievable patterns with cosine similarity > 0.99) versus neuron count for the MNIST dataset. Larger networks yield higher capacity, with our method surpassing other learning algorithms. (c) Impact of stuck-at-fault defects on system capacity (400 neurons, MNIST dataset). While capacity declines with increasing faults, our method exhibits superior defect tolerance, maintaining higher performance.

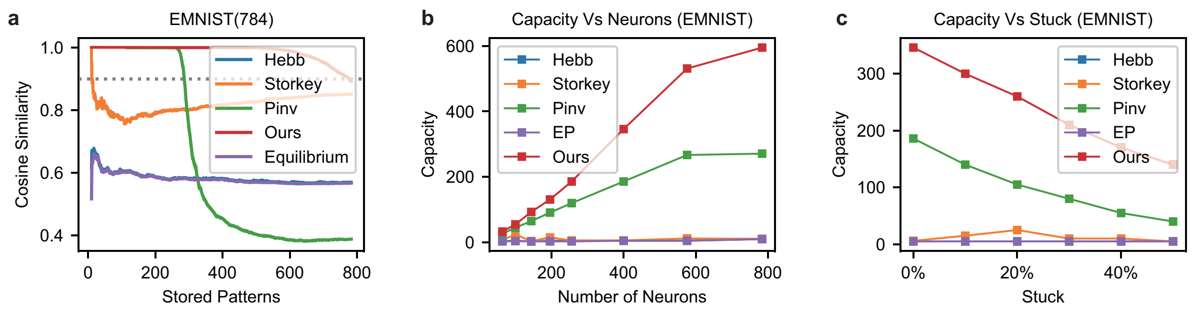


Figure S5: **Performance evaluation on EMNIST dataset:** (a) Cosine similarity versus the number of stored pattern int the system with 784 neurons (EMNIST). our algorithm retrieves patterns with better quality. (b) System capacity (maximum number of retrievable patterns with cosine similarity > 0.99) versus neuron count for the EMNIST dataset. Larger networks yield higher capacity, with our method surpassing other learning algorithms. (c) Impact of stuck-at-fault defects on system capacity (400 neurons, EMNIST dataset). While capacity declines with increasing faults, our method exhibits superior defect tolerance, maintaining higher performance.

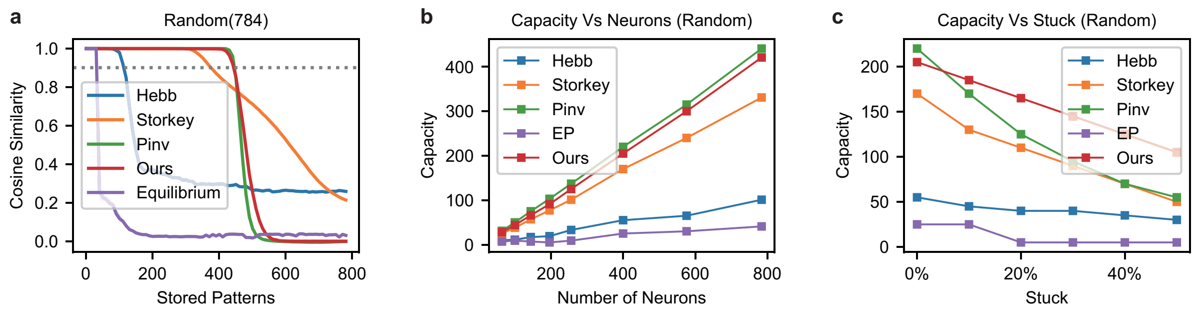


Figure S6: **Performance evaluation on random patterns:** (a) Cosine similarity versus the number of stored pattern int the system with 784 neurons (random patterns). our algorithm retrieves patterns with better quality. (b) System capacity (maximum number of retrievable patterns with cosine similarity > 0.99) versus neuron count for the random patterns. Larger networks yield higher capacity, with our method surpassing other learning algorithms. (c) Impact of stuck-at-fault defects on system capacity (400 neurons, random patterns). While capacity declines with increasing faults, our method exhibits superior defect tolerance, maintaining higher performance.

