PAPER • **OPEN ACCESS**

Distributed representations enable robust multi-timescale symbolic computation in neuromorphic hardware

To cite this article: Madison Cotteret *et al* 2025 *Neuromorph. Comput. Eng.* **5** 014008

NEUROMORPHIC
Computing and Engineering

PAPER

Distributed representations enable robust multi-timescale symbolic computation in neuromorphic hardware

Madison Cotteret[1,2,3,*], **Hugh Greatorex**[1,2], **Alpha Renner**[4], **Junren Chen**[5], **Emre Neftci**[4], **Huaqiang Wu**[6], **Giacomo Indiveri**[5], **Martin Ziegler**[7] and **Elisabetta Chicca**[1,2]

[1] Bio-Inspired Circuits and Systems (BICS) Lab, Zernike Institute for Advanced Materials University of Groningen, Groningen, The Netherlands
[2] Groningen Cognitive Systems and Materials Center (CogniGron) University of Groningen, Groningen, The Netherlands
[3] Micro- and Nanoelectronic Systems (MNES), Technische Universität Ilmenau, Ilmenau, Germany
[4] Forschungszentrum Jülich, Jülich, Germany
[5] Institute of Neuroinformatics, University of Zurich and ETH Zurich, Zurich, Switzerland
[6] School of Integrated Circuits, Tsinghua University, Beijing, People's Republic of China
[7] Energy Materials and Devices, Department of Materials Science, Kiel University, Kiel, Germany
[*] Author to whom any correspondence should be addressed.

E-mail: m.cotteret@rug.nl

Keywords: attractor networks, distributed computation, vector symbolic architectures, neuromorphic hardware abstractions, hyperdimensional computing, hardware interoperability, fault-tolerant neuromorphic algorithms

Supplementary material for this article is available online

OPEN ACCESS

RECEIVED
27 September 2024

REVISED
18 December 2024

ACCEPTED FOR PUBLICATION
9 January 2025

PUBLISHED
7 February 2025

Abstract

Programming recurrent spiking neural networks (RSNNs) to robustly perform multi-timescale computation remains a difficult challenge. To address this, we describe a single-shot weight learning scheme to embed robust multi-timescale dynamics into attractor-based RSNNs, by exploiting the properties of high-dimensional distributed representations. We embed finite state machines into the RSNN dynamics by superimposing a symmetric autoassociative weight matrix and asymmetric transition terms, which are each formed by the vector binding of an input and heteroassociative outer-products between states. Our approach is validated through simulations with highly nonideal weights; an experimental closed-loop memristive hardware setup; and on Loihi 2, where it scales seamlessly to large state machines. This work introduces a scalable approach to embed robust symbolic computation through recurrent dynamics into neuromorphic hardware, without requiring parameter fine-tuning or significant platform-specific optimisation. Moreover, it demonstrates that distributed symbolic representations serve as a highly capable representation-invariant language for cognitive algorithms in neuromorphic hardware.

1. Introduction

Neuromorphic computing promises to match the efficiency of information processing in the brain by emulating biological neural and synaptic dynamics in hardware [1]. In contrast to more conventional artificial neural networks, biological neurons have rich internal temporal dynamics and interact sparsely with unary events (spikes). Performing multi-timescale tasks—such as motor planning and execution—with biologically-plausible spiking neurons requires information to be retained and processed for periods much longer than the timescales available in individual neurons and synapses [2, 3]. Information must then be represented by the collective dynamics of recurrently-connected populations of neurons instead. Programming recurrent spiking neural networks (RSNNs) to perform long-timescale tasks with short-timescale neurons remains a formidable challenge however, due to the conflicting demands that the network should react quickly to input but otherwise be stable on long timescales [4].

In the face of these challenges, a possible approach would be to apply gradient-based iterative learning algorithms to train the network parameters to solve a particular task [5–7]. However, besides being computationally intensive and biologically implausible, such approaches are not well-suited to tasks with

long timescales, due to the need to backpropagate gradients into the distant (potentially infinite) past [8, 9]. Furthermore, these approaches do not have guarantees of stability and robustness, compared to those afforded by much simpler biologically-inspired models like the Hopfield attractor network [10, 11].

If the trained networks are to be deployed on mixed-signal neuromorphic hardware, one must usually perform calibrating chip-in-the-loop training, lest device-specific nonidealities absent during training lead to a catastrophic degradation in network performance upon deployment [6, 12–15]. To overcome these difficulties and to shift to more robust training procedures, it has been suggested that a more appropriate level of description for programming neuromorphic hardware may be at the level of distributed high-dimensional patterns of neuron activity, rather than individual neuron and synapse parameters [16]. With vector symbolic architectures (VSAs)—also known as hyperdimensional computing—arbitrary symbolic data structures can be represented by high-dimensional random vectors, known as *hypervectors*, such that information is distributed across the entire vector [17–20]. These representations are then not dependent upon the precise functioning of individual components, which is a critical property for robust functioning in neuromorphic hardware and biology [21–24].

In this work, we show how distributed representations can be leveraged to program arbitrary state machines into RSNNs in a scalable and robust manner. For each state, we add an autoassociative outer product to the recurrent weight matrix to store it as a fixed-point attractor in the RSNN [10]. Additional heteroassociative outer product terms are then superimposed, which are each responsible for storing an input-triggered transition between attractor states. Neural state machines have already been shown to be important computational primitives for reproducing cognitive behaviours in neuromorphic agents [25], and have been used in the context of solving constraint satisfaction problems [26].

We first embed multiple state machines into simulated RSNNs with nonideal synaptic weights. To validate these simulations in neuromorphic hardware, we create a proof-of-concept implementation using a memristive crossbar setup run in a closed-loop with simulated neurons. Finally, we implement these networks on Intel's asynchronous digital neuromorphic research chip Loihi 2 [27], to demonstrate the algorithm's scalability.

This work demonstrates the capability of distributed symbolic representations as a functional abstraction layer for neuromorphic hardware [16]. It permits the description of complex RSNN behaviours from a high level while being invariant to the choice of underlying neural representation, and is highly robust to hardware-specific nonidealities. It is a step towards a general framework for interoperability of cognitive algorithms across varying neuromorphic hardware architectures, without requiring significant optimisation or fine-tuning upon deployment to each particular platform.

2. Results

State machines are perhaps the most comprehensive model of multi-timescale dynamics, as they react quickly to input but are otherwise stable on indeterminately-long timescales. They also have a clear utility for building more complex models of cognitive behaviours [25, 26, 28]. We focus on the task of embedding deterministic finite automata (DFAs) into RSNNs by expressing the DFA in terms of high-dimensional attractor states and transitions between them. For every state q in the finite set of states Q, we embed an associated fixed-point attractor state in the RSNN dynamics that is stable on long timescales. When input is given to the network, the RSNN should quickly transition between attractor states according to the desired state transition function $F : Q \times S \to Q$ in the DFA, where S is the set of all input symbols.

We generate independent hypervectors $\mathbf{q} \in \{0, 1\}^N$ to represent each DFA state q, and store them as attractors in the RSNN using a Hopfield-like outer-product learning rule [11, 29, 30]. The hypervectors follow a sparse block structure, i.e. the N elements are split up into M blocks of equal length L, and exactly one entry in each block is nonzero [31, 32].

We use sparse rather than dense vectors to benefit from the increased capacity in attractor networks [29, 30, 33] and distributed rather than localist representations for improved robustness properties [34]. We use a sparse block structure due to the simplicity of the required neural activation function, and compatibility with the VSA framework [32, 35]. Additionally, we move closer to a more plausible model of biological neural dynamics, where sparse representations enforced by local competitive mechanisms are abundant [36, 37].

For every state transition, we embed additional outer product terms such that if the correct subset of neurons is masked by external inhibition, the RSNN will transition between attractor states (see Methods). The terms are of the form $(\mathbf{q}_1 - \mathbf{q}_0)(\mathbf{q}_0 \circ \bar{\mathbf{s}})^\intercal$ where $\bar{\mathbf{s}}$ is another (bipolar) hypervector, and '$\circ$' is a Hadamard product, acting as a hypervector binding operation. Without input to the network, these terms are pseudo-orthogonal to the network state, i.e. $\langle(\mathbf{q}_0 \circ \bar{\mathbf{s}}) \cdot \mathbf{q}_0\rangle = 0$ and so have negligible effect. When the network is masked by the correct (binary) hypervector $\mathbf{s}$ however (effectively an unbinding operation), they become similar to the network state, i.e. $(\mathbf{q}_0 \circ \bar{\mathbf{s}}) \cdot (\mathbf{q}_0 \wedge \mathbf{s}) = \frac{1}{2}M$ where '$\wedge$' is component-wise AND,

implementing the masking. This allows us to effectively modulate the strength of superimposed terms by masking neurons in the RSNN [22].

We then wish to program this weight matrix onto neuromorphic hardware to reliably and robustly realise the desired dynamics without requiring significant optimisation or adaptation to the target hardware. In contrast, deployment of RSNNs to neuromorphic hardware usually requires some degree of chip-in-the-loop training or post-deployment fine-tuning to ensure robustness to hardware-specific nonidealities [6, 12, 13, 38, 39]. This requirement severely limits the ability of neuromorphic hardware to be deployed at scale.

We demonstrate the reliability and generality of our approach by running experiments in three SNN environments. First in simulation, where we show the ideal functioning of the RSNN even with considerably nonideal weights. Second, a closed-loop memristive hardware setup with 64×64 resistive RAM (RRAM) devices acting as the synaptic weights, demonstrating the suitability of the approach for novel beyond-CMOS memristor-integrated neuromorphic hardware. Third, we demonstrate on Intel's neuromorphic research chip Loihi 2 that the approach scales up seamlessly to large state machines.

2.1. SNN simulation demonstrates robustness to noisy weights

The simulated RSNN consists of N leaky integrate-and-fire (LIF) neurons, with all-to-all synaptic connectivity, except between neurons in the same block, which have winner-take-all (WTA) connectivity between themselves (see Methods). This enforces that only one neuron in each block may spike at once, thereby restricting the neural activity to have a sparse block code structure.

Although the method generalises to arbitrary DFAs [22], we chose to implement a 23-state DFA which can perform modular division of binary numbers (figure 1), as it is a clear example of useful computation. Two inputs, s_0 and s_1, with associated random hypervectors $\mathbf{s}_0$ and $\mathbf{s}_1$, are input to the network to represent '0' and '1', respectively. A sequence of inputs thus corresponds to a dividend d in binary representation, and the final state corresponds to the result of the modular division d mod 23. The full state transition function is described by the transitions $q_n \xrightarrow{s_0} q_{(2n \bmod 23)}$ and $q_n \xrightarrow{s_1} q_{(2n+1 \bmod 23)}$ for all states q_n, $n \in \{0, \ldots, 22\}$. The weight matrix to implement this DFA was constructed as described in Methods. We then stochastically binarized the ideal weights and added independently-sampled Gaussian noise of standard deviation 0.5 to each binary weight value, such that the two weight distributions were highly overlapping. This demonstrates robustness to low-precision and noisy weights, inherent in mixed-signal neuromorphic hardware platforms. Despite these nonidealities, the RSNN performs the correct walk between attractor states for every input sequence (figure 1). The network is not reliant on a specific input timing scheme, as long as the inputs are given for long enough for a transition to take place (figure 2). A similar 300-state DFA with ideal weights (figure S3) was constructed in the same way, to demonstrate the seamless scaling to large state machines.

