Cellular Automata Can Reduce Memory Requirements of Collective-State Computing

Denis Kleyko, *Member, IEEE*, Edward Paxon Frady, and Friedrich T. Sommer

***Abstract*— Various nonclassical approaches of distributed information processing, such as neural networks, reservoir computing (RC), vector symbolic architectures (VSAs), and others, employ the principle of collective-state computing. In this type of computing, the variables relevant in computation are superimposed into a single high-dimensional state vector, the collective state. The variable encoding uses a fixed set of random patterns, which has to be stored and kept available during the computation. In this article, we show that an elementary cellular automaton with rule 90 (CA90) enables the space–time tradeoff for collective-state computing models that use random dense binary representations, i.e., memory requirements can be traded off with computation running CA90. We investigate the randomization behavior of CA90, in particular, the relation between the length of the randomization period and the size of the grid, and how CA90 preserves similarity in the presence of the initialization noise. Based on these analyses, we discuss how to optimize a collective-state computing model, in which CA90 expands representations on the fly from short seed patterns—rather than storing the full set of random patterns. The CA90 expansion is applied and tested in concrete scenarios using RC and VSAs. Our experimental results show that collective-state computing with CA90 expansion performs similarly compared to traditional collective-state models, in which random patterns are generated initially by a pseudorandom number generator and then stored in a large memory.**

***Index Terms*— Cellular automata (CA), collective-state computing, distributed representations, hyperdimensional computing, random number generation, reservoir computing (RC), rule 90, vector symbolic architectures (VSAs).**

I. Introduction

COLLECTIVE-STATE computing is an emerging paradigm of computing, which leverages interactions of nodes or neurons in a highly interconnected network [1]. This paradigm was first proposed in the context of neural networks and neuroscience for exploiting the parallelism of complex network dynamics to perform challenging computations. The classical examples include reservoir computing (RC) [2]–[5] for buffering spatiotemporal inputs and attractor networks for associative memory [6] and optimization [7]. In addition, many other models can be regarded as collective-state computing, such as random projection [8], compressed sensing [9], [10], randomly connected feedforward neural networks [11], [12], and vector symbolic architectures (VSAs) [13]–[15]. Interestingly, these diverse computational models share a fundamental commonality—they all include an initialization phase in which high-dimensional independent and identically distributed (i.i.d.) random vectors or matrices are generated, which have to be memorized.

In different models, these memorized random vectors or matrices serve a similar purpose: to represent inputs and variables that need to be manipulated as distributed patterns across all neurons. For example, in RC, random matrices are used as weights for projecting input to the reservoir (commonly denoted as $\mathbf{W}^{\text{in}}$) as well as weights of recurrent connections between the neurons in the reservoir (commonly denoted as $\mathbf{W}$). The collective state is the (linear) superposition of these distributed representations. Decoding a particular variable from the collective state can be achieved by a linear projection onto the corresponding representation vector. Since high-dimensional random vectors are pseudo-orthogonal, the decoding interference is rather small, even if the collective state contains many variables.[1] In contrast, if representations of different variables are not random but contain correlations or statistical dependencies, then the interference becomes large when decoding and collective-state computing ceases to work. In order to achieve near orthogonality and low decoding interference, a large dimension of the random vectors is essential.

When implementing a collective-state computing model in hardware [e.g., in field-programmable gate arrays (FPGAs)], the memory requirements are typically a major bottleneck for scaling the system [16]. It seems strangely counterintuitive to spend a large amount of memory just to store random vectors. Thus, our key question is whether collective-state computing

Manuscript received October 7, 2020; revised June 25, 2021; accepted October 4, 2021. The work of Denis Kleyko was supported in part by the European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska-Curie Individual Fellowship Grant 839179 and in part by the Defense Advanced Research Projects Agency's (DARPA's) Virtual Intelligence Processing (VIP, Super-HD Project) and Artificial Intelligence Exploration (AIE, HyDDENN Project) Programs. The work of Friedrich T. Sommer was supported by NIH under Grant R01-EB026955. *(Corresponding author: Denis Kleyko.)*

Denis Kleyko is with the Redwood Center for Theoretical Neuroscience, University of California at Berkeley, Berkeley, CA 94720 USA, and also with the Intelligent Systems Laboratory, Research Institutes of Sweden, 164 40 Kista, Sweden (e-mail: denkle@berkeley.edu).

Edward Paxon Frady and Friedrich T. Sommer are with the Neuromorphic Computing Laboratory, Intel Labs, Santa Clara, CA 95054 USA, and also with the Redwood Center for Theoretical Neuroscience, University of California at Berkeley, Berkeley, CA 94720 USA (e-mail: epaxon@berkeley.edu; fsommer@berkeley.edu).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2021.3119543.

Digital Object Identifier 10.1109/TNNLS.2021.3119543

[1] Although decoding by projection would work even better for exactly orthogonal distributed patterns, such a coding scheme is less desirable: i.i.d random generation of distributed patterns is computationally much cheaper and does not pose a hard limit on the number of encoded variables to be smaller or equal than the number of nodes or neurons.

can be achieved without memorizing the full array of random vectors. Instead of memorization, can memory requirements be traded off by computation?

Cellular automata (CA) are simple discrete computations capable of producing complex random behavior [17]. Here, we study the randomization behavior of an elementary cellular automaton with rule 90 (CA90) for generating distributed representations for collective-state computing. CA90 is chosen because of its highly parallelizable implementation and randomization properties, which, in our opinion, are amenable for collective-state computing (see Section II-C). We demonstrate in the context of RC and VSAs that collective-state computing at full performance is possible by storing only short random seed patterns, which are then expanded "on the fly" to the full required dimension by running rule CA90. Thus, CA90 provides the space–time tradeoff for collective-state computing models since, instead of memorizing random vectors/matrices, CA90 allows to recompute them in real time while using only a fraction of the fully memorized solution at the cost of running CA90 computations every time when the access is required. This work is partially inspired by [18], which proposed that RC, VSAs, and CA can benefit from each other, by expanding low-dimensional representations via CA computations into high-dimensional representations that are then used in RC and VSA models. The specific contributions of this article are given in the following.

1) Characterization of the relation between the length of the randomization period of CA90 and the size of its grid.
2) Analysis of the similarity between CA90 expanded representations in the case when the seed pattern contains errors.
3) Experimental evidence is that for RC and VSAs, the CA90 expanded representations are functionally equivalent to the representations obtained from a standard pseudorandom number generator.[2]

This article is structured as follows. The main concepts used in this study are presented in Section II. The effect of randomization of states by CA90 is described in Section III. The use of RC and VSAs with the CA90 expanded representations is reported in Section IV. The findings and their relation to the related work are discussed in Section V.

II. Concepts

A. Collective-State Computing

As explained in Section I, collective-state computing subsumes numerous types of network computations that employ distributed representation schemes based on i.i.d. random vectors. One type is VSA or hyperdimensional computing [13], [19], [20], which has been proposed in cognitive neuroscience as a formalism for symbolic reasoning with distributed representations. Recently, the VSA formalism has been used to formulate other types of collective-state computing models, such as RC [5], compressed sensing [21], and randomly connected feedforward neural networks [12] such as random vector functional link networks [11] and extreme learning machines [22]. Following this lead, we will formulate the types of collective-state computing, which are used in Section IV to study the CA90 expansion.[3]

VSAs are defined for different spaces (e.g., real or complex), but here, we focus on VSAs with dense binary [23] or bipolar [24] vectors where the similarity between vectors is measured by normalized Hamming distance (denoted as d_h) for binary vectors or by dot product for bipolar ones (denoted as d_d). The VSA formalism will be introduced as we go. Please also refer to Section S.2 in the Supplementary Materials for a concise introduction to VSAs.