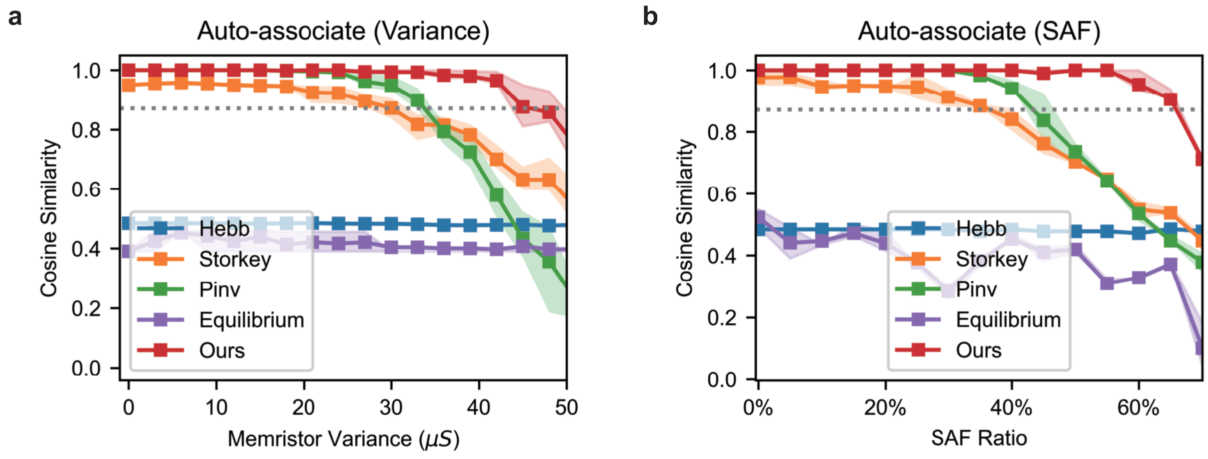


Figure S7: **Impact of device non-idealities of single layer experiments:** The cosine similarity between stored patterns and retrieved patterns is evaluated under varying conditions: (a) memristor device variations and (b) stuck-at-fault ratios, for an associative memory storing 10 digital binary patterns. The results demonstrate that the system exhibits strong robustness against these device non-idealities. Corrupted patterns are generated by flipping the pixels of the original patterns with a 10% probability.

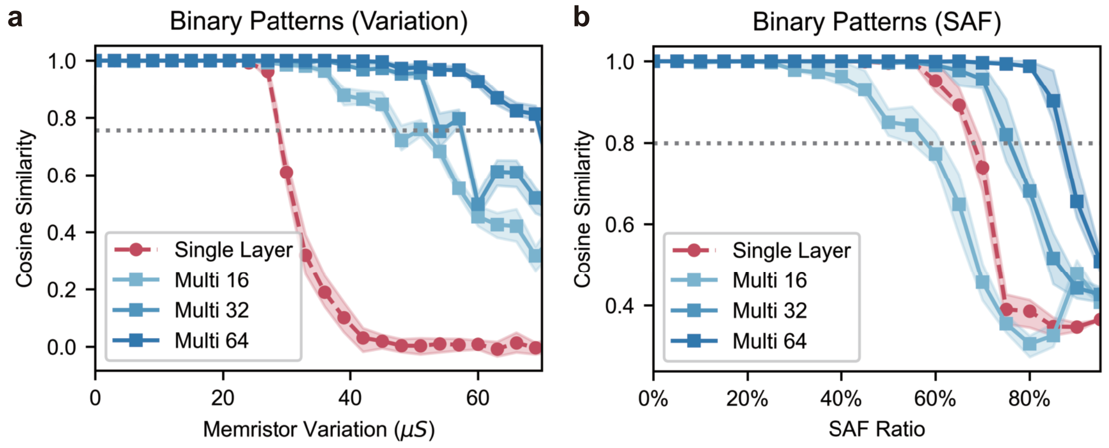


Figure S8: **Impact of device non-idealities of multi-layer experiments for binary patterns:** Cosine similarity between stored and retrieved patterns is evaluated under: (a) memristor variation and (b) stuck-at-fault ratio in an associative memory storing 10 digital binary patterns, using varying numbers of hidden neurons. In a multilayer structure, increasing the number of hidden neurons improves system non-idealities tolerance. Additionally, the multilayer architecture demonstrates superior tolerance to memristor variation compared to a single-layer structure, while requiring only half the number of synapses. Corrupted patterns are generated by flipping the pixels of the original patterns with a 10% probability.