2.2. Memristive crossbar hardware implementation

To demonstrate that the approach is robust to nonidealities introduced by memristive devices, we run a closed-loop experiment with synapses on a memristive crossbar and neurons in simulation (figure 3). Whenever the simulated neurons spiked, the memristive crossbar was read by applying the binary vector of spiking neurons as a voltage input vector to the crossbar via the on-board DACs. The readout currents were read by the ADCs and then fed back into the simulation as postsynaptic currents. With this hybrid 'computer in the loop' approach, we demonstrate the readiness of the algorithm for implementation on emerging hybrid SNN-memristor platforms [40].

We used an RRAM crossbar with 4096 devices. Due to this size constraint, we implemented smaller DFAs than in simulation. The weights were mapped to ternary values by thresholding, and then written to three conductance states on the RRAM devices. Although the device programming procedure in principle supports more weight states—limited by the precision of the ADCs (12-bit) – relaxation effects in the filamentary RRAM cause the conductance values to disperse in the milliseconds to seconds after programming, reducing the effective precision of the weights [41, 42]. After this dispersion, the RRAM devices store the synaptic weights in a non-volatile fashion [43]. We first implemented a DFA representing a 4-state counter. The RSNN performs the correct walk between attractor states, despite the fixed-pattern-noise and trial-to-trial weight nonidealities in the crossbar current readouts (figure 4). We then implemented another 4-state DFA but with two inputs, following an identical procedure. For every input sequence tested, the network performs the correct walk between attractor states (figure S4).

2.3. Large state machines on digital neuromorphic hardware

To demonstrate that the approach scales to large state machines on neuromorphic hardware and can be easily transferred to different types of neuromorphic systems, we implemented the RSNN on Intel's neuromorphic research chip Loihi 2. We implemented a functionally equivalent block-WTA mechanism with shunting inhibition synapses using a custom neuron microcode. This is required to give each neuron both a

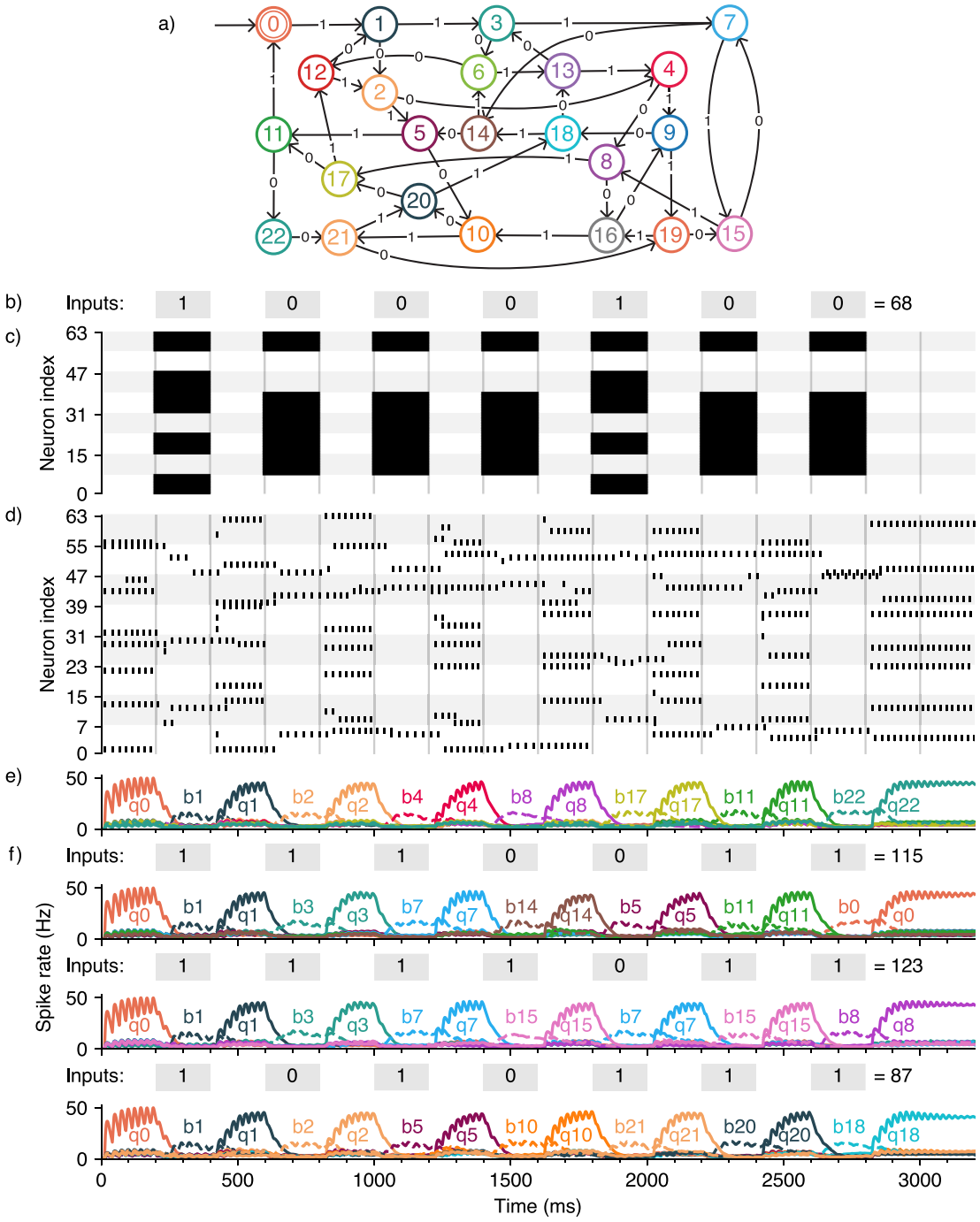

Figure 1. An RSNN performing a walk on a 23-state DFA, using noisy 1-bit weights. (a) The embedded DFA. If a binary number is input most-significant-bit first, the final state indicates the result of the input mod 23. (b) The symbolic input to the RSNN. (c) The input vector to the RSNN at any time, which each masks out half of the neurons therein. Each different input s corresponds to a different hypervector $\mathbf{s}$. Only the first 64 neurons are shown. (d) A spike raster plot of activity within the RSNN. When the input to the network is constant, the network stabilises in an attractor state. When the input changes, some blocks are masked out or revealed, which may cause transitions to a new attractor state. The block-WTA mechanism ensures that only one neuron is persistently active in each block. (e) The kernel-filtered mean firing rate of the neurons active in the attractor states $\mathbf{q}_0, \ldots, \mathbf{q}_{22}$, as well as the transition-facilitating bridge states $\mathbf{b}_0, \ldots, \mathbf{b}_{22}$. The bridge states are represented by dashed lines and coloured the same as their corresponding q attractor states. We here serially input the number 68 in binary format, and the RSNN correctly halts in the q_{22} state. (f) The same RSNN is given different sequences of inputs, and the RSNN performs the correct walk between attractor states in all cases.

conventional integrative synapse and a shunting inhibition synapse. When a shunting synapse is stimulated, it resets the membrane potential of the target neuron to a subthreshold value (figure S5), implementing the within-block WTA and the block-wise masking of the RSNN by input. The integrative synapses implement the recurrent weight matrix as specified in Methods. We implemented the same 23-state DFA as in

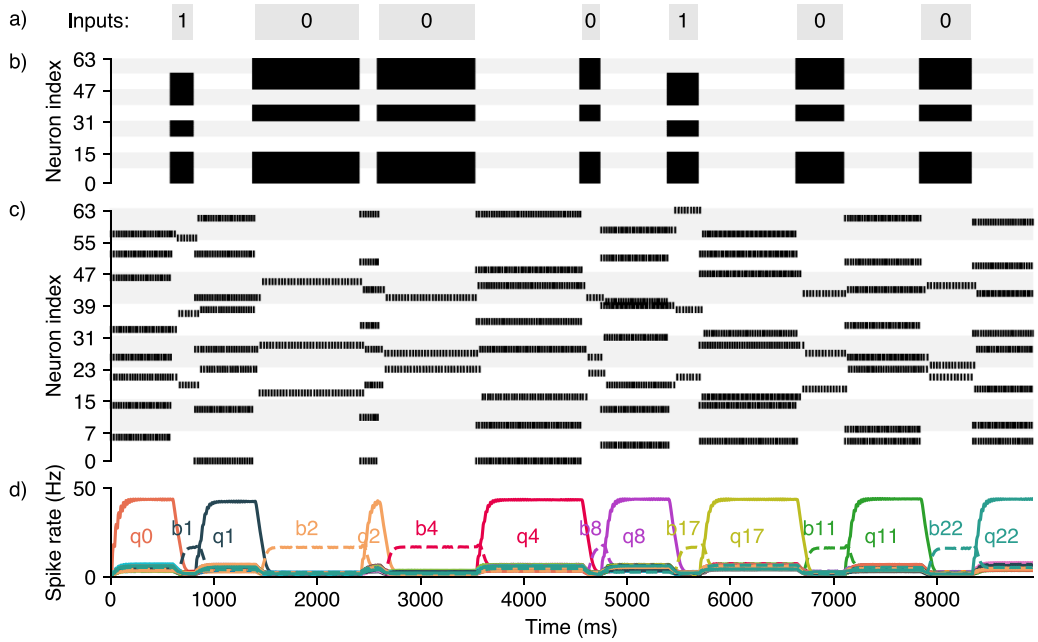

Figure 2. The simulated RSNN with an irregular input timing scheme. (a) The symbolic input to the network. Inputs were given for between 200 and 1000 ms. (b) The corresponding input hypervector masking the neurons. (c) A raster plot of spike activity in the RSNN. (d) The mean firing rate of the neurons in each attractor state. The network is not reliant on an exact input timing scheme to perform the correct sequence of transitions. There is however still a minimum duration for which an input (or lack thereof) must be given, determined by the synaptic time constant.