1) Item Memory With Nearest Neighbor Search: A common feature in collective-state computing is that a set of basic concepts/symbols[4] is defined and assigned with i.i.d. random high-dimensional atomic vectors. In VSAs, these atomic vectors are stored in the so-called item memory (denoted as $\mathbf{H}$), which in its simplest form is a matrix with the size dependent on the dimensionality of vectors (denoted as K) and the number of symbols (denoted as D). The item memory $\mathbf{H}$ enables associative or content-based search, that is, for a given query vector $\mathbf{q}$, it returns the nearest neighbor. Given such a noisy query, the memory returns the best match using the nearest neighbor search

$$\arg\min_i (d_h(\mathbf{H}_i, \mathbf{q})). \tag{1}$$

The search returns the index of the vector that is closest to the query in terms of the similarity metric [e.g., the Hamming distance as in (1)]. In VSAs and, implicitly, in most types of collective-state computing, this content-based search is required for selecting and error-correcting results of computations that, in noisy form, have been produced by dynamical transformations of the collective state.

2) Memory Buffer: RC is a prominent example of collective-state computing as in echo state networks [3], liquid state machines [2], and state-dependent computation [35]. In these models, the dynamics of a recurrent network is used to memorize or buffer the structure of a spatiotemporal input pattern. In essence, the memory buffer accumulates the input history over time into a compound vector. For example, the recurrent network dynamics of echo state networks can be viewed as attaching time stamps to the inputs arriving at different times.[5] The time stamps can be used to isolate past inputs at a particular time from the compound vector describing the present network state. For a detailed explanation of this view of echo state networks, see [5].

It has been recently shown how the memory buffer task can be analyzed using the VSA formalism [5], [36], which builds

[2]Please note that while a (pseudo-) random number generator is not part of collective-state computing models (RC, VSA, and so on) during run time, it is required for model initialization.

[3]There are several different terms, describing computations in CA, e.g., "expansion," "evolution," and development. In the context of this article, we use terms expansion and evolution interchangeably. The term "expansion" is used to emphasize the fact that CA90 is used to produce distributed representations, while the term "evolution" is used when referring to running CA computations.

[4]Examples of the basic concepts are distinct features in machine learning problems [25]–[28] or distinct symbols in data structures [29]–[34].

[5]This is done implicitly since the recurrent connection matrix is applied a different number of times to inputs presented at different time stamps.

on earlier VSA proposals of the memory buffer task under the name trajectory association task [37], [38].

Here, we use a simple variant of the echo state network [39], called integer echo state network [40], which uses a random binary matrix for projecting the input to the reservoir. The memory buffer involves the item memory and two other VSA operations: permutation and bundling. The item memory contains a random binary/bipolar vector assigned for each symbol from a dictionary of symbols of size D. The item memory corresponds to $\mathbf{W}^{\text{in}}$ weight matrix for projecting input to the reservoir. A fixed permutation (rotation) of the components of the vector (denoted as ρ)[6] is used to represent the position of a symbol in the input sequence. In other words, the permutation operation is used as a systematic transformation of a symbol as a function of its serial position. For example, a symbol a represented by $\mathbf{a}$ is associated with its position i in the sequence by the result of permutation (denoted as $\mathbf{r}$) as

$$\mathbf{r} = \rho^i(\mathbf{a}) \tag{2}$$

where $\rho^i(*)$ denotes that some fixed permutation $\rho()$ has been applied i times.

The bundling operation forms a linear superposition of several vectors, which in some form is present in all collective-state computing models. Its simplest realization is a componentwise addition. However, when using the componentwise addition, the vector space becomes unlimited, and therefore, it is practical to limit the values of the result. In general, the normalization function applied to the result of superposition is denoted as $f_n(*)$.[7] The vector $\mathbf{x}$ resulting from the bundling of several vectors, e.g.,

$$\mathbf{x} = f_n(\mathbf{a} + \mathbf{b} + \mathbf{c}) \tag{3}$$

is similar to each of the bundled vectors, which allows storing information as a superposition of multiple vectors [5], [41]. Therefore, in the context of the memory buffer, the bundling operation is used to update the buffer with new symbols.

The memory buffer task involves two stages: memorization and recall, which are done in discrete time steps. At the memorization stage, at every time step t, we add a vector $\mathbf{H}_{\mathbf{s}(t)}$ representing the symbol $\mathbf{s}(t)$ from the sequence $\mathbf{s}$ to the current memory buffer $\mathbf{x}(t)$, which is formed as

$$\mathbf{x}(t) = f_n\big(\rho(\mathbf{x}(t-1)) + \mathbf{H}_{\mathbf{s}(t)}\big) \tag{4}$$

where $\mathbf{x}(t-1)$ is the previous state of the buffer. Note that the symbol added d step ago is represented in the buffer as $\rho^{d-1}(\mathbf{H}_{\mathbf{s}(t-d)})$.

At the recall stage, at every time step, we use $\mathbf{x}(t)$ to retrieve the prediction of the delayed symbol stored d steps ago $[\hat{\mathbf{s}}(t-d)]$ using the readout matrix $(\mathbf{W}^d)$ for particular d using the nearest neighbor search

$$\hat{\mathbf{s}}(t-d) = \arg\max_i \big(d_d\big(\mathbf{W}_i^d, \mathbf{x}(t)\big)\big). \tag{5}$$

[6]The cyclic shift is used frequently due to its simplicity.

[7]In the case of dense binary VSAs, the arithmetic sum vector of two or more vectors is thresholded back to binary space vector by using the majority rule/sum [denoted as $f_m(*)$] where ties are broken at random.

Due to the use of a normalization function $f_n(*)$, the memory buffer possesses the recency effect, and therefore, the average accuracy of the recall is higher for smaller values of delay. There are several approaches how to form the readout matrix $\mathbf{W}^d$ and the normalization function $f_n(*)$. Please see Section S.1 in the Supplementary Materials for additional details on integer echo state networks.

3) Factorization With Resonator Network: General symbolic manipulations with VSA require one other operation, in addition to bundling, permutation, and item memory. The representation of an association of two or more symbols, such as a role-filler pair, is achieved by a binding operation, which associates several vectors (e.g., $\mathbf{a}$ and $\mathbf{b}$) together and produces another vector (denoted as $\mathbf{z}$) of the same dimensionality

$$\mathbf{z} = \mathbf{x} \oplus \mathbf{y} \tag{6}$$

where the notation $\oplus$ denotes componentwise XOR used for the binding in dense binary VSAs. While bundling leads to a vector that is correlated with each of its components, in binding, the resulting vector $\mathbf{z}$ is pseudo-orthogonal to the component vectors. Another important property of binding is that it is conditionally invertible. Given all but one components, one can simply compute from the binding representation of the unknown factor, e.g., $\mathbf{z} \oplus \mathbf{x} = \mathbf{x} \oplus \mathbf{x} \oplus \mathbf{y} = \mathbf{y}$.

If none of the factors are given, but are contained in the item memory, the unbinding operation is still feasible but becomes a combinatorial search problem, whose complexity grows exponentially with the number of factors. This problem often occurs in symbolic manipulation problems, for example, in finding the position of a given item in a tree structure [42]. Let us assume that each component (factor, denoted as $\mathbf{f}_i$) comes from a separate item memory ($^1\mathbf{H}, ^2\mathbf{H}, \ldots$), which is called factor item memory, e.g., a general example of a vector with four factors is

$$\mathbf{f}_1 \oplus \mathbf{f}_2 \oplus \mathbf{f}_3 \oplus \mathbf{f}_4. \tag{7}$$

Recent work [43] proposes an elegant mechanism called the resonator network to address the challenge of factoring. In the nutshell, the resonator network [43] is a novel recurrent neural network design that uses VSAs principles to solve combinatorial optimization problems.