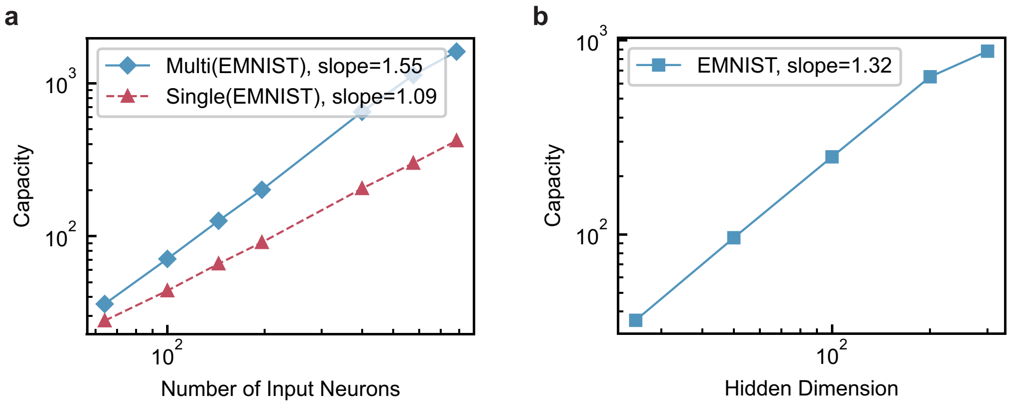


Figure S9: **Capacity analysis versus the number of input neurons and hidden dimension for the EMNIST dataset (binary patterns):** (a) Comparison of capacity between single-layer HNN and multilayer structures, where the hidden layer size in the multilayer networks is set to half the number of input neurons to ensure both architectures have an equal number of synapses. The results show that multilayer networks exhibit a superlinear capacity growth, whereas single-layer networks demonstrate a linear increase in capacity with the number of input neurons. (b) Capacity of the multilayer network as a function of hidden neuron count, with the input size fixed at 400 neurons. The observed capacity exhibits a superlinear relationship with the number of hidden neurons.

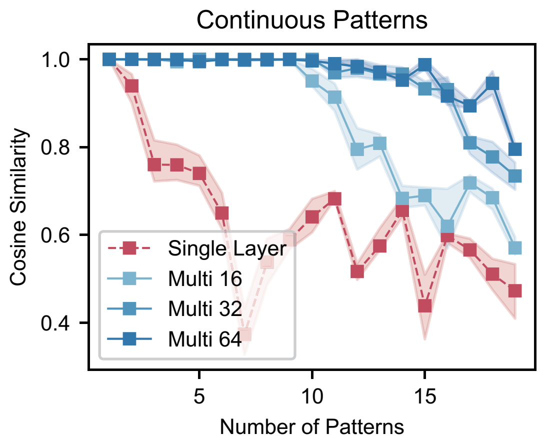


Figure S10: **Cosine Similarity vs. Number of Stored Patterns for the MNIST Dataset:** This analysis compares cosine similarity across single-layer and multilayer architectures with varying numbers of hidden neurons. The single-layer structure demonstrates limited capacity, whereas the multilayer architecture achieves superior performance even with fewer synapses, highlighting its efficiency. Moreover, increasing the number of hidden neurons in the multilayer structure further improves performance. The evaluation includes the addition of Gaussian noise with an amplitude of 0.5.