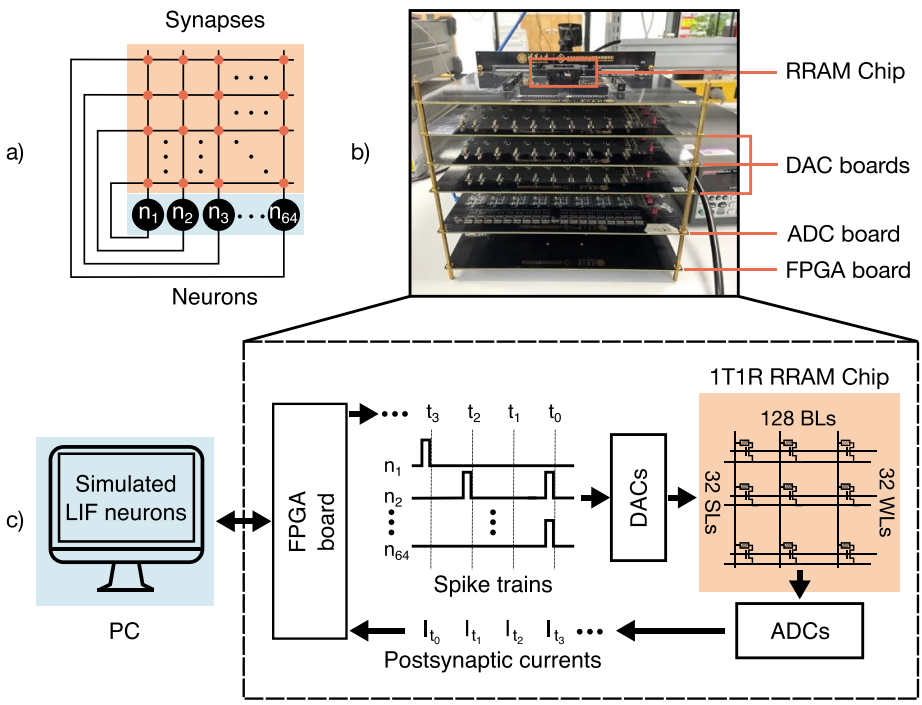

Figure 3. The closed-loop experimental setup for running the RSNN using the 4096-device RRAM system. (a) A simplified schematic of the 64-neuron RSNN. (b) The physical RRAM measurement system, containing an FPGA board, three DAC boards, and one ADC board. (c) The scheme of running the closed-loop experiment in which the LIF neurons are simulated on a PC. The FPGA board receives the control commands from the PC and operates all word-, source- and bit lines (WLs, SLs, BLs) of the RRAM chip in parallel. At each time step (t_n), the spikes generated by the neurons are sent to the synapse array on WLs, which are represented by voltage pulses. Meanwhile, a 0.2 V read voltage is applied to the SLs. The postsynaptic currents (I_{t_n}) produced by the RRAM cells accumulate on each BL, and the readout results are returned to the PC through the ADCs and FPGA.

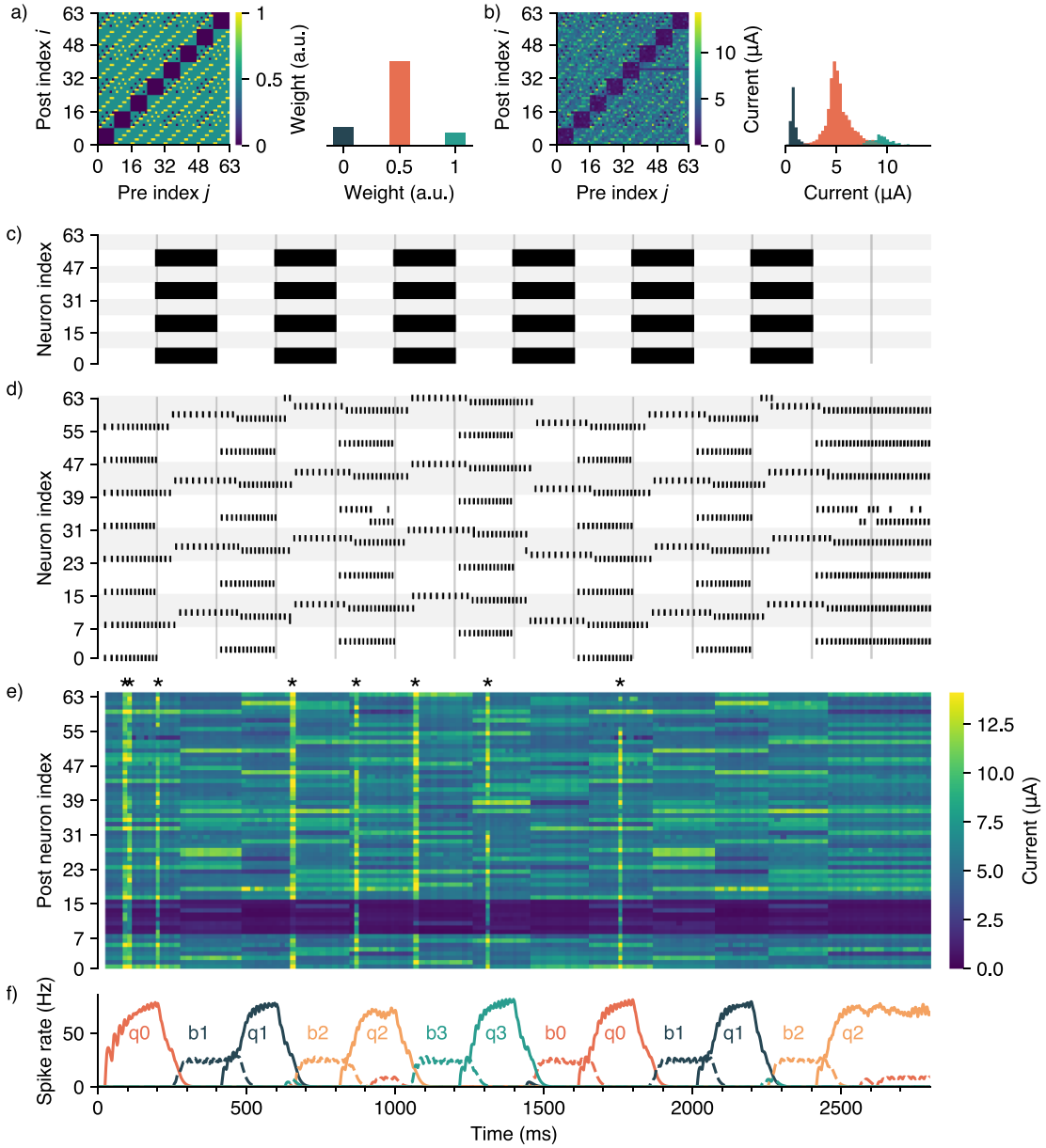

Figure 4. A 4-state DFA embedded into an RSNN using a memristive crossbar with 64×64 RRAM devices as the synaptic weights. The DFA is described by $q_0 \xrightarrow{s} q_1 \xrightarrow{s} q_2 \xrightarrow{s} q_3 \xrightarrow{s} q_0$ for a single input s. (a) The ternary weight matrix to be written to the RRAM crossbar, and a histogram of the values. (b) Readout currents from each of the 64×64 RRAM devices after programming, and separate histograms for each of the ternary weight values. There is notable mismatch between the measured currents and the ideal values; a row of faulty devices giving almost no current; and devices giving anomalously large currents. (c) The masking input to the network. (d) A spike raster plot of the neurons within the RSNN. Due to the size constraints introduced by the crossbar, we chose the attractor hypervectors $\mathbf{q}$ to be orthogonal rather than random. (e) Measured postsynaptic current readings from whenever a neuron in the second block spiked, chosen for the prominence of trial-to-trial current variation. The weights between neurons in the same block were programmed to the lowest weight, hence the horizontal band of low currents. At some times, multiple neurons fired within the same time step (labelled by $*$). (f) The mean firing rates of the neurons in each attractor state. Despite the considerable nonidealities present, the RSNN performs the correct walk between attractor states.

simulation (figure 1), now quantizing the ideal weights to the available 8-bit values. The network performs the correct walk between attractor states reliably for all given input sequences (figure 5).

2.4. Network capacity

Since the RSNNs used in this work are based upon a single-layer attractor network structure, we are limited by the theoretical memory capacity of such networks. That is, for a given network size N and block length L (determining the activation sparsity $1/L$), a maximum number of attractors can be embedded before unwanted cross-talk between patterns leads to a breakdown in attractor dynamics [11]. This limits the maximum size of DFA that can be embedded for a fixed N. For dense Hopfield networks, this results in the well-known linear capacity limit of $P < 0.14N$, where P is the number of storable patterns [10]. For sparse

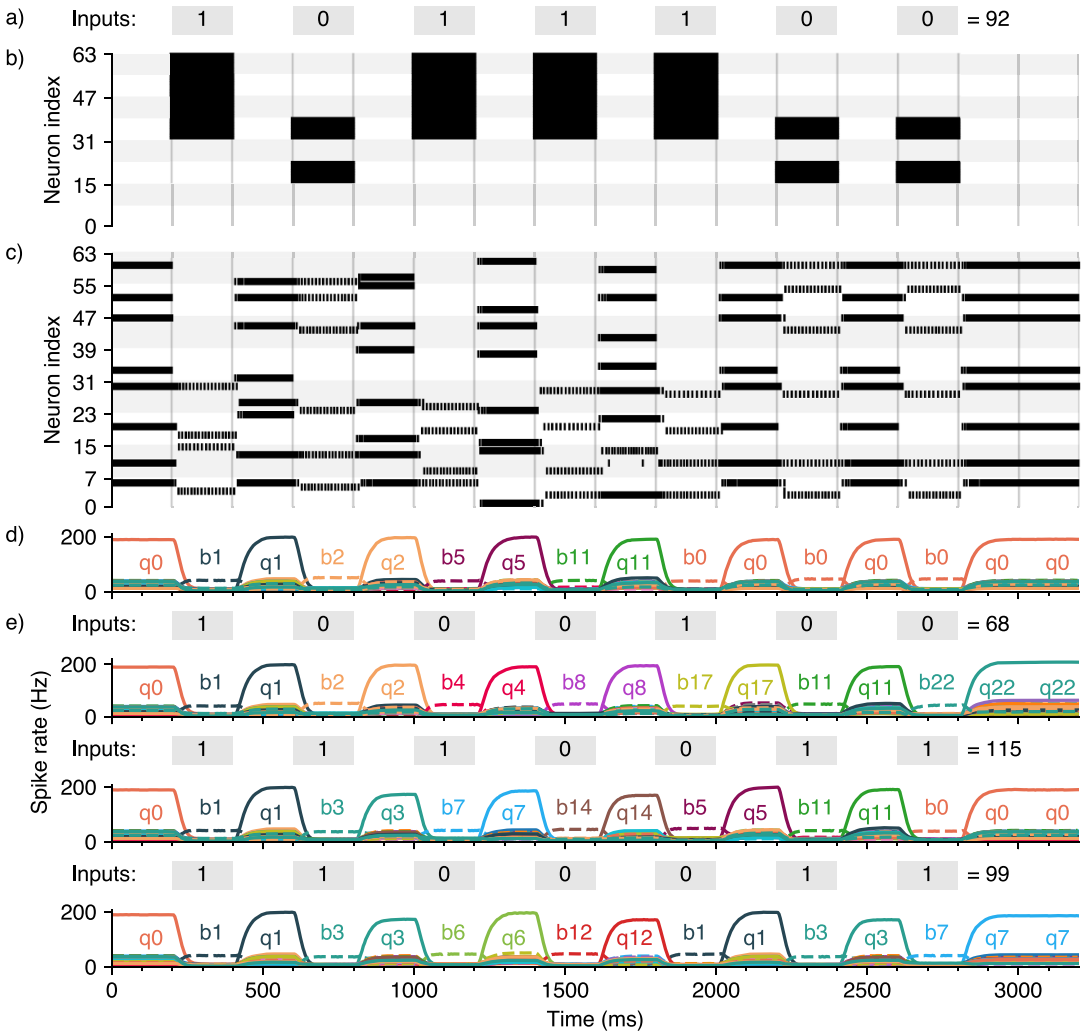

Figure 5. The 23-state DFA on Intel's asynchronous digital neuromorphic research chip, Loihi 2. (a) The input symbol to the network at any time, and (b) the corresponding input vectors, which mask the network activity. (c) A spike raster plot of the first 64 neurons. The shunting-inhibition WTA mechanism ensures that only one neuron in every block may spike at once (see Methods). (d) Kernel firing-rate estimates of each q and b state, choosing arbitrarily that 1 time step on Loihi represents 1 ms. The sequence of inputs given corresponds to the binary representation of the number 92. The RSNN halts in the q_0 state, indicating correctly that 92 is divisible by 23. (e) For all sequences of inputs, the correct walk between attractor states is performed.

attractor networks, the capacity instead scales almost quadratically as $P \propto N^2/(\log N)^2$, provided the sparsity also scales appropriately [29, 30, 33]. We conducted non-spiking simulations to numerically investigate whether these scaling relations also hold with our approach, for both real ideal and stochastically binarised weights. We found that the capacity is proportional to $N^2/(\log N)^2$ in both cases, while the network capacity in the binary weight case is approximately $4\times$ smaller than with ideal weights (figure 6).