To factor the components from the input vector $\mathbf{f}_1 \oplus \mathbf{f}_2 \oplus \mathbf{f}_3 \oplus \mathbf{f}_4$ representing the binding of several vectors, the resonator network uses several populations of units, $\hat{\mathbf{f}}_1(t), \hat{\mathbf{f}}_2(t), \ldots$, each of which tries to infer a particular factor from the input vector. Each population, called a resonator, communicates with the input vector and all other neighboring populations to invert the input vector using the following dynamics:

$$\begin{aligned}
\hat{\mathbf{f}}_1(t+1) &= f_n\Big(^1\mathbf{H}^1\mathbf{H}^\intercal\Big(\mathbf{z} \oplus \hat{\mathbf{f}}_2(t) \oplus \hat{\mathbf{f}}_3(t) \oplus \hat{\mathbf{f}}_4(t)\Big)\Big)\\
\hat{\mathbf{f}}_2(t+1) &= f_n\Big(^2\mathbf{H}^2\mathbf{H}^\intercal\Big(\mathbf{z} \oplus \hat{\mathbf{f}}_1(t) \oplus \hat{\mathbf{f}}_3(t) \oplus \hat{\mathbf{f}}_4(t)\Big)\Big)\\
\hat{\mathbf{f}}_3(t+1) &= f_n\Big(^3\mathbf{H}^3\mathbf{H}^\intercal\Big(\mathbf{z} \oplus \hat{\mathbf{f}}_1(t) \oplus \hat{\mathbf{f}}_2(t) \oplus \hat{\mathbf{f}}_4(t)\Big)\Big)\\
\hat{\mathbf{f}}_4(t+1) &= f_n\Big(^4\mathbf{H}^4\mathbf{H}^\intercal\Big(\mathbf{z} \oplus \hat{\mathbf{f}}_1(t) \oplus \hat{\mathbf{f}}_2(t) \oplus \hat{\mathbf{f}}_3(t)\Big)\Big).
\end{aligned} \tag{8}$$

Note that the process is iterative and progresses in discrete time steps, t. In essence, at time t, each resonator $\hat{\mathbf{f}}_i(t)$ can hold multiple weighted guesses for a vector from each factor item memory through the VSAs principle of superposition, which is used for the bundling operation. Each resonator also uses the current guesses for factors from other resonators. These guesses from the other resonators are used to invert the input vector and infer the factor of interest in the given resonator. The principle of superposition allows a population to hold multiple estimates of factor identity and test them simultaneously. The cost of superposition is a crosstalk noise. The inference step is, thus, noisy when many guesses are tested at once. However, the next step is to use factor item memory ${}^{i}\mathbf{H}$ to remove the extraneous guesses that do not fit. Thus, the guess $\hat{\mathbf{f}}_i$ for each factor is cleaned up by constraining the resonator activity only to the allowed atomic vectors stored in ${}^{i}\mathbf{H}$. Finally, a regularization step [denoted as $f_n(*)$] is needed. Successive iterations of this inference and clean-up procedure (8) eliminate the noise as the factors become identified and find their place in the input vector. When the factors are fully identified, the resonator network reaches a stable equilibrium and the factors can be read out from the stable activity pattern. Please refer to Section S.3 in the Supplementary Materials for additional motivation and explanation of the resonator network.

B. CA-Based Expansion

The CA is a discrete computational model consisting of a regular grid of cells [17] of size N. Each cell can be in one of a finite number of states (the elementary CA is binary). States of cells evolve in discrete time steps according to some fixed rule. In the elementary CA, the new state of a cell at the next step depends on its current state and the states of its immediate neighbors. Despite the seeming simplicity of the system, among the elementary CAs, there are rules (e.g., rule 110) that make CA dynamics operate at the edge of chaos [44] and were proven to be Turing complete [45]. In the scope of this article, we consider another rule—rule 90 (CA90) as it possesses several properties highly relevant for collective-state computing.

In the elementary CA, the state of a cell is updated using the current states of the cell itself and its left and right neighboring cells. A computation step in CA refers to the simultaneous update of states of all the cells in a grid. CA with binary states, there are in total $2^3 = 8$ possible input combinations for each input there are two possible assignments for the output cell, which makes in total $2^8 = 256$ combinations where each particular assignment defines a rule. Fig. 1 shows all input combinations and corresponding assignment of output states for CA90. CA90 assigns the next state of a central cell based on the previous states of the neighbors. In particular, the new state is the result of XOR operation on the states of the neighboring cells. This is particularly attractive because CA90 has a computational advantage since the CA implementation can be easily vectorized and implemented in hardware (especially when working with cyclic boundary conditions[8]). For example,

[8]Cyclic boundary condition means that the first and the last cells in the grid are considered to be neighbors.

Fig. 1. Assignment of new states for a center cell when the CA uses rule 90. A hollow cell corresponds to zero state, while a shaded cell marks one state.

if at time step t, vector $\mathbf{x}(t)$ describes the states of the CA grid, then the grid state at $t+1$ is computed as

$$\mathbf{x}(t+1) = \rho^{+1}(\mathbf{x}(t)) \oplus \rho^{-1}(\mathbf{x}(t)) \tag{9}$$

where $\rho^{\{+1,-1\}}$ is the notation for cyclic shift to the right or left by one. Note that (9) is identical to one step of evolution of CA90 with cyclic boundary conditions. This observation is important as it allows for highly parallelized implementation in, e.g., FPGA or application-specific integrated circuits (ASICs).

Since, in the context of this study, we use CA90 for the purposes of randomization, we will call the state of the grid $\mathbf{x}(0)$ at the beginning of computations as an initial short seed. It is worth pointing out that CA90 formulated as in (9) is a sequence of VSA operations [46]. Given the vector $\mathbf{x}$ as an argument, by performing two rotations ($\rho^{+1}(\mathbf{x})$ and $\rho^{-1}(\mathbf{x})$) and then binding the results of rotations together ($\rho^{+1}(\mathbf{x}) \oplus \rho^{-1}(\mathbf{x})$), we implement CA90.

The core idea of this article is to use CA90 to generate a distributed representation of expanded dimensionality that can be used within the context of collective-state computing. This expansion must have certain randomization properties and be robust to perturbations. Fig. 2 presents the basic idea of obtaining an expanded dense binary distributed representation from a short initial seed. In essence, the seed is used to initialize the CA grid. Once initialized, CA90 computations are applied for several steps (denoted as L).

To illustrate the idea, let us consider a concrete example. In the context of RC, the CA90 expansion might be used to significantly reduce the memory footprint required to store a random matrix $\mathbf{W}^{\text{in}}$ of an echo state network containing the weights between input and reservoir layers. Normally, $\mathbf{W}^{\text{in}} \in [K \times u]$, where K corresponds to the size of the reservoir and u denotes the size of the input. Instead of storing the whole $\mathbf{W}^{\text{in}}$, the use of CA90 expansion allows storing a smaller matrix $\mathbf{S}(0) \in [N \times u]$, with $N = K/L$. Thus, the usage of CA90 in RC reduces the memory requirements by a factor of L. In order to explicitly rematerialize $\mathbf{W}^{\text{in}}$ obtained from the expansion, $\mathbf{S}(0)$ is used to initialize the grid and CA90 (9) is run for $L-1$ steps

$$\begin{aligned}\mathbf{W}^{\text{in}} = \big[&\mathbf{S}(0); \rho^{+1}(\mathbf{S}(0)) \oplus \rho^{-1}(\mathbf{S}(0)); \ldots; \rho^{+1}(\mathbf{S}(L-3)) \\ &\oplus \rho^{-1}(\mathbf{S}(L-3)); \rho^{+1}(\mathbf{S}(L-2)) \oplus \rho^{-1}(\mathbf{S}(L-2))\big]\end{aligned} \tag{10}$$

where $[\cdot,\cdot]$ denotes the concatenation along the vertical dimension and $\mathbf{S}(i) = \rho^{+1}(\mathbf{S}(i-1)) \oplus \rho^{-1}(\mathbf{S}(i-1)), 1 \leq i \leq L-1$. Thus, the expansion comes at a computational cost of executing $L-1$ steps of CA90. At every step i of CA90 evolution, the states of the grid in $\mathbf{S}(i)$ provide a new burst of $N \times u$ bits, which can be either used on the fly (without memorization) to make the necessary manipulations and then

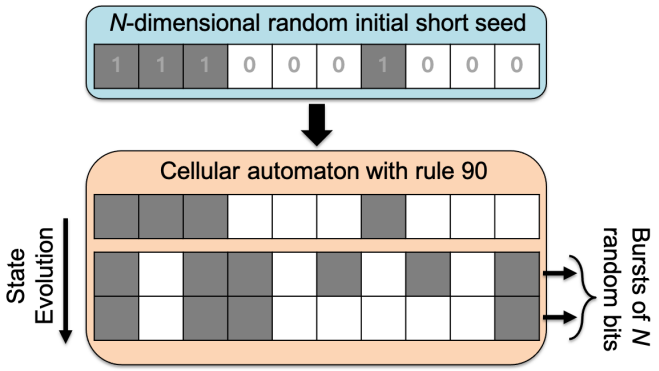


Fig. 2. Basic scheme for expanding distributed representations with CA90 from some initial short seed. The dimensionality of the seed N is equal to the size of the CA grid.

erased or concatenated [with memorization as shown in (10)] to the previous states if the distributed representation should be rematerialized explicitly. In any case, the dimensionality of the expanded representation is $K = NL$.