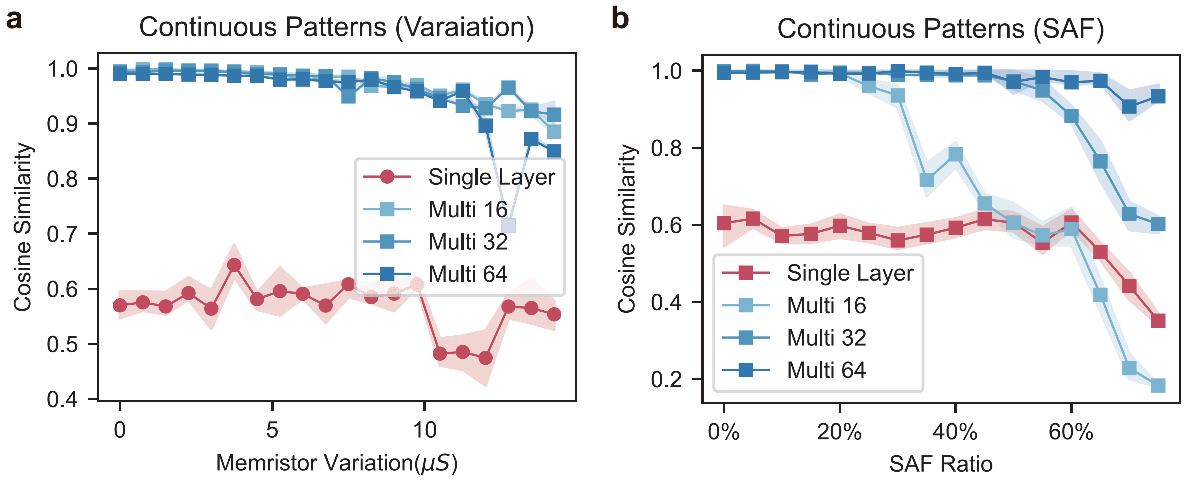


Figure S11: **Impact of device non-idealities of multi-layer experiments for continuous patterns:** Cosine similarity between stored and retrieved patterns is evaluated under two conditions: (a) memristor variation and (b) stuck-at fault ratio, in an associative memory storing 10 digital continuous patterns. The evaluation is conducted across varying numbers of hidden neurons. In a multilayer architecture, increasing the number of hidden neurons enhances the system's tolerance to non-idealities. In contrast, the single-layer structure lacks the capacity to store all patterns effectively under the same conditions.

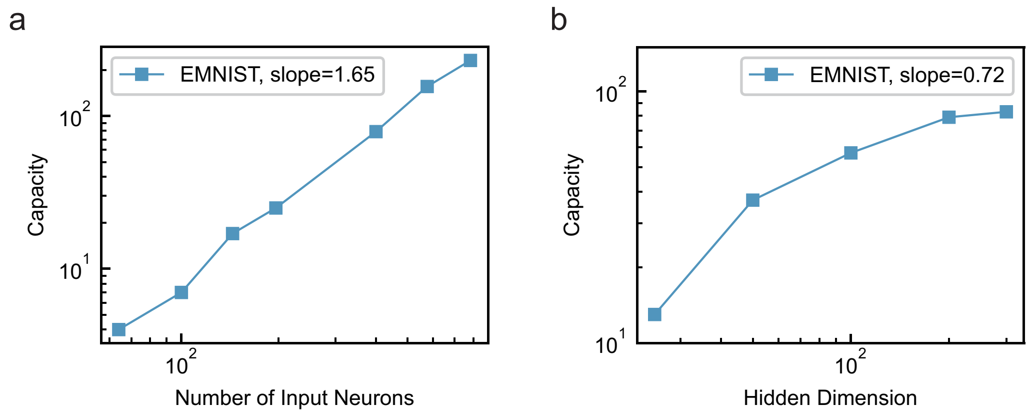
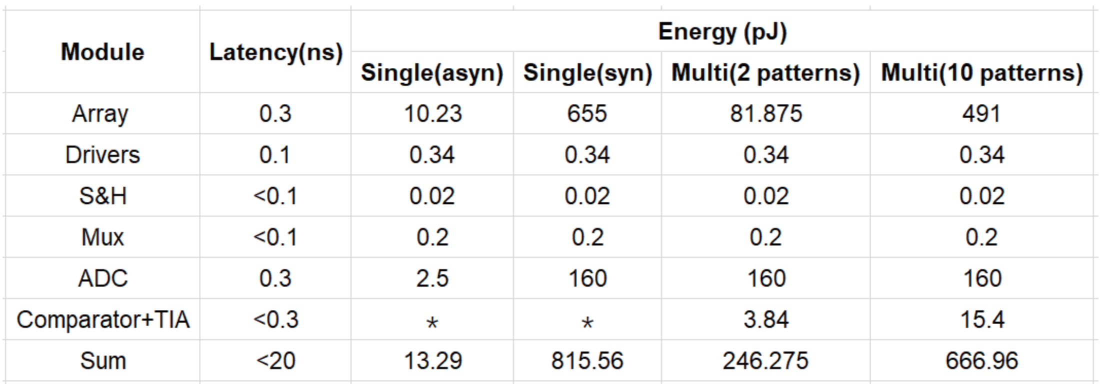