3. Discussion

It is widely believed that recurrent attractor dynamics are fundamental to the brain's ability to temporarily represent information in stable neuron firing patterns, without the need for components with intrinsically-long timescales [44], and despite the nonidealities present in biological systems [3, 45–48]. Networks that switch between discrete attractor states are used to explain neural recordings and are thought to underlie higher decision-making processes [28, 49–51]. These robust dynamics inspired the embedding of controlled attractor transition dynamics in RNNs, in particular in neuromorphic systems that mimic the brain's parallelism and asynchrony, but suffer from a similar inherent unreliability of single components as biological systems [52] (see [22] for a more comprehensive overview).

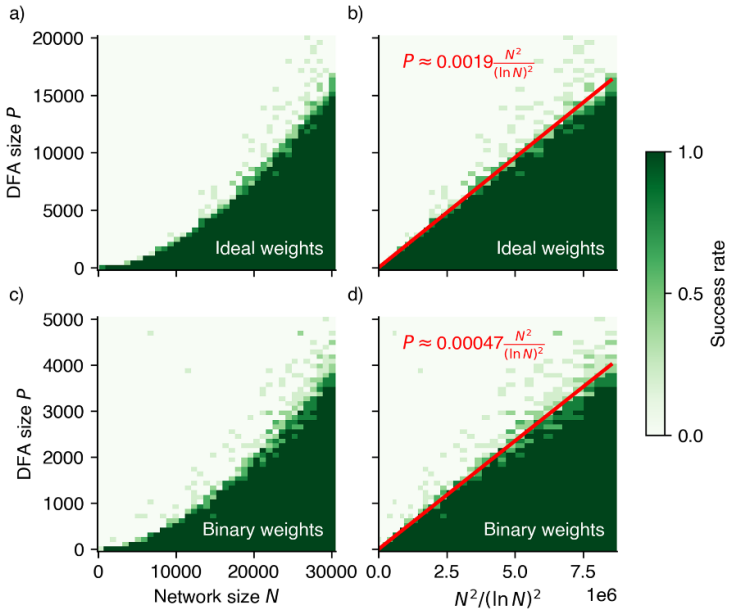

Figure 6. Network capacity for varying network size N, in terms of the number of DFA states P that can reliably be emulated. Each DFA state has two outgoing edges, like in figure 1. (a) A network with ideal weights displays an approximately quadratic capacity relation. (b) The same data, but plotted against $N^2/(\log N)^2$, with a separation boundary obtained by a linear SVM. c,d) A network with stochastic binary weights displays the same scaling relation but reduced by a factor of ≈ 4. In both cases, the block length L was varied such that $L \propto N/\log N$, with reference values $L_0 = 8$, $N_0 = 2048$.

Distributed representations are scalable and robust

In previous efforts to embed attractor-based state machines into spiking neuromorphic hardware [25, 26, 53], each state was represented by a dedicated orthogonal (non-overlapping) population of neurons with a synaptic gating mechanism to trigger state transitions. Although this simplifies the state encoding, these localist state representations limit the robustness, flexibility, and scalability of the network in comparison to a distributed approach [34]. If each state is represented by a unique population of M neurons that are active for only one pattern, then a network of N neurons can represent at most N/M patterns. In contrast, sparse distributed single-layer attractor networks can store $N^2/(\log N)^2$ patterns, making more efficient use of the available synaptic resources [29, 30, 33]. Additionally, adding new states with localist representations requires adding or finding new neurons that are inactive for every other pattern. This is at odds with the mixed selectivity of neurons found throughout the brain [54].

We instead utilised the distributed representational framework of VSAs, in which high-dimensional random vectors can be composed to represent arbitrary data structures and algorithms [17]. The state-representing hypervectors thus have a nonzero probabilistic degree of overlap, but it is nonetheless possible to generate an exponential (in N) number of hypervectors for which significant overlap between any pair of hypervectors is unlikely (section S1) [55, 56]. The sparsity of the hypervectors not only increases the network's memory capacity, but makes them inherently suitable for exploiting the efficiency of sparse activity in neuromorphic hardware [57].

The distributed representations also make the network highly robust to various nonidealities, such as noise, individual component failure and nonideal neuron dynamics. Their robustness to low-precision (quantized) and inaccurate synaptic weights is particularly relevant for memristive in-memory applications. This has prompted recent interest in combining VSAs with memristive crossbars [21, 58] and spiking neural networks [23, 59–61], where synaptic and neural nonidealities are abundant. The results obtained from simulations and the closed-loop memristive crossbar setup confirm the system's reliable operation under hardware constraints, notably without requiring parameter fine-tuning, while the restriction of binary weights causes only a constant factor decrease in the network capacity.

Simultaneous auto- and heteroassociative memory

By exploiting the pseudo-orthogonality property of the hypervector binding operation, we were able to superimpose autoassociative and heteroassociative outer-product terms in the recurrent weight matrix without incurring considerable interference between them. Transitions are triggered by masking a subset of the neurons—constituting an effective unbinding operation—while in previous work, additional mechanisms were required to modulate between attractor and transition dynamical regimes [62–69].

Furthermore, the simplicity of this construction method means that the weight matrix could reasonably be learned via local Hebbian learning rules, such that states and transitions are learned and unlearned as needed to solve a particular task [70].

Fault-tolerant computation

Programming a certain DFA into an RSNN in one shot may be sufficient for some applications, however. In space, ionizing solar radiation causes unwanted bit-flips in systems using conventional charge-based memories, and so necessitates fault-tolerant computation schemes. Our system is intrinsically robust to such events since the attractor dynamics quickly reverse any erroneous flips in the neural state. Furthermore, RRAM-based memories are highly resistant to solar radiation, making memristive VSA-programmed attractor-based neuromorphic implementations of conventional algorithms potentially suitable for space and other high-energy applications [71].

The flexibility and robustness of the model enable a wide range of applications for neuromorphic hardware. It could be used to coordinate information flow in complex neuromorphic systems [53] or be directly incorporated into existing works leveraging VSA representations [72], such as online learning of robotics schemas [73] or general computation with distributed stack machines [16, 74].

Neuromorphic hardware abstraction

By utilizing distributed symbolic representations, we programmed an RSNN from an abstract higher level of description, which is separated from the exact details of the hardware. It relied on the existence of a fixed stereotypical low-level neural motif (the block-WTA connectivity), but adjusted only the connections between neurons in different blocks, to easily program the same neural hardware for different purposes. This ability to abstract the function of lower layers and then build upon them relatively carefree is the foundation of conventional digital design approaches and bears resemblance to Marr's levels of analysis [16, 75].

The distributed representations of VSAs are especially suited as an abstraction layer to divorce high-level function from low-level neural dynamics, in part due to their invariance of the choice of hypervector representation [16, 17]. For example, in place of sparse block code representations, we could have instead used phasor representations [19, 76] or their sparse variants, which have an elegant connection to the periodic spiking of resonate-and-fire neurons [23, 59]. The same algorithm could be implemented on different neuromorphic hardware platforms, using whichever representation is best suited for the available neuron circuit. Neuromorphic computing would greatly benefit from a high-level neurally-inspired abstract programming language, which can be robustly implemented across different neuromorphic hardware platforms [16, 77, 78]. Although there have been many works towards such a unification [79–84], the majority of these focus on finding common ways to interface with different neuromorphic hardware platforms or to compose functional SNN modules, rather than on defining higher-level abstract primitives that are invariant of the neural representations on the hardware on which they are embedded.

This work demonstrates the capability of distributed representations for embedding computation through neural dynamics into neuromorphic hardware [16]. We embedded arbitrary state machines into RSNNs, as they are a clear example of multi-timescale computation: the switching between states is fast, but the autoassociative attractor dynamics ensure stability on long timescales [10, 11]. The extension of this approach to embedding general computational primitives beyond DFAs into neuromorphic RSNNs, such as continuous attractor networks [3], presents an exciting direction for future research. The fully-distributed representations ensure increasing robustness with increasing dimensionality, while the invariance to the choice of hypervector representation—many of which have elegant spiking implementations—may allow neuromorphic hardware to be programmed at a level that is independent of the underlying neural representations, but that can reliably and robustly be implemented on different neuromorphic hardware platforms without the need for fine-tuning.

4. Methods

4.1. VSA arithmetic

VSAs allow compositional data structures to be represented in a fully distributed manner, using high-dimensional random vectors, hypervectors, as the smallest units of representation [17–20]. They rely upon the fact that independently-generated hypervectors will very likely be approximately orthogonal to each other (the *pseudo-orthogonality* property) and so new hypervectors can easily be generated to represent new items. For most hypervector representations, the similarity between hypervectors is defined by the normalised inner product

$$\mathrm{sim}\left(\mathbf{a},\mathbf{b}\right)=\frac{\mathbf{a}\cdot\mathbf{b}}{||\mathbf{a}||_2||\mathbf{b}||_2} \tag{1}$$

such that, for a set of independently-generated hypervectors $\{\mathbf{x}_\nu\}$ the pseudo-orthogonality property is then written as

$$\mathrm{sim}\left(\mathbf{x}_\nu, \mathbf{x}_\mu\right) \begin{cases} = 1 & \text{if} \quad \mu = \nu \\ \approx 0 & \text{otherwise} \end{cases} . \tag{2}$$

Each VSA model is equipped with two operators to generate hypervectors which represent relations between other hypervectors, and thus also the symbols they represent. They are the superposition and binding operators, often corresponding simply to component-wise addition and multiplication respectively. The superposition operator produces a hypervector of the same dimensionality, which retains similarity to its operands. That is, if $\mathbf{a}$ and $\mathbf{b}$ are independent hypervectors, then $\mathrm{sim}(\mathbf{a} \oplus \mathbf{b}, \mathbf{a}) \approx \mathrm{sim}(\mathbf{a} \oplus \mathbf{b}, \mathbf{b}) \approx 1$ where '$\oplus$' represents a general superposition operator. Superpositions of hypervectors are thus used to efficiently represent sets of objects [55, 56].

The binding operator produces a hypervector that is dissimilar to both its operands. That is, if $\mathbf{a}$ and $\mathbf{b}$ are again independently-generated, then $\mathrm{sim}(\mathbf{a} \odot \mathbf{b}, \mathbf{a}) \approx \mathrm{sim}(\mathbf{a} \odot \mathbf{b}, \mathbf{b}) \approx 0$ where '$\odot$' is a general binding operation. Binding may be inverted by unbinding the same hypervector, i.e. $(\mathbf{a} \odot \mathbf{b}) \odot^{-1} \mathbf{b} = \mathbf{a}$. Superimposing pairs of bound hypervectors could then be used to create a hypervector representing a set of key-value pairs [18], for example, but more complex objects can similarly be represented such as graphs [17, 85], algorithms [74] or images [86].