C. CA90 and VSAs

Section V presents the joint use of RC, VSAs, and CA90 expansion. Among the related works discussed [47]–[52] (see Section V-B1), [50] is the most relevant, as it uses the randomization property of CA90. In particular, this work identified the following useful properties of CA90 for VSAs:

1) random projection;
2) preservation of the binding operation;
3) preservation of the cyclic shift.

By random projection, we mean that when CA90 is initialized with a random state $\mathbf{x}(0)$ ($p_1 \approx p_0 \approx 0.5$), which should be seen as an initial short seed, its evolved state at step t is a vector $\mathbf{x}(t)$ of the same size and density. Moreover, during the randomization period (see Section III-A), $\mathbf{x}(t)$ is dissimilar to the initial short seed $\mathbf{x}(0)$, i.e., $d_h(\mathbf{x}(0), \mathbf{x}(t)) \approx 0.5$ as well as to the other states in the evolution of the seed.

The preservation of the binding operation refers to the fact that if a seed $\mathbf{c}(0)$ is the result of the binding of two other seeds: $\mathbf{c}(0) = \mathbf{a}(0) \oplus \mathbf{b}(0)$, then after t computational steps of CA90, the evolved state $\mathbf{c}(t)$ can be obtained by binding the evolved states of the initial seeds $\mathbf{a}(0)$ and $\mathbf{b}(0)$ used to form $\mathbf{c}(0)$, i.e., $\mathbf{c}(t) = \mathbf{a}(t) \oplus \mathbf{b}(t)$.

Finally, CA90 computations preserve a special case of the permutation operation—cyclic shift by i cells. Suppose that $\mathbf{d}(0) = \rho^i(\mathbf{a}(0))$ is an initial seed. After t computational steps of CA90, the cyclic shift of the evolved seed $\mathbf{a}(t)$ by i cells equals the evolved shifted seed $\mathbf{d}(t)$ so that $d_h(\mathbf{d}(t), \rho^i(\mathbf{a}(t)) = 0$.

III. Randomization of States by CA

A. Errorless Randomization

Usually, distributed representations in collective-state computing use i.i.d. random vectors. Similarly, we start with i.i.d. random vectors for short seeds. However, in contrast to the conventional approach, we are going to expand the dimensionality of the seed vectors via CA90 with boundary conditions. An important question for expansion is what are the limits of CA90 in terms of producing randomness?

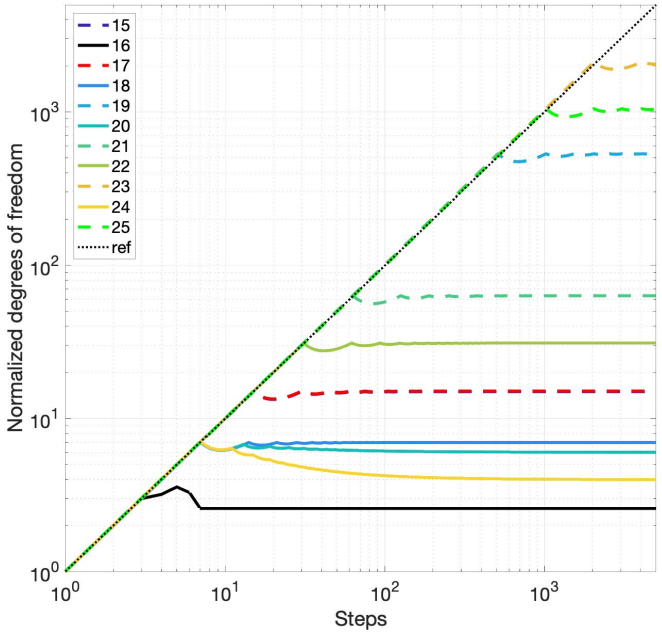


Fig. 3. Normalized degrees of freedom for different values of the grid size of CA90. The evolution of CA90 is reported for 5000 steps. The number of short seeds in the item memory was fixed to 100. The reported values were averaged over 100 simulations randomizing initial short seeds. Note the logarithmic scales of the axes.

One very useful empirical tool for answering this question is calculation of degrees of freedom from the statistics of normalized Hamming distances between binary vectors (see [53] for an example). Given that p_h denotes the average normalized Hamming distance and σ_h denotes its standard deviation, the degrees of freedom are calculated as

$$F = p_h(1 - p_h)/\sigma_h^2. \tag{11}$$

Due to the randomization properties of CA90, we expect that after a certain number of steps, it will produce new degrees of freedom. To be able to compare different grid sizes, we will report the degrees of freedom normalized by the grid size, i.e., F/N. In other words, if a single step of CA90 increased F by N (best case), the normalized value would increase by 1.

Fig. 3 presents the normalized degrees of freedom measured for 5000 steps of CA90 for different grid sizes using the same values as in [17, p. 259]. From the figure, we can draw several interesting observations. First, for all of the considered grid sizes, the degrees of freedom grow linearly at the beginning (following the reference, black dashed line, which indicates degrees of freedom in random binary vectors of the corresponding size). At some point, however, the degrees of freedom reach a maximum value and start to saturate, as we would expect. We are interested in the period of linear growth, and we call this the randomization period. Second, we see that larger grid sizes typically have longer randomization periods. For example, the longest randomization period of 2047 steps was observed for $N = 23$ (but this is not the largest

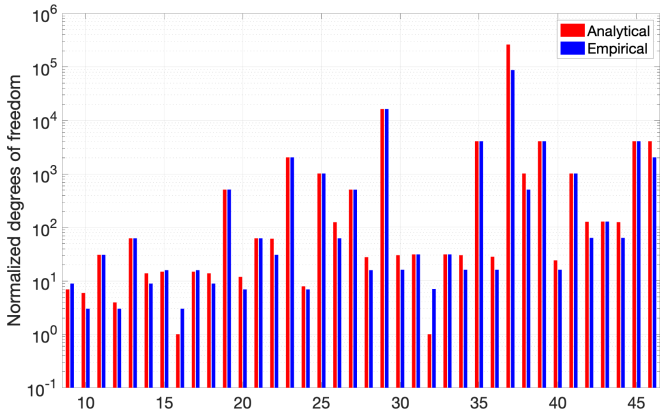


Fig. 4. Empirically measured randomization period (blue) and the analytical periodic cycles [54] (red) for the grid size in the range $[9, 46]$. Note the logarithmic scale of y-axis.

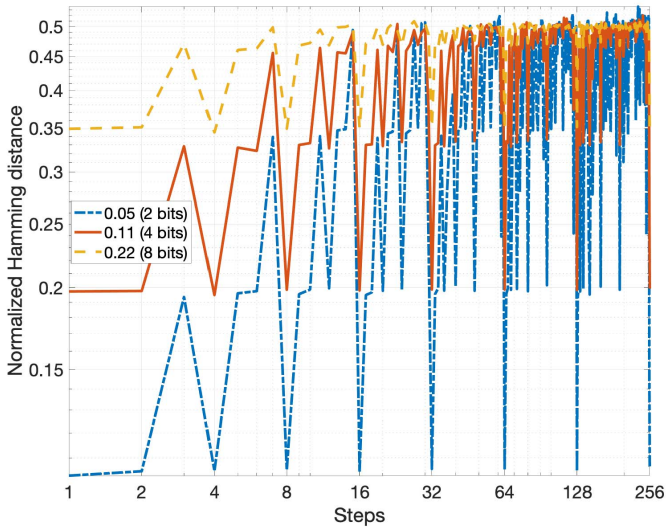


Fig. 5. Normalized Hamming distance between the original and noisy vectors obtained from short seeds for $N = 37$ during the first 256 steps of CA90 evolution. The reported values were averaged over 500 simulations where both initial short seeds and errors were chosen at random. Note the logarithmic scales of axes.

grid size).[9] Third, the randomization period of odd grid sizes is always longer than that of the even ones. For example, the randomization periods for $N = 22$ and $N = 24$ were only 31 and 7, respectively (cf. 2047 for $N = 23$). Thus, there is a dependency between N and the randomization period, but, despite the above observations, there is no clear general pattern connecting the grid size to the length of the randomization period.