Figure S12: **Capacity analysis versus the number of input neurons and hidden dimension for the EMNIST dataset (continuous patterns):** (a) Capacity vs. number of neurons in multi-layer networks.For the multi-layer architecture, the hidden layer size is set to half the number of input neurons. The results demonstrate that multi-layer networks exhibit superlinear capacity growth on the EMNIST dataset. (b) Capacity vs. hidden neuron count in fixed-input networks. With the input size fixed at 400 neurons, the network's capacity increases as the number of hidden neurons grows.



Module	Latency(ns)	Energy (pJ)			
		Single(asyn)	Single(syn)	Multi(2 patterns)	Multi(10 patterns)
Array	0.3	10.23	655	81.875	491
Drivers	0.1	0.34	0.34	0.34	0.34
S&H	<0.1	0.02	0.02	0.02	0.02
Mux	<0.1	0.2	0.2	0.2	0.2
ADC	0.3	2.5	160	160	160
Comparator+TIA	<0.3	*	*	3.84	15.4
Sum	<20	13.29	815.56	246.275	666.96

Figure S13: **Latency and Energy Consumption for Different Methods per Iteration:** This section compares the latency and energy consumption of various methods for associative memory. To ensure a fair comparison, all estimations are based on a 180 nm technology node. For the analog-to-digital converter (ADC), we consider a state-of-the-art 5-bit successive-approximation register ADC as reported in [6]. The associative memory implementations are benchmarked using the same operating frequency as mem-HNN, i.e., 25 MHz (corresponding to 20 ns per iteration). For the multilayer structure, performance evaluation is carried out using a comparator and trans-impedance amplifier (TIA) as the analog interconnection between layers. The power consumption for this setup is approximately 12 µW, as reported in [7].

Figure S13 presents the energy consumption and latency per iteration for various methods, along with different numbers of stored patterns. It is important to note that the number of iterations required to retrieve a stored pattern varies depending on the method used.

For the asynchronous update in a single-layer architecture, the expected number of iterations to retrieve a pattern is approximately 166.12. In contrast, when two patterns are stored in the system, the synchronous update method requires only about 1.003 iterations, while the multilayer architecture consistently requires just 1.0 iteration. When the number of stored patterns increases to ten, the single-layer architecture with synchronous update needs around 2.13 iterations, yet the multilayer architecture still reliably retrieves the correct pattern in a single step.

This consistency in the multilayer design highlights its robustness and efficiency in handling associative memory. Using these iteration counts, we can straightforwardly estimate the total energy consumption and latency of each method by multiplying the per-iteration values with the respective number of iterations needed for retrieval. This provides a clear comparative metric for evaluating the overall performance and scalability of each approach.

References

[1] Hebb Do. The organization of behavior. *New York*, 1949.

[2] Amos J Storkey and Romain Valabregue. The basins of attraction of a new hopfield learning rule. *Neural Networks*, 12(6):869–876, 1999.

[3] I Kanter and Haim Sompolinsky. Associative recall of memory without errors. *Physical Review A*, 35(1):380, 1987.

[4] Gianluca Zoppo, Francesco Marrone, and Fernando Corinto. Equilibrium propagation for memristor-based recurrent neural networks. *Frontiers in neuroscience*, 14:501774, 2020.

[5] Su-in Yi, Jack D Kendall, R Stanley Williams, and Suhas Kumar. Activity-difference training of deep neural networks using memristor crossbars. *Nature Electronics*, 6(1):45–51, 2023.

[6] Can Li, Jim Ignowski, Xia Sheng, Rob Wessel, Bill Jaffe, Jacqui Ingemi, Cat Graves, and John Paul Strachan. Cmos-integrated nanoscale memristive crossbars for cnn and optimization acceleration. In *2020 IEEE International Memory Workshop (IMW)*, pages 1–4. IEEE, 2020.

[7] Ben Feinberg, Ryan Wong, T Patrick Xiao, Christopher H Bennett, Jacob N Rohan, Erik G Boman, Matthew J Marinella, Sapan Agarwal, and Engin Ipek. An analog preconditioner for solving linear systems. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 761–774. IEEE, 2021.