In this work, we used two types of VSA models: dense bipolar hypervectors, also known as 'Multiply-Add-Permute' (MAP) hypervectors, and sparse block code (SBC) hypervectors [17]. MAP hypervectors are randomly sampled from $\{-1, 1\}^N$, with approximately equal numbers of positive and negative entries. The binding operation for MAP hypervectors is the Hadamard product '$\circ$' (element-wise multiplication), which also serves as the inverse binding operation [20]. SBC hypervectors are binary-valued and have a sparse-block structure. That is, the N components are divided into M equally-sized blocks of length $L = N/M$, and we randomly choose exactly one component in each block to be 1-valued, with the rest 0 [31, 32]. By binding MAP and SBC hypervectors with a Hadamard product operation, we produce a bipolar sparse hypervector that is pseudo-orthogonal to the original SBC hypervector (see section S4 for further discussion on the joint MAP-SBC VSA model). We can then superimpose weight matrices constructed using these hypervectors, each corresponding to a different network behaviour, without significant interference. Masking out some components constitutes an effective Hadamard unbinding operation. This operation allows that terms in the weight matrix bound to a MAP hypervector (and thus pseudo-orthogonal to all SBC hypervectors) may be unbound and align with the network state. As a result, state transitions in the RSNN can be triggered by selectively masking neurons with inhibitory input.

4.2. State machine embedding

A DFA is a finite state machine consisting of a set of states Q, an initial state $q_0 \in Q$, a finite set of input symbols S, and a state transition function $F : Q \times S \to Q$. In addition, a DFA denotes a subset $A \subseteq Q$ of its states as 'accepting' states. A string of input symbols is then said to have been accepted by the DFA if it terminates in one of the acceptance states. As such, DFAs can be used to perform pattern matching—accepting or rejecting strings based on a predefined set of rules, regular expressions [87].

The DFA is embedded into the RSNN by first generating for each state $q \in Q$ an SBC hypervector $\mathbf{q} \in \{0, 1\}^N$ to represent it. For every input symbol $s \in S$, a dense binary hypervector $\mathbf{s} \in \{0, 1\}^N$ is generated, which, when input to the network, should trigger the correct transitions between attractor states. We restrict these hypervectors to be block-constant, with the same block length L as the SBC hypervectors, such that components in the same block have the same value. The 0 and 1 entries were chosen with equal probability. We will denote $\bar{\mathbf{s}} \in \{-1, 1\}^N$ as the equivalent bipolar representation of these hypervectors, given by $\bar{\mathbf{s}} = 2\mathbf{s} - 1$. For every state q, an additional SBC hypervector $\mathbf{b}$ is generated, which we refer to as its *bridge* state. They will be attractors of the RSNN dynamics only when there is input to the network. Transitions will be constructed such that, when we give input to the RSNN, it will not transition directly from a source state $\mathbf{q}$ to target state $\mathbf{q}'$, but rather will first transition from $\mathbf{q}$ to $\mathbf{b}'$, the bridge state belonging to the target state $\mathbf{q}'$. When the input is removed, the RSNN will finally flow from the $\mathbf{b}'$ to $\mathbf{q}'$ state, completing the transition. Splitting each transition up in this way ensures they are not dependent upon a specific input timing scheme (figure 2). The construction of the recurrent weight matrix $\mathbf{W} \in R^{N \times N}$ can be divided into three parts, first to embed the $\mathbf{q}$ states as attractors

$$\mathbf{W}_{\text{attr.}} = \sum_{q \in Q} (\mathbf{q} - f)(\mathbf{q} - f)^\intercal \tag{3}$$

where the coding level $f = 1/L$ is the fraction of nonzero elements in each $\mathbf{q}$ hypervector [29, 30]. We then embed the bridge states as

$$\mathbf{W}_{\text{brdg.}} = \sum_{q \in Q} \left[(\mathbf{q} - f)(\mathbf{b} - f)^{\mathsf{T}} + \sum_{s \in S} (\mathbf{b} - \mathbf{q})((\mathbf{b} - f) \circ \bar{\mathbf{s}})^{\mathsf{T}} \right] \tag{4}$$

where '$\circ$' is the Hadamard product binding operation. We are thus using an unorthodox mixture of the MAP [17, 20] and SBC [31, 32] VSA models (section S4). These terms cause the bridge states $\mathbf{b}$ flow to their corresponding $\mathbf{q}$ states, except when a valid input is present, then they themselves become attractors of the RSNN's dynamics. To store transitions between states, outer product terms are then similarly added for each non-loop transition, the set of which we define as $E = \{(q, s, q') \text{ s.t. } q' = F(q, s),\ q' \neq q\}$. They are included as

$$\mathbf{W}_{\text{trans.}} = \sum_{\eta \in E} (\mathbf{b}' - \mathbf{q})((\mathbf{q} - f) \circ \bar{\mathbf{s}})^{\mathsf{T}} \tag{5}$$

where $\mathbf{q}$ is the attractor state for source state q for edge η, and $\mathbf{b}'$ the bridge state belonging to the target state $q' = F(q, s)$, as defined by E. The full weight matrix is then given by the superposition $\mathbf{W} = \mathbf{W}_{\text{attr.}} + \mathbf{W}_{\text{brdg.}} + \mathbf{W}_{\text{trans.}}$.

4.3. RSNN attractor dynamics

To understand how the described weight construction method results in the desired DFA being emulated, we coarsely model the RSNN by a neural state vector $\mathbf{z} \in \{0, 1\}^N$, indicating which neuron is active. We then consider the postsynaptic sum $\mathbf{h} = \mathbf{W}\mathbf{z}$, with and without input to the RSNN. It is instructive to work with the expectation of $\mathbf{h}$, where terms containing inner products between pseudo-orthogonal hypervectors can be discarded. We must, however, keep in mind that for finite N, the postsynaptic sum will only approximate its expected value, due to the accumulation of nonzero inner products between realised hypervectors. See [55, 56] for a rigorous theoretical analysis. We first consider the dynamics when the network is in a DFA state, $\mathbf{z} = \mathbf{q}^\mu$. The expected postsynaptic sum $\langle \mathbf{h} \rangle$ is then given by

$$\begin{aligned} \langle \mathbf{h} \rangle &= \langle \mathbf{W}\mathbf{q}^\mu \rangle \\ &= \langle \mathbf{W}_{\text{attr.}}\mathbf{q}^\mu \rangle + \langle \mathbf{W}_{\text{brdg.}}\mathbf{q}^\mu \rangle + \langle \mathbf{W}_{\text{trans.}}\mathbf{q}^\mu \rangle \\ &\propto (\mathbf{q}^\mu - f) + \mathbf{0} + \mathbf{0} \\ &= \mathbf{q}^\mu - f \end{aligned} \tag{6}$$

such that the state $\mathbf{q}^\mu$ projects back to itself (section S2). Neither $\mathbf{W}_{\text{brdge.}}$ nor $\mathbf{W}_{\text{trans.}}$ contribute to $\langle \mathbf{h} \rangle$, since they are constructed either from independent SBC hypervectors, or the same SBC hypervector $\mathbf{q}^\mu$ but bound to a bipolar hypervector $\bar{\mathbf{s}}$, both of which are pseudo-orthogonal to the state $\mathbf{q}^\mu$. Hence, in the large-N limit, and sufficiently close to the states $\mathbf{q}$, we can approximate the dynamics as being governed only by the $\mathbf{W}_{\text{attr.}}$ matrix. Since $\mathbf{W}_{\text{attr.}}$ is symmetric, this results in fixed-point attractor dynamics with an energy landscape consisting of stable minima at the states $\mathbf{q}$ (section S3) [29, 30, 88].

4.4. RSNN transition dynamics

We now consider how $\langle \mathbf{h} \rangle$ changes when we apply an input $s^\lambda \in S$ to the RSNN, which we assume corresponds to a valid transition $\eta = (q^\mu, s^\lambda, q^{\mu\prime}) \in E$ from state q^μ to $q^{\mu\prime}$. Inputting s^λ means applying the corresponding hypervector $\mathbf{s}^\lambda \in \{0, 1\}^N$ as a mask to the RSNN, such that all neurons in the blocks for which $\mathbf{s}^\lambda$ is zero are prevented from spiking. The expected postsynaptic sum is then given by

$$\begin{aligned} \langle \mathbf{h} \rangle &= \langle \mathbf{W}(\mathbf{q}^\mu \wedge \mathbf{s}^\lambda) \rangle \\ &= \langle \mathbf{W}_{\text{attr.}}(\mathbf{q}^\mu \wedge \mathbf{s}^\lambda) \rangle + \langle \mathbf{W}_{\text{brdg.}}(\mathbf{q}^\mu \wedge \mathbf{s}^\lambda) \rangle + \langle \mathbf{W}_{\text{trans.}}(\mathbf{q}^\mu \wedge \mathbf{s}^\lambda) \rangle \\ &\propto (\mathbf{q}^\mu - f) + \mathbf{0} + (\mathbf{b}^{\mu\prime} - \mathbf{q}^\mu) \\ &= \mathbf{b}^{\mu\prime} - f \end{aligned} \tag{7}$$

where $\wedge$ is a component-wise AND, implementing the masking operation (section S2). Since the masked $\mathbf{q}^\mu$ state now does not project back to itself, but to $\mathbf{b}^{\mu\prime}$ (the bridge state belonging to the target $q^{\mu\prime}$ state), the RSNN transitions to the $\mathbf{b}^{\mu\prime}$ state. By applying $\mathbf{s}^\lambda$ as a mask to the RSNN, we caused heteroassociative terms to be projected out of the edge summation in $\mathbf{W}_{\text{trans.}}$, which push the RSNN from $\mathbf{q}^\mu$ to $\mathbf{b}^{\mu\prime}$.

A similar calculation can be done to show that the bridge states $\mathbf{b}$ are attractors only when the network is being masked by a valid input, i.e. $\langle \mathbf{W}(\mathbf{b}^{\mu\prime} \wedge \mathbf{s}) \rangle = \mathbf{b}^{\mu\prime} - f$. When the input is removed, however, this

becomes $\langle \mathbf{W}\mathbf{b}^{\mu\prime} \rangle = \mathbf{q}^{\mu\prime} - f$, driving the RSNN towards the target state, completing the transition. By applying a sequence of inputs to the RSNN as masks (with pauses in between), we cause the RSNN to perform the desired sequence of transitions between attractor states. See figure S6 for a depiction of how this unfolds at the neural level of description. Including the bridge states in the state machine construction ensures that transitions are not dependent upon a specific input timing scheme, since an input must be given—and crucially then removed—to complete a transition (figure 2) [22]. If an input s were given that did not correspond to a valid transition for the current state q, no transition occurs.