The good news, however, is that the length of the randomization period is closely related to the length of periodic cycles (denoted as Π_N) in CA90 discovered in [54]. In short, the irregular behavior of randomization periods and periodic cycles is a consequence of their dependence on number theoretical properties of N; Martin *et al.* [54] provided the following characterization of periodic cycles Π_N in CA90.

1) For CA90 with N of the form 2^j, $\Pi_N = 1$.
2) For CA90 with N even but not of the form 2^j, $\Pi_N = 2\Pi_{N/2}$.
3) For CA90 with N odd, $\Pi_N | \Pi^*_N = 2^{\text{sord}_N(2)} - 1$, where $sord_N(2)$ is the multiplicative "suborder" function of 2 modulo N, defined as the least integer j such that $2^j = \pm 1 \mod N$.

Fig. 4 presents the empirically measured randomization periods as well as analytically calculated periodic cycles Π^*_N [54] for the grid size in the range $[9, 46]$. First, we see that when N is odd, the randomization period equals the periodic cycle. The only exception is the case when $N = 37$, but this is just the first exception where $\Pi_N = \Pi^*_N/3$. Second, when N is of the form 2^j, the randomization period does not equal one because CA90 is producing activity for 2^{j-1} steps, which increases the degrees of freedom. In fact, the randomization period in this case is $2^{j-2} - 1$. Third, when N is even, the CA90 produces Π_N unique grid states but the patterns of Hamming distances between the states evolved from two random initial short seeds start to repeat after $\Pi_N/2$ steps, and thus, they do not contribute new degrees of freedom. Therefore, the randomization period is two times lower than the periodic cycle. Aggregating these points, with respect to the randomization period of CA90, we have the following.

1) For CA90 with N of the form 2^j, the randomization period is $2^{j-2} - 1$.
2) For CA90 with N even but not of the form 2^j, the randomization period is $\Pi_{N/2}$.
3) For CA90 with N odd, the randomization period is divide of $\Pi^*_N = 2^{\text{sord}_N(2)} - 1$.

[9]As will be explained later in this section, this is because the length of the randomization period depends on N and primes tend to have long randomization periods. Since $N = 23$ is the largest prime used in Fig. 3, this grid size exhibits the longest randomization period among all considered grid sizes.

B. Effect of Noise in the Short Seed

In Section III-A, we have seen how CA90 can be used to expand initial short seeds into longer randomized representations. This property could be utilized by a collective-state system for efficient communication by exchanging only short seeds and expanding the seed with CA90. Since, in reality, communication channels are noisy, one must be able to handle some amount of error in the communicated short seeds. Therefore, it is important to understand how the evolution of an expanded distributed representation is affected by errors in the initial short seed.

We address this issue by observing the empirical behavior for $N = 37$ and the first 256 steps of CA90 evolution. Fig. 5 reports the averaged normalized Hamming distance for an errorless short seed and a noisy version of it, where either 2 (dashed-dotted line), 4 (solid line), or 8 (dashed line) bits were flipped randomly. The results reported are for the corresponding states of the grid at a given step, not for the concatenated states. This shows that even a single step of CA90 increases the normalized Hamming distance between the evolved states. For example, when 4 bits were flipped, the normalized Hamming distance between the seeds was $4/37 \approx 0.11$, while after a single step of CA90, it increased

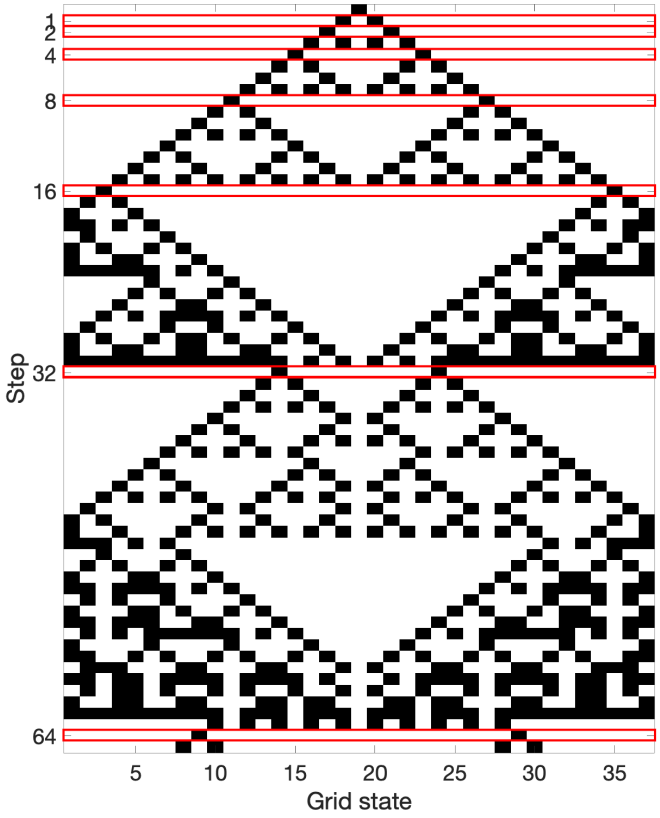


Fig. 6. Evolution of CA90 for 65 steps, $N = 37$; the initial state includes one active cell, which can be thought as introducing one bit flip to some random initial short seed. All steps of the form 2^j are highlighted by red rectangles.

to almost 0.2, almost doubling. Furthermore, we observe that the normalized Hamming distance will never be lower than after the first step.

What is very interesting is that the distances induced by errors change in a predictable manner. We see that the errors reset to the lowest possible value at regular intervals: each CA90 step of the form 2^j. This behavior of CA90 suggests that we can mitigate the impact of errors when expanding short seeds. In order to minimize the distance between the errorless evolutions and their noisy versions, one should only use the CA90 expansion in steps of the form 2^j, which places additional limits for the possibility of dimensionality expansion.

To understand the observed cyclic behavior of CA90, we examine the case when the initial state of the grid includes only one active cell. Due to the fact that CA90 is additive, the active cell can be interpreted as one bit flip of error introduced to some random initial short seed. Fig. 6 shows the evolution of the considered configuration for the first 65 steps. Red rectangles in Fig. 6 highlight the steps of the form 2^j. At these steps, there are only two active cells. The behavior of the configuration with the single active cell explains both why for a small number of bit flips in Fig. 5 the normalized Hamming distance approximately doubled after the first step and why the distance reset for every step of the form 2^j.

Given the bit error rate (BER; number of bit flips) in the short seed (p_{bf}), we can calculate the BER after CA90 expansion (denoted as p_{CA}) for steps of the form 2^j as follows:

$$p_{\text{CA}} = 2p_{\text{bf}}^2(1-p_{\text{bf}}) + 2p_{\text{bf}}(1-p_{\text{bf}})^2 = 2p_{\text{bf}}(1-p_{\text{bf}}). \quad (12)$$

The intuition here is that due to the local interactions of CA, it is enough to only consider cases as in Fig. 1. In particular, we are only interested in cases, which result in active cells at the next step. There are only four such cases: two with two active cells and two with one active cell, enumerated in (12).

Fig. 7 plots the analytical p_{CA} according to (12) against the empirical one obtained in numerical simulations, and we see that the curves match.