Constructing attractor networks with transition dynamics by superimposing symmetric autoassociative terms and asymmetric heteroassociative terms, is not a new idea [62]. In previous works, to ensure that neither attractor nor transition dynamics dominated always, one had to either ensure a fine balance between the relative strengths of the two terms [63, 64, 89] or to introduce additional mechanisms to modulate their relative strengths, such as delayed synapses [65–67, 90] or short-term synaptic plasticity [68, 69]. By exploiting the properties of VSAs, we have achieved the same result without such mechanisms and without sacrificing robustness. By adding to $\mathbf{W}$ latent heteroassociative terms which were each bound to a bipolar hypervector $\bar{\mathbf{s}}$, in the absence of input, we could ignore their contribution to the RSNN dynamics, due to them being pseudo-orthogonal to the network state (equation (6)). However, when we mask the RSNN with the same $\mathbf{s}$—an effective unbinding operation—these heteroassociative terms are revealed, and trigger the transition dynamics (equation (7)). Crucially, this is achieved without ever altering the synaptic weight matrix $\mathbf{W}$ during operation. We use the masking mechanism to realise an unbinding operation because it is not reliant on the synchronous arrival of inputs [22].

4.5. Spiking neuron model

We simulate LIF neurons, defined by

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = -\frac{1}{\tau_\mathrm{m}}\left(u_i - u_\mathrm{rest}\right) + \frac{1}{C} I_i(t) \tag{8}$$

where u_i is dynamical membrane voltage of the neuron i, τ_m and u_rest are the membrane time constant and rest potential respectively, I_i is the postsynaptic current to that neuron, and $C = 1\mathrm{F}$. When u exceeds the spiking threshold u_θ, the neuron emits a spike and u is quickly clamped to a subthreshold reset voltage (here 0 mV) for an amount of time determined by the refractory period τ_ref. The postsynaptic currents $I_i(t)$ we modelled as second-order low-pass filters over spike inputs,

$$\begin{aligned} \tau_\mathrm{syn}\frac{\mathrm{d}I_i}{\mathrm{d}t} &= -I_i + J_i(t) \\ \tau_\mathrm{syn}\frac{\mathrm{d}J_i}{\mathrm{d}t} &= -J_i + \sum_{j=1}^{N} w_{ij} \sum_{\mathrm{spks}\ k} \delta\left(t - t_j^k\right) \end{aligned} \tag{9}$$

where we introduce the synaptic time constant τ_syn, the Dirac delta function $\delta(\cdot)$, and we sum over the spike times t_j^k from all presynaptic neurons j, with synaptic weight w_{ij} measured in coulombs. This has solution

$$\begin{aligned} I_i(t) &= \sum_{j=1}^{N} w_{ij} \left(K(t) \circledast \sum_k \delta\left(t - t_j^k\right) \right)(t) \\ &= \left[\mathbf{W}\mathbf{z}(t)\right]_i \end{aligned} \tag{10}$$

where $K(t)$ is an exponentially-decaying alpha kernel with time constant τ_syn, $\circledast$ indicates temporal convolution, and $\mathbf{z}(t)$ defines the kernel-filtered spike activity. The current is continuous everywhere (in contrast to the more conventional first-order temporal filters [91]), which ensures that I_i is not affected by the precise timing of presynaptic spikes [92]. In the Loihi implementation, only a first-order synapse was used, so this is not strictly necessary, however.

To stabilise the network activity and ensure it always represents an SBC hypervector, we included local WTA connectivity between all neurons in the same block, as often done to enforce constraints in RSNNs [93]. It is implemented such that if any neuron spikes, then all neurons in the same block are forced into a refractory state (figure 7). Each block thus performs an argmax neural activation function on the vector of postsynaptic currents $\mathbf{I}(t) = \mathbf{W}\mathbf{z}(t)$ integrated between spike times. This behaviour is easily implementable in analog VLSI [94, 95]. The same mechanism is used to mask out a subset of neurons within the RSNN with external input. The firing rates $\mathbf{z}(t)$ and postsynaptic currents $\mathbf{I}(t)$ approximate the discrete dynamics of the network state $\mathbf{z}$ and the postsynaptic sum $\mathbf{h}$ respectively, as described in section 4.3.

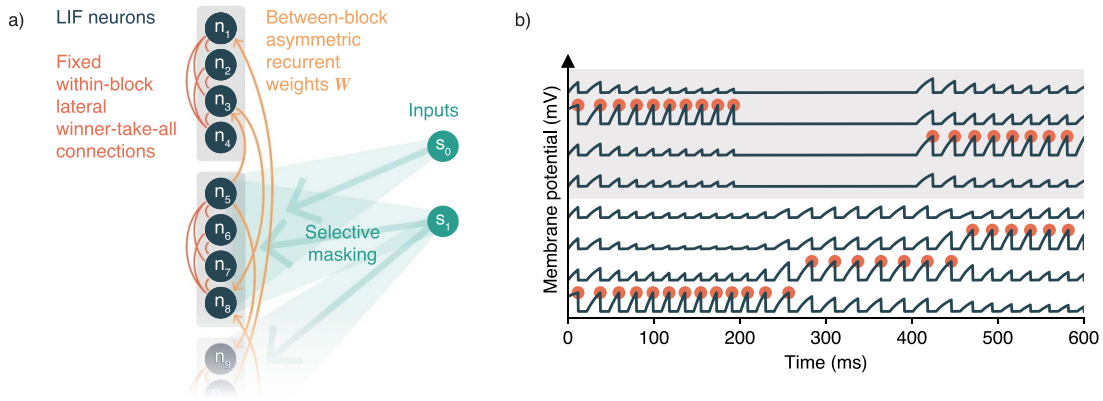

Figure 7. (a) RSNN network diagram. The neurons are split into blocks of equal length (here $L = 4$) with WTA connectivity between all neurons in the same block. The between-block weights **W** are programmed to embed the desired DFA in the RSNN dynamics. The connectivity from each input s is set according to its binary hypervector **s**. (b) Membrane potential traces $u_i(t)$ of the first 8 neurons within an RSNN. Whenever a neuron spikes (red dots), all neurons within the same block are forced into a refractory state. Thus only the neuron with the greatest postsynaptic current spikes, realising an argmax function in each block and confining the neural activity to sparse block codes. Between 200 and 400 ms, the second block is masked by an input (either s_0 or s_1), which causes the network to switch between attractor states.

4.6. Spiking simulation

The neuron equations were simulated in the *Brian 2* SNN simulator [96], using the Euler ODE solver with a default time step of 0.05 ms. The weight matrix to embed the chosen DFA was constructed as described in Methods, with $N = 2048$ neurons and blocks of length $L = 8$. These ideal weights were transformed to binary values, to match that biological synapses may, and neuromorphic synapses often do have very few bits of precision [97]. This binarisation was performed stochastically, via

$$w_{ij}^{\text{binary}} = \begin{cases} 1 \text{ w.p. } \left(1 + \exp\left[-\beta\left(w_{ij} - \langle w\rangle\right)/\sigma_w\right]\right)^{-1} \\ 0 \text{ otherwise} \end{cases} \tag{11}$$

where $\langle w\rangle$ and σ_w are the mean and standard deviation of the ideal weight values respectively, and the steepness parameter β is heuristically set to $\beta = 2$. This is a common way to embed analog values into binary-valued memristive devices [98]. We then add noise to each weight value, to emulate that the distribution of the binary weight states may themselves be imperfect and even overlapping. The nonideal weights used in simulation are then given by

$$w_{ij}^{\text{nonideal}} = |w_{ij}^{\text{binary}} + \chi_{ij}| \tag{12}$$

where χ_{ij} are independent samples from a centered Gaussian distribution with standard deviation 0.5. Histograms of the ideal; binary; and nonideal weights are shown in figure S7. It is well known that attractor networks are robust to such nonidealities and that their effect is to reduce the number of attractors that can be reliably stored [11, 33, 99, 100].

The RSNN is initialised in the q_0 state, and then a sequence of inputs s_0 or s_1 are given to the network for 200 ms at a time, with gaps in-between. We used this particular timing scheme for simplicity, but the proper functioning of the RSNN is not dependent upon the exact timing of inputs. Each input (and gap) may be given for arbitrarily long periods (figure 2), needing only to be longer than the attractor-switching time, which is of the order τ_{syn}.

The firing rate of each attractor state q is given by the dot product of its hypervector **q** and the vector of kernel-filtered spike activity, i.e.

$$m_q(t) = \frac{1}{M}\sum_{i=1}^{N}[\mathbf{q}]_i \left(K(t) \circledast \sum_{\text{spks } k}\delta\left(t - t_i^k\right)\right)(t) \tag{13}$$

where $m_q(t)$ is the firing rate of state q, and $K(t)$ is a normalised alpha kernel with time constant τ_{readout}. If the RSNN is exactly representing a state q, then $\langle m_q\rangle = \langle\nu\rangle$ where $\langle\nu\rangle$ is the average firing rate of the active neurons. Since the **q** states are not orthogonal, the expected firing rate of the other $q' \neq q$ attractor states $\langle m_{q'}\rangle = \langle\nu\rangle/L$ is nonzero. The firing rates of the bridge states **b** are likewise calculated.

4.7. RRAM crossbar experiment

The neuron equations were simulated in Python using Euler's method with the NumPy package. At each time step, if one or multiple neurons spiked, the memristive crossbar was read, and the measured currents were linearly scaled and then applied to the postsynaptic neurons in simulation. The experimental measurements were made using non-volatile 1T1R HfO_x-based RRAM chips integrating 32×128 devices. The hardware setup is shown in figure 3. An FPGA board in the measurement system supports the parallel operation of the WLs, BLs, and SLs of the RRAM chip.

With the 4096 RRAM devices, we could create a fully-connected RSNN with 64 neurons. In simulation and on Loihi, the dimensionality was large enough that we could rely upon the pseudo-orthogonality of randomly-generated hypervector representations. This is not reliably achieved for $N = 64$, and so we instead manually chose our sparse block code hypervectors $\mathbf{q}$ and $\mathbf{b}$ to be orthogonal. The ideal weight matrix $\mathbf{W}$ was constructed as described in Methods but with the f terms set to $f = 0$, to compensate for the fact that our attractor state vectors now have exactly 0 inner product. Since the RRAM devices supported ternary weight states (normalised to $0, 0.5, 1$), we mapped the ideal weights to ternary weights by applying two thresholds on either side of 0. The ternary weights were then written to the RRAM crossbar by a write-verify iterative programming scheme. The distributions of these ternary conductance states are shown in figure 4.

To store the 64×64 weight matrix on the physical RRAM chip, we partitioned it into two 32×64 matrices and then stored their concatenation on the available 32×128 devices (figure S8). To perform a current readout (whenever a neuron spiked), we input the binary vector of the first 32 neurons as a voltage via the DACs in parallel, and read the first 64 currents. Then, we input the binary vector of the latter 32 neurons' spike activity and added the two current vectors together on the PC. Therefore, all multiply-and-accumulate steps besides the last addition were performed in-memory.

4.8. Loihi implementation

The experiments were run on the Intel Loihi 2 Oheo Gulch research system using 1024 neurons (128 blocks of 8 neurons) on 95 neurocores. The code was written in Python using Intel's Lava package.

The network uses the regular current-based leaky integrate and fire neuron model on Loihi. However, different from the hard-coded model, each neuron has a conventional integrative synapse, with weights set by $\mathbf{W}$, and a shunting inhibitory synapse to implement the block-WTA and masking behaviour. To achieve the shunting inhibition synapse, we implemented the neuron using custom neuron microcode, a novel feature of Loihi 2. The microcode on Loihi 2 allows arbitrary instructions from a limited instruction set. To avoid having to create a neuron with 2 registers to store the input, we discriminate the integrative synapse's input and the shunting synapse's input by different weight ranges. The shunting synapse's weight was set to -2048, a value that is never reached by integrating regular input. So, in the microcode, before the regular neuron behaviour is computed, a comparison is made if the input value is below 1000. If it is, the membrane potential is reset to 0 (shunting) and 2048 is added to the input in order not to interfere with the integrative synapse.

The weight matrix was constructed as described in section 4.2, and then quantized by linearly mapping the 4-sigma range of $\mathbf{W}$ to the available even integer weight values in the interval $[-254, 254]$. These recurrent weights were then written to the all-to-all recurrent synapse weights. Attractor transitions were triggered by stimulating the shunting inhibition synapses, with the input vectors $\mathbf{s}_0$ or $\mathbf{s}_1$ to input a 0 or 1, respectively. At the start of the experiment, the RSNN is initialised in the first attractor state q_0. Inputs are then given for 200 time steps each, with a 200 time step gap in between, wherein no input is given.

4.9. Capacity simulation

The RSNN was modelled as a single-layer fully-connected artificial RNN, with the discrete-time update rule

$$\mathbf{z}_{t+1} = \mathrm{bWTA}\left[\mathbf{W} \cdot (\mathbf{z}_t \wedge \mathbf{i}_t)\right] \tag{14}$$

where $\mathbf{z}_t \in \{0,1\}^N$ is a sparse block vector representing the network state on time step t, $\mathbf{W}$ is the recurrent weight matrix (see section 4.2), bWTA is the block-WTA activation function, and $\mathbf{i}_t \in \{0,1\}^N$ is the masking input to the network. If there is no input to the network, then $\mathbf{i}_t$ is a vector of all ones.

The network was tested with modular-division DFAs of varying sizes P, like in figure 1 for $P = 23$. Note that P does not directly correspond to the number of stored attractor states, since each DFA state actually stores two attractor states and, for these DFAs, two asymmetric transition terms. For various (P, N) pairs, we evaluated the network's ability to correctly emulate the DFA by giving sequences of 5 inputs (each lasting 10 time steps, with 10 time steps of no input between) to the network. A run was deemed to be successful if the hypervector $\mathbf{q}$ corresponding to the correct final DFA state had the greatest overlap with the network's final state. The block length L was varied such that the active number of neurons N/L scaled with $\log N$, as

required to obtain the quadratic capacity relation [33]. Each (P, N) pair was tested 5 times with different hypervector instantiations.

Data availability statement

All data that support the findings of this study are included within the article (and any supplementary files).

Acknowledgment

Thanks to Friedrich Sommer, Denis Kleyko, Chris Kymn and Anthony Thomas for enlightening discussions and suggestions. This work has been supported by DFG projects NMVAC (432009531) and MemTDE (441959088). The authors would like to acknowledge the financial support of the CogniGron research center and the Ubbo Emmius Funds (Univ. of Groningen). A R acknowledges funding by VolkswagenStiftung (CLAM 9C854). We thank the Center for Information Technology of the University of Groningen for their support and for providing access to the Hábrók high-performance computing cluster. We thank Intel for providing access to the Loihi 2 research platform and Philipp Stratmann for support with the microcode.

Author contributions

M C conceived the initial algorithm and its SNN implementation. H G, A R implemented the algorithm on Loihi. J C, M C implemented the algorithm on the memristive crossbar setup, supported by H W. E N, H W, G I, M Z, E C supervised the experiments. All authors contributed to writing the manuscript.

Conflict of interest

The authors declare no competing interests.

ORCID iDs

Madison Cotteret https://orcid.org/0000-0002-4891-4835
Hugh Greatorex https://orcid.org/0000-0002-3716-3992
Alpha Renner https://orcid.org/0000-0002-0724-4169
Junren Chen https://orcid.org/0009-0000-9851-0903
Huaqiang Wu https://orcid.org/0000-0001-8359-7997
Giacomo Indiveri https://orcid.org/0000-0002-7109-1689
Elisabetta Chicca https://orcid.org/0000-0002-5518-8990

References

[1] Mead C 1990 Neuromorphic electronic systems *Proc. IEEE* **78** 1629–36
[2] Ganguli S, Huh D and Sompolinsky H 2008 Memory traces in dynamical systems *Proc. Natl Acad. Sci.* **105** 18970–5
[3] Khona M and Fiete I R 2022 Attractor and integrator networks in the brain *Nat. Rev. Neurosci.* **23** 744–66
[4] Hochreiter S and Schmidhuber J 1997 Long short-term memory *Neural Comput.* **9** 1735–80
[5] Bellec G, Scherr F, Subramoney A, Hajek E, Salaj D, Legenstein R and Maass W 2020 A solution to the learning dilemma for recurrent networks of spiking neurons *Nat. Commun.* **11** 3625
[6] Cramer B *et al* 2022 Surrogate gradients for analog neuromorphic computing *Proc. Natl Acad. Sci.* **119** 4
[7] Neftci E O, Mostafa H and Zenke F 2019 Surrogate gradient learning in spiking neural networks (arXiv:1901.09948 [cs, q-bio])
[8] Miller J and Hardt M 2019 Stable recurrent models *Int. Conf. on Learning Representations*
[9] Bengio Y, Simard P and Frasconi P 1994 Learning long-term dependencies with gradient descent is difficult *IEEE Trans. Neural Netw.* **5** 157–66
[10] Hopfield J J 1982 Neural networks and physical systems with emergent collective computational abilities *Proc. Natl Acad. Sci.* **79** 2554–8
[11] Amit D J 1989 *Modeling Brain Function: The World of Attractor Neural Networks*
[12] Bohnstingl T *et al* 2022 Biologically-inspired training of spiking recurrent neural networks with neuromorphic hardware *2022 IEEE 4th Int. Conf. on Artificial Intelligence Circuits and Systems (AICAS)* pp 218–21
[13] Schmitt S *et al* 2017 Neuromorphic hardware in the loop: training a deep spiking network on the BrainScaleS wafer-scale system *2017 Int. Joint Conf. on Neural Networks (IJCNN)* pp 2227–34
[14] Demirag Y, Frenkel C, Payvand M and Indiveri G 2021 Online training of spiking recurrent neural networks with phase-change memory synapses (arXiv:2108.01804 [cs])
[15] Cakal U, Wu M C, Ulusoy I and Muir D R 2024 Gradient-descent hardware-aware training and deployment for mixed-signal neuromorphic processors *Neuromorph. Comput. Eng.* **4** 014011
[16] Kleyko D *et al* 2022 Vector symbolic architectures as a computing framework for emerging hardware *Proc. IEEE* **110** 1538–71
[17] Kleyko D, Rachkovskij D A Osipov E and Rahimi A 2022 A survey on hyperdimensional computing aka vector symbolic architectures, part I: models and data transformations *ACM Comput. Surv.* **55** 40

[18] Kanerva P 2009 Hyperdimensional computing: an introduction to computing in distributed representation with high-dimensional random vectors *Cogn. Comput.* **1** 139–59
[19] Plate T A 1995 Holographic reduced representations *IEEE Trans. Neural Netw.* **6** 623–41
[20] Gayler R W 1998 Multiplicative binding, representation operators & analogy *Advances in Analogy Research: Integration of Theory and Data From the Cognitive, Computational and Neural Sciences*
[21] Karunaratne G, Le Gallo M, Cherubini G, Benini L, Rahimi A and Sebastian A 2020 In-memory hyperdimensional computing *Nat. Electr.* **3** 327–37
[22] Cotteret M, Greatorex H, Ziegler M and Chicca E 2024 Vector symbolic finite state machines in attractor neural networks *Neural Comput.* **36** 549–95
[23] Renner A, Supic L, Danielescu A, Indiveri G, Olshausen B A, Sandamirskaya Y, Sommer F T and Frady E P 2024 Neuromorphic visual scene understanding with resonator networks *Nat. Mach. Intell.* **6** 641–52
[24] Faisal A A, Selen L P J and Wolpert D M 2008 Noise in the nervous system *Nat. Rev. Neurosci.* **9** 292–303
[25] Neftci E, Binas J, Rutishauser U, Chicca E, Indiveri G and Douglas R J 2013 Synthesizing cognition in neuromorphic electronic systems *Proc. Natl Acad. Sci.* **110** E3468–76
[26] Liang D and Indiveri G 2019 A neuromorphic computational primitive for robust context-dependent decision making and context-dependent stochastic computation *IEEE Trans. Circ. Syst.* II **66** 843–7
[27] Orchard G, Frady E P, Rubin D B D, Sanborn S, Shrestha S B, Sommer F T and Davies M 2021 Efficient neuromorphic signal processing with Loihi 2 *2021 IEEE Workshop on Signal Processing Systems (SiPS)* pp 254–9
[28] Dayan P 2008 Simple substrates for complex cognition *Front. Neurosci.* **2** 31
[29] Amari S-i 1989 Characteristics of sparsely encoded associative memory *Neural Netw.* **2** 451–7
[30] Tsodyks M V and Feigel'man M V 1988 The enhanced storage capacity in neural networks with low activity level *Europhys. Lett.* **6** 101–5
[31] Laiho M, Poikonen J H, Kanerva P and Lehtonen E 2015 High-dimensional computing with sparse vectors *2015 IEEE Biomedical Circuits and Systems Conf. (BioCAS)* pp 1–4
[32] Frady E P, Kleyko D and Sommer F T 2023 Variable binding for sparse distributed representations: theory and applications *IEEE Trans. Neural Netw. Learn. Syst.* **34** 2191–204
[33] Knoblauch A, Palm G and Sommer F T 2010 Memory capacities for synaptic and structural plasticity *Neural Comput.* **22** 289–341
[34] Rumelhart D E and McClelland J L 1986 *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations* (MIT Press)
[35] Renner A, Sandamirskaya Y, Sommer F and Frady E P 2022 Sparse vector binding on spiking neuromorphic hardware using synaptic delays *Proc. Int. Conf. on Neuromorphic Systems 2022* pp 1–5
[36] Palm G 2013 Neural associative memories and sparse coding *Neural Netw.* **37** 165–71
[37] Rozell C J, Johnson D H, Baraniuk R G and Olshausen B A 2008 Sparse coding via thresholding and local competition in neural circuits *Neural Comput.* **20** 2526–63
[38] Neftci E and Indiveri G 2010 A device mismatch compensation method for VLSI neural networks *2010 Biomedical Circuits and Systems Conf. (BioCAS)* pp 262–5
[39] Büchel J, Zendrikov D, Solinas S, Indiveri G and Muir D R 2021 Supervised training of spiking neural networks for robust deployment on mixed-signal neuromorphic processors *Sci. Rep.* **11** 23376
[40] Greatorex H *et al* 2024 TEXEL: a neuromorphic processor with on-chip learning for beyond-CMOS device integration (arXiv:2410.15854)
[41] Fantini A, Gorine G, Degraeve R, Goux L, Chen C Y, Redolfi A, Clima S, Cabrini A, Torelli G and Jurczak M 2015 Intrinsic program instability in HfO2 RRAM and consequences on program algorithms *2015 IEEE Int. Electron Devices Meeting (IEDM)* pp 7.5.1–7.5.4
[42] Wang C, Wu H, Gao B, Dai L, Deng N, Sekar D C, Lu Z, Kellam M, Bronner G and Qian H 2016 Relaxation effect in RRAM arrays: demonstration and characteristics *IEEE Electron Device Lett.* **37** 182–5
[43] Chen J, Yang S, Wu H, Indiveri G and Payvand M 2024 Scaling limits of memristor-based routers for asynchronous neuromorphic systems *IEEE Trans. Circ. Syst.* II **71** 1576–80
[44] D'Agostino S, Moro F, Torchet T, Demirağ Y, Grenouillet L, Castellani N, Indiveri G, Vianello E and Payvand M 2024 DenRAM: neuromorphic dendritic architecture with RRAM for efficient temporal processing with delays *Nat. Commun.* **15** 3446
[45] Little W A 1974 The existence of persistent states in the brain *Math. Biosci.* **19** 101–20
[46] Rolls E 2013 The mechanisms for pattern completion and pattern separation in the hippocampus *Front. Syst. Neurosci.* **7** 74
[47] Chaudhuri R and Fiete I 2016 Computational principles of memory *Nat. Neurosci.* **19** 394–403
[48] Schneidman E, Berry M J, Segev R and Bialek W 2006 Weak pairwise correlations imply strongly correlated network states in a neural population *Nature* **440** 1007–12
[49] Mante V, Sussillo D, Shenoy K V and Newsome W T 2013 Context-dependent computation by recurrent dynamics in prefrontal cortex *Nature* **503** 78–84
[50] Sussillo D and Barak O 2013 Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks *Neural Comput.* **25** 626–49
[51] Miller P 2016 Itinerancy between attractor states in neural systems *Curr. Opin. Neurobiol.* **40** 14–22
[52] Rutishauser U and Douglas R 2009 State-dependent computation using coupled recurrent networks *Neural Comput.* **21** 478–509
[53] Baumgartner S, Renner A, Kreiser R, Liang D, Indiveri G and Sandamirskaya Y 2020 Visual pattern recognition with on on-chip learning: towards a fully neuromorphic approach *2020 IEEE Int. Symp. on Circuits and Systems (ISCAS)* pp 1–5
[54] Fusi S, Miller E and Rigotti M 2016 Why neurons mix: high dimensionality for higher cognition *Curr. Opin. Neurobiol.* **37** 66–74
[55] Thomas A, Dasgupta S and Rosing T 2022 A theoretical perspective on hyperdimensional computing *J. Artif. Intell. Res.* **72** 215–49
[56] Clarkson K L, Ubaru S and Yang E 2023 Capacity analysis of vector symbolic architectures (arXiv:2301.10352 [cs])
[57] Boahen K 2022 Dendrocentric learning for synthetic intelligence *Nature* **612** 43–50
[58] Barkam H E, Yun S, Chen H, Gensler P, Mema A, Ding A, Michelogiannakis G, Amrouch H and Imani M 2023 Reliable hyperdimensional reasoning on unreliable emerging technologies *2023 IEEE/ACM Int. Conf. on Computer Aided Design (ICCAD)* pp 1–9
[59] Frady E P and Sommer F T 2019 Robust computation with rhythmic spike patterns *Proc. Natl Acad. Sci.* **116** 18050–9
[60] Zou Z, Alimohamadi H, Zakeri A, Imani F, Kim Y, Najafi M H and Imani M 2022 Memory-inspired spiking hyperdimensional network for robust online learning *Sci. Rep.* **12** 7641