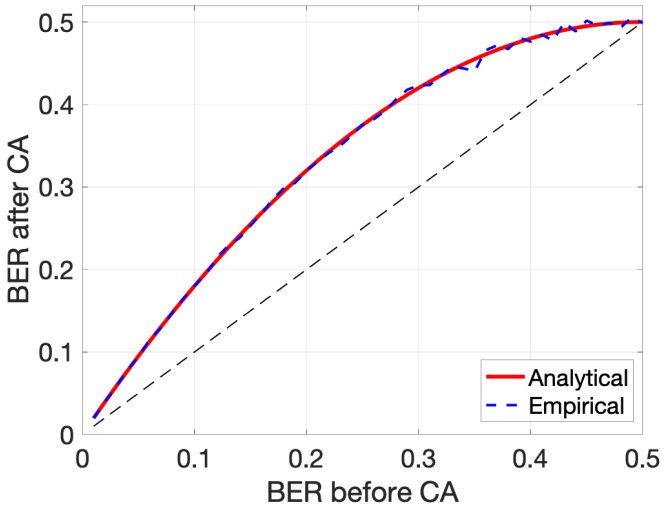


Fig. 7. Expected BER for CA90 steps of the form 2^j against the BER in the short seeds. The solid line is the analytical calculation, while the dashed line was measured empirically. The empirical results were averaged over ten simulations.

IV. Experimental Demonstration of CA90 Expansion for Collective-State Computing

This section focuses on using CA90 expansion for RC and VSAs. In several scenarios, we provide empirical evidence that expanded vectors obtained via CA90 computations are functionally equivalent to i.i.d. random vectors.[10] The code for reproducing the results of the experiments is available as the Supplementary Materials to this article.

A. Nearest Neighbor Search in Item Memories

One potential application of CA90 expansion of vectors will be for "on the fly" generation of item memories as used in RC and VSAs. An item memory is used to decode the output of a collective-state computation, where often the nearest neighbor to a query vector within the item memory is to be found. As mentioned before, when there is noise in the query vector, the outcome of the search may not always be correct. Therefore, we explored the accuracy of the nearest neighbor search when the query vector was significantly distorted by

[10]In the scope of this article by random vectors, we mean vectors generated with the use of a standard pseudorandom number generator. Thus, strictly speaking, they should be called pseudorandom vectors, but the term random is used to oppose them to the vectors obtained with CA90 computations.

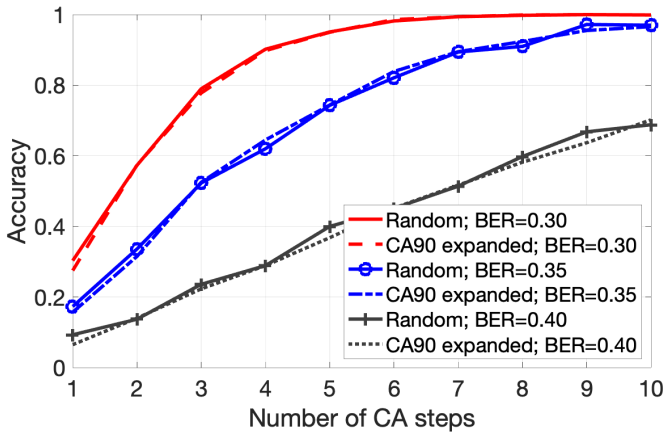


Fig. 8. Usage of i.i.d. random vectors against the CA90 expanded representations in the item memory. The figure reports the accuracy of the nearest neighbor search where a query was a noisy version of one of the vectors stored in the item memory. The noise was introduced in the form of bit flips. Three different values of BERs were simulated ($\{0.30, 0.35, 0.40\}$). The dimensionality of the initial short seeds was set to $N = 23$. The size of the item memory was set to 100. The reported values were averaged over 1000 simulations randomizing initial short seeds, random item memories, and noise added to queries.

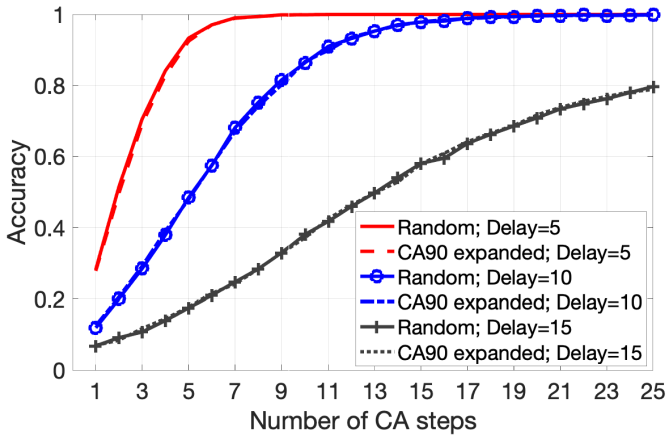


Fig. 9. Usage of i.i.d. random vectors against the CA90 expanded representations in the memory buffer task; $D = 27$ in the experiments. The figure reports the accuracy of the correct recall of symbols for three different values of delay ($\{5, 10, 15\}$). The dimensionality of the initial short seeds was set to $N = 37$. The reported values were averaged over ten simulations randomizing initial short seeds, random item memories, and traces of symbols to be memorized.

noise. We compare two item memories: one with i.i.d. random vectors and the other with CA90 expanded vectors where only initial short seeds ($N = 23$) were i.i.d. random. Fig. 8 shows the accuracy results of simulation experiments. The item memory with vectors based on CA90 expansion demonstrated the same accuracy as the item memory with fully i.i.d. random vectors.

B. Memory Buffer

Next, we demonstrate the use of CA90 expanded representations in the memory buffer task described in Section II-A2. These experiments were done with integer echo state networks. In these experiments, we measured the accuracy of recall from the memory buffer when the item memory was created from initial short seeds ($N = 37$) by concatenating the results of CA90 computations for several steps so that $K = NL$. As a benchmark, we used an item memory with i.i.d. random vectors of matching dimensionality. Three different values of delay were considered: $\{5, 10, 15\}$. The results are reported in Fig. 9. As expected, we observe that increasing the dimensionality of the memory buffer increased the accuracy of the recall. The main point, however, is that the memory buffer made from CA90 expanded representations demonstrates the same accuracy as the memory buffer made from i.i.d. random vectors.

C. Resonator Network Factoring in the Errorless Case

To further assess the quality of vectors obtained via the results of CA90 computations, we also examined their use in the resonator network [42]. Please see Section II-A3 and Section S.3 in the Supplementary Materials for details of the resonator network.

It is important to emphasize that due to the preservation of the binding operation by CA90, multiple aspects of the resonator network can benefit from CA90 expansion. Both the composite input vector and the factor item memories do not have to be memorized explicitly, but rather can be expanded from seeds. The factorization process is computed at each level of CA90 expansion, with the vector dimensions increasing by N for each CA step. The outputs of the resonator network are collected and compared to the ground truth and averaged over many randomized simulation experiments. Fig. 10 presents the average accuracies (left column) and the average iterations until convergence (right column) for three different dimensionalities of initial short seeds $\{100, 200, 300\}$ and three sizes of factor item memories $\{8, 16, 32\}$. The number of factors was set to four. The simulations considered the first 100 steps of CA90, which was less than the randomization period (1023) of the shortest seed ($N = 100$).

The performance of the resonator network was as expected. For a given dimensionality of short seed and item memory size, the average accuracy increased with the increased number of CA90 steps—as practically it means using vectors of larger dimensionality. The number of iterations in contrast decreased for larger vectors. Importantly, there was no notable difference in the performance of the resonator network both in terms of the accuracy and number of iterations when operating with CA90 expanded representations. This further confirms that it is reasonable to use CA90 expanded representations in order to trade off memory for computation.