[61] Morris J, Lui H W, Stewart K, Khaleghi B, Thomas A, Marback T, Aksanli B, Neftci E and Rosing T 2022 HyperSpike: hyperdimensional computing for more efficient and robust spiking neural networks *2022 Design, Automation & Test in Europe Conf. & Exhibition (DATE)* pp 664–9
[62] Horn D and Usher M 1989 Neural networks with dynamical thresholds *Phys. Rev.* A **40** 1036–44
[63] Sommer F T and Wennekers T 2005 Synfire chains with conductance-based neurons: internal timing and coordination with timed input *Neurocomputing* **65-66** 449–54
[64] Kambara T, Kashimori Y, Usuba N and Hoshino O 1997 Role of itinerancy among attractors as dynamical map in distributed coding scheme *Neural Netw.* **10** 1375–90
[65] Gutfreund H and Mezard M 1988 Processing of temporal sequences in neural networks *Phys. Rev. Lett.* **61** 235–8
[66] Amit D J 1988 Neural networks counting chimes *Proc. Natl Acad. Sci. USA* **85** 2141–5
[67] Drossaers M F J 1992 Hopfield models as nondeterministic finite-state machines *Proc. 14th Conf. on Computational Linguistics - Volume 1, COLING'92* (Association for Computational Linguistics) pp 113–9
[68] Chen B and Miller P 2020 Attractor-state itinerancy in neural circuits with synaptic depression *J. Math. Neurosci.* **10** 15
[69] Peretto P and Niez J J 1986 Collective properties of neural networks *Disordered Systems and Biological Organization* ed E Bienenstock, F F Soulié and G Weisbuch (Springer) pp 171–85
[70] Osipov E, Kleyko D and Legalov A 2017 Associative synthesis of finite state automata model of a controlled object with hyperdimensional computing *IECON 2017 - 43rd Annual Conf. of the IEEE Industrial Electronics Society* pp 3276–81
[71] Maestro-Izquierdo M, González M B, Martín-Holgado P, Morilla Y and Campabadal F 2021 Gamma radiation effects on HfO2-based RRAM devices *2021 13th Spanish Conf. on Electron Devices (CDE)* pp 23–26
[72] Kleyko D, Rachkovskij D, Osipov E and Rahimi A 2023 A survey on hyperdimensional computing aka vector symbolic architectures, part II: applications, cognitive models and challenges *ACM Comput. Surv.* **55** 175
[73] Neubert P, Schubert S and Protzel P 2019 An introduction to hyperdimensional computing for robotics *KI - Kunstliche Intelligenz* **33** 319–30
[74] Yerxa T, Anderson A and Weiss E 2018 The hyperdimensional stack machine *Cognitive Computing Conference 2018* (*Hannover*) 1–2
[75] Marr D 1982 *Vision: A Computational Investigation Into the Human Representation and Processing of Visual Information* (W. H. Freeman and Company)
[76] Noest A J 1987 Phasor neural networks *Proc. 1987 Int. Conf. on Neural Information Processing Systems, (NIPS'87)* (MIT Press) pp 584–91
[77] Schuman C D, Kulkarni S R, Parsa M, Mitchell J P, Date P and Kay B 2022 Opportunities for neuromorphic computing algorithms and applications *Nat. Comput. Sci.* **2** 10–19
[78] Jaeger H, Noheda B and van der Wiel W G 2023 Toward a formal theory for computing machines made out of whatever physics offers *Nat. Commun.* **14** 4911
[79] Stefanini F, Neftci E O, Sheik S and Indiveri G 2014 PyNCS: a microkernel for high-level definition and configuration of neuromorphic electronic systems *Front. Neuroinf.* **8** 73
[80] Davison A P, Brüderle D, Eppler J M, Kremkow J, Muller E, Pecevski D, Perrinet L and Yger P 2009 PyNN: a common interface for neuronal network simulators *Front. Neuroinf.* **2** 08
[81] Aimone J B, Severa W and Vineyard C M 2019 Composing neural algorithms with Fugu *Proc. Int. Conf. on Neuromorphic Systems, (ICONS'19)* (Association for Computing Machinery) pp 1–8
[82] Zhang Y *et al* 2020 A system hierarchy for brain-inspired computing *Nature* **586** 378–84
[83] Eliasmith C 2005 A unified approach to building and controlling spiking attractor networks *Neural Comput.* **17** 1276–314
[84] Pedersen J E *et al* 2024 Neuromorphic intermediate representation: a unified instruction set for interoperable brain-inspired computing *Nat. Commun.* **15** 8122
[85] Poduval P, Alimohamadi H, Zakeri A, Imani F, Najafi M H, Givargis T and Imani M 2022 GrapHD: graph-based hyperdimensional memorization for brain-like cognitive learning *Front. Neurosci.* **16** 25
[86] Frady E P, Kleyko D, Kymn C J, Olshausen B A and Sommer F T 2022 Computing on functions using randomized vector representations (in brief) *Neuro-Inspired Computational Elements Conf., (NICE 2022)* (Association for Computing Machinery) pp 115–22
[87] Minsky M L 1967 *Computation: Finite and Infinite Machines* (Prentice-Hall, Inc.)
[88] Cohen M A and Grossberg S 1983 Absolute stability of global pattern formation and parallel memory storage by competitive neural networks *IEEE Trans. Syst. Man Cybern.* **SMC-13** 815–26
[89] Buhmann J and Schulten K 1987 Noise-driven temporal association in neural networks *Europhys. Lett.* **4** 1205–9
[90] Kleinfeld D 1986 Sequential state generation by model neural networks *Proc. Natl Acad. Sci. USA* **83** 9469–73
[91] Chicca E, Stefanini F, Bartolozzi C and Indiveri G 2014 Neuromorphic electronic circuits for building autonomous cognitive systems *Proc. IEEE* **102** 1367–88
[92] Richter O, Greatorex H, Hucko B, Cotteret M, Soares Girao W, Janotte E, Mastella M and Chicca E 2023 A subthreshold second-order integration circuit for versatile synaptic alpha kernel and trace generation *Proc. 2023 Int. Conf. on Neuromorphic Systems* (ACM) pp 1–4
[93] Mostafa H, Müller L K and Indiveri G 2015 Rhythmic inhibition allows neural networks to search for maximally consistent states *Neural Comput.* **27** 2510–47
[94] Cotteret M, Richter O, Mastella M, Greatorex H, Janotte E, Girão W S, Ziegler M and Chicca E 2023 Robust spiking attractor networks with a hard winner-take-all neuron circuit *2023 IEEE Int. Symp. on Circuits and Systems (ISCAS)* pp 1–5
[95] Abrahamsen J, Hafliger P and Lande T 2004 A time domain winner-take-all network of integrate-and-fire neurons *2004 IEEE Int. Symp. on Circuits and Systems (ISCAS)* vol 5 p V–V
[96] Stimberg M, Brette R and Goodman D F 2019 Brian 2, an intuitive and efficient neural simulator *eLife* **8** e47314
[97] Bartol Jr T M, Bromer C, Kinney J, Chirillo M A, Bourne J N, Harris K M and Sejnowski T J 2015 Nanoconnectomic upper bound on the variability of synaptic plasticity *eLife* **4** e10778
[98] Zahari F, Pérez E, Mahadevaiah M K, Kohlstedt H, Wenger C and Ziegler M 2020 Analogue pattern recognition with stochastic switching binary CMOS-integrated memristive devices *Sci. Rep.* **10** 14450
[99] Sompolinsky H 1987 The theory of neural networks: the Hebb rule and beyond *Heidelberg Colloquium on Glassy Dynamics* (*Lecture Notes in Physics*) ed J L van Hemmen and I Morgenstern (Springer) pp 485–527
[100] Willshaw D J, Buneman O P and Longuet-Higgins H C 1969 Non-holographic associative memory *Nature* **222** 960–2