D. Resonator Network Factoring in the Case of Errors

In order to examine the capabilities of CA90 expanded representations when the initial short seeds were subject to errors, we performed simulations for two dimensionalities of initial short seeds ($N = 37$ and $N = 39$). Similar to the experiments in Fig. 10, we used the resonator network to reconstruct a randomly chosen combination of factors. The difference was that some bit flips were added to the initial short seed (i.e., vector to be factored) where the number of

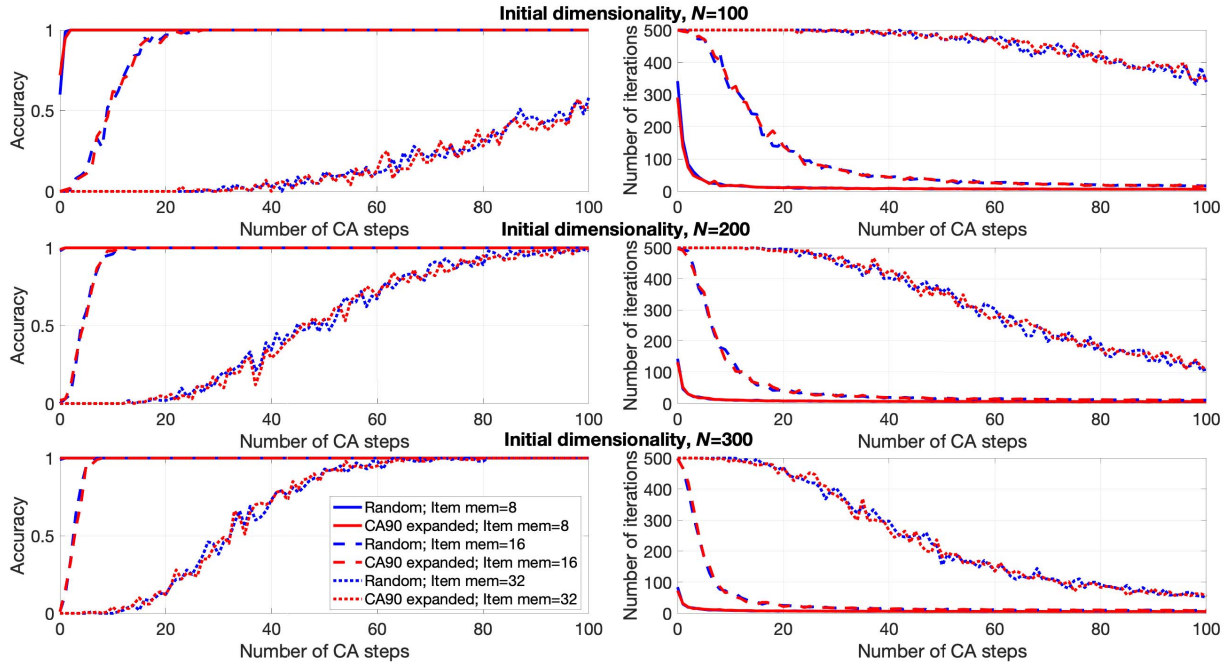


Fig. 10. Usage of random vectors against CA90 expanded representations in the resonator network. Left column reports the average accuracies, while right column reports the average number of iterations until convergence. The maximal number of iterations was set to 500. The dimensionality of the initial short seeds varied between $\{100, 200, 300\}$. The evolution of CA90 is reported for the first 100 steps. The size of an individual item memory varied between $\{8, 16, 32\}$. The number of factors was fixed to four. The reported values were averaged over 100 simulations randomizing initial short seeds.

bit flips was in the range $[0, 5]$ with step 1. The results are reported in Fig. 11. To minimize the noise introduced by CA90 computations, we only used steps (x-axis in Fig. 11) of the form 2^j (cf. Fig. 6) to expand the dimensionality. Columns in Fig. 11 correspond to different amounts of information carried by the vector to be factored.

The experiments were done for two configurations of the resonator network: with three factors (dashed lines) and with four factors (solid lines). Clearly, when given the same conditions, the resonator network with three factors outperforms the one with four factors. This observation is in line with the expected behavior of the resonator network. It should be noted, however, that the resonator network with three factors requires larger item memories in order to store the same amount of information. For example, for 16.00 bits in the case of four factors, the size of individual item memory was 16, while in the case of three factors, it was 40, i.e., the resonator network with three factors required about 2.5 times more memory. Thus, the use of a resonator network with fewer factors results in a better performance, but it requires more memory to be allocated.

We also see that even in the absence of errors (BER $= 0.00$), the accuracy is not perfect when a vector carries a lot of information because we are limited by the capacity of the resonator network—which does fail at factorization when the size of the factorization problem is too large. For example, for 16.00 bits, none of the expanded dimensionalities reached the perfect accuracy as opposed to the other two cases. Naturally, the inclusion of errors hurts the accuracy, but the performance degradation is gradual.

When comparing the performance of the resonator networks for the expanded vectors using all 21 CA90 steps, we made a counterintuitive observation that the performance for $N = 37$ is better despite shorter vectors and higher BERs. Recall from Fig. 4 that the chosen grid sizes have different randomization periods: 87 381 and 4095 for $N = 37$ and $N = 39$, respectively. The longer randomization period for $N = 37$ means that the use of $N = 37$ provides more randomness for a large number of CA90 steps. This is the main reason for the counterintuitive observation that the use of shorter seed at higher BER resulted in a better performance. When considering only the steps of the form 2^j, the corresponding randomization period results in about 16.41 and 12.00. These are exactly the values for which the performance of the resonator networks starts to saturate since concatenating additional dimensions after the randomization period stops adding extra randomness.

V. Discussion

A. Summary of Our Results

The use of CA computations for the generation of random numbers is not new, for instance, a seminal work [55] has proposed to generate random sequences with CA with rule 30.[11] Numerous studies followed on building CA-based pseudorandom number generators, e.g., [56] (see [57] for a recent overview of this work, some of it specifically investigating CA90). Here, we focused on how collective-state computing can benefit from the randomness produced by CA90.

[11]CA with rule 30 is also used for random number generation in Wolfram Mathematica [17].

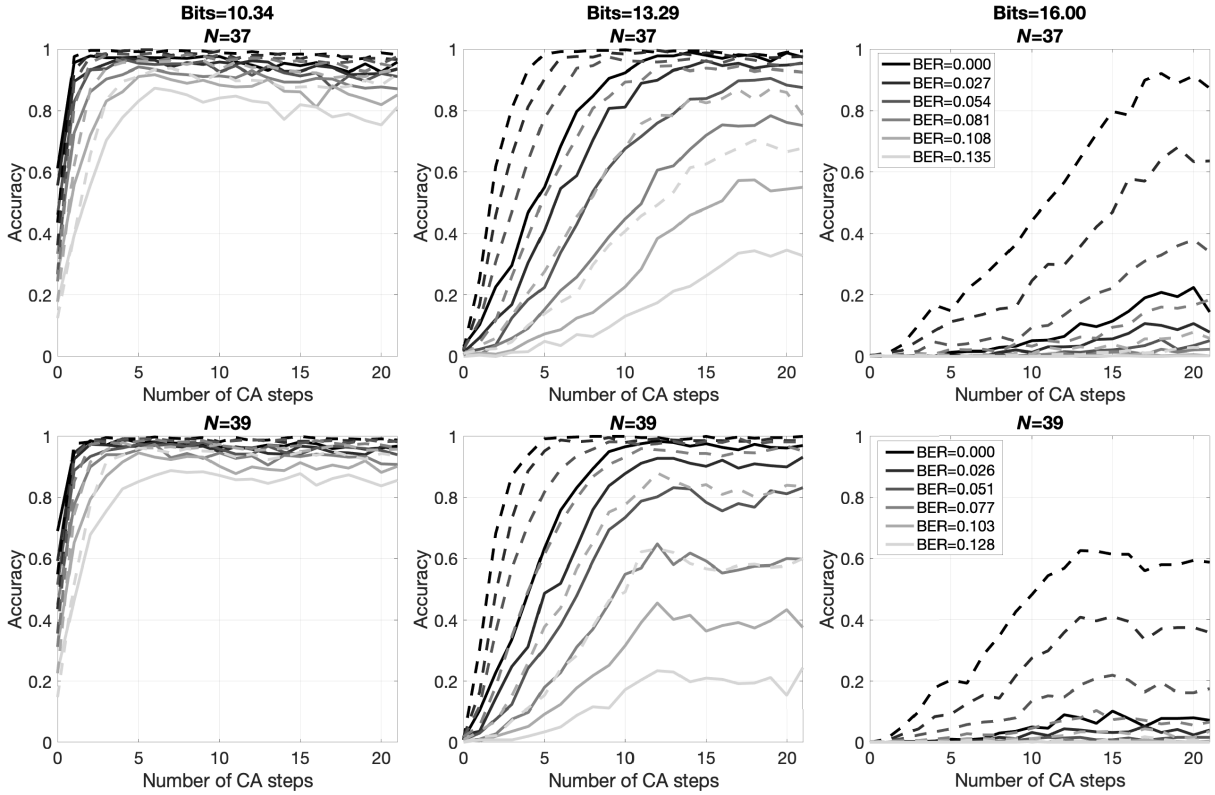


Fig. 11. Usage of the CA90 expanded representations in the resonator network in the case when the initial short seed might have errors. The upper panels report the case for $N = 37$ while the lower panels correspond to $N = 39$. The noise was introduced in the form of bit flips. The number of bit flips was in the range $[0, 5]$ with step 1. The legends show the corresponding BERs relative to N. The solid lines depict the resonator network with four factors, while the dashed lines depict the resonator network with three factors. Columns correspond to different amount of information carried by a vector, which was determined by the size of an item memory for one factor. The sizes of item memories for resonator networks with three and four factors were set to approximately match each other in terms of amount of information. The reported values were averaged over 100 simulations randomizing initial short seeds as well as introduced bit flips.

Our results are based on a key observation that collective-state computing typically relies on high-dimensional random representations, which have to be initially generated and stored for, e.g., being accessible by nearest neighbor searches during compute time. In many models, the representations are dense binary random patterns. Rather than storing the full random representations in memory (e.g., $\mathbf{W}^{\text{in}}$ in RC), we proposed to store just short seed patterns and to use CA90 for rematerializing the full random representations when required. The usage of CA90 expanded high-dimensional representations was demonstrated in the context of RC and VSAs. Our results provided empirical evidence that the expansion of representations on-demand (rematerialization solution) is functionally equivalent to storing the full i.i.d. random representations in the item memory (memorization solution).

Specifically, we have shown that the randomization period of CA90 is closely connected to its periodic cycle length and depends on the size of the grid. We provided the exact relation between the grid size and the length of the randomization period. The general trend is that larger grid sizes yield longer randomization periods. However, period length depends on the number-theoretic properties of the grid size integer. In general, odd-numbered grid sizes have longer randomization periods than even-numbered ones. In particular, one should avoid grid sizes that are powers of two (2^j), as they have the shortest randomization period. The longest periods are obtained when the size of the grid is a prime number. Thus, given a memory constraint, it is best to choose the largest prime within the constraint.

We have also demonstrated that it is possible to use the expansion even in the presence of errors in the short seed patterns. Unfortunately, CA90 introduces additional errors to the ones present in the seed pattern, so the error rate after CA90 increases. The distribution of introduced errors is, however, not uniform—some of the steps introduce more errors than the others. We have shown that the lowest amount of errors (cf. Fig. 5) is introduced for CA90 steps that are powers of two (2^j). Thus, in order to minimize the errors in the expanded representation, one should use only steps of the form 2^j. This, of course, limits the possibilities for expansion as practically not that many steps of the form 2^j can be computed (e.g., we used up to 20 in the experiments).

B. Related Work

1) Combining RC, VSAs, and CA: It has been demonstrated recently [5], [40] that echo state networks [39], an instance of RC, can be formulated in the VSA formalism. CA has been first introduced to RC and VSA models for expanding

low-dimensional feature representations into high-dimensional representations to improve classification [18], [47]. Due to the local interactions in CA, the evolution of the initial state (i.e., low-dimensional representation) over several steps produces a richer and higher dimensional representation of features while preserving similarity. This method was applied to activation patterns from neural networks [47] and to manually extracted features [58]. The expanded representations were able to improve classification results for natural [47] and medical [59] images.

Works [18], [47] suggested that the expanded representations could be seen as a reservoir, but they have not studied its memorization capabilities in detail. The characteristics of a formed memory were studied in [48]. Later, work [49] proposed to form a reservoir using a pair of CA by first evolving the initial state for several steps with one CA and then continuing the computations with some other CA rule. Similarly, works [51], [52] investigated nonuniform binary CA [51] and CA rules with larger neighborhoods [52] (five) and more states (three, four, and five).

Similar to these works, our approach also employs CA to expand the dimension of representations. However, we have applied expansion not to feature vectors, but just to i.i.d. random seed patterns. All we need is the property of CA90 that the resulting high-dimensional vectors are still pseudo-orthogonal. In our study, similarity preservation is only required if the random seed patterns contain errors.

Our work is most directly inspired by [16] and [50] who both used CA to expand item memory with short [16] or long [50] i.i.d. random seed patterns. In [16], the expansion was done with the CA30 rule, which is known to exhibit chaotic behavior. Here, as in [50], we used the CA90 rule.[12]

For collective-state computing, CA90 has the great advantage that it distributes over the binding and the cyclic shift operation. We have seen this advantage in action when studying the resonator network in Section IV-C. Since CA90 distributes over the binding operation, it was possible to expand the collective-state (i.e., the input vector (7) with factors) on-demand during the factorization. Going beyond [50] and [60], we also systematically explored the randomization properties of CA90, such as the length of the randomization period. Moreover, none of the previous work has studied the randomization behavior of CA90 in the presence of errors in the initial seed.

2) Other Computation Methods That Use Randomness: A complementary approach of computing with randomness is sampling-based computation [61]. This approach differs fundamentally from collective-state computing, which exploits a concentration of mass phenomenon of random patterns making them pseudo-orthogonal. Once generated, a fixed set of random patterns can serve as unique and well distinguishable identifiers for handling variables and values during compute time. In contrast, in the sampling-based computation, each compute step produces independent randomness to provide good mixing properties. Good mixing ensures that even a small set of samples is representative for the entire probability distribution and, therefore, constitutes a compact, faithful representation (see [62, Ch. 29]). We should add that the "frozen" randomness in VSA can be used to form different types of compact representation of probability distributions. For example, a combination of binding and bundling can constitute compact representations of large histograms describing the n-gram statistics of languages [30]. The advantage of such a representation is that it is a vector of the same fixed dimension as the atomic random patterns, somewhat independent of the number of nonzero entries in the histogram.

[12]Note that McDonald and Davis [60] extended [50] by studying other elementary rules and their suitability for cloning VSA vectors.

C. Future Work

1) Potential for Hardware Implementation: The space–time or memory–computation tradeoff introduced by the inclusion of CA can be used to optimize the implementation of collective-state computations in hardware. Of course, this optimization depends on the context of a computational problem and a particular hardware platform, which is outside the scope of this article.

Furthermore, despite the fact that the use of CA90 incurs additional computational costs, it enables highly energy-efficient collective-state computing models. For example, recently, an ASIC hardware implementation of an RC system with the proposed CA90 expansion has been reported [63]. This work reports that the CA90 expansion improved the energy efficiency by $4.8\times$ over state-of-the-art implementations. The reason for this is that the use of CA90 allows to flexibly choose the operation point between the two (suboptimal extremes): large leakage power when storing all randomly generated parts of the model explicitly (no CA90 expansion) versus large dynamic power due to many CA90 computational steps (with CA90 expansion when N is very small relative to K). All told, we expect the CA90 expansion to become a standard primitive for designing efficient hardware implementations of collective-state computing models.

The optimized hardware implementation of models we described involves another interesting topic for future research, the question how to parallelize the computation of CA90. While the implementation of CA90 in FPGA is quite straightforward, see (9), the implementation with neural networks and on neuromorphic hardware [64] is still an open problem.

2) Integration of CA Computations in Neural Associative Memories: Another interesting future direction is to investigate how associative memories [65] can trade off synaptic memory with neural computation implementing the CA. Such CA-based approaches could be compared to other suggestions in the literature how to replace memory by computation, e.g., [66].

ACKNOWLEDGMENT

The authors would like to thank the members of the Redwood Center for Theoretical Neuroscience and the Berkeley Wireless Research Center for stimulating discussions. They would also like to thank Evgeny Osipov and Abbas Rahimi for fruitful discussions on the potential role of cellular automata in vector symbolic architectures, which inspired the current work.

REFERENCES

[1] G. Csaba and W. Porod, "Coupled oscillators for computing: A review and perspective," *Appl. Phys. Rev.*, vol. 7, no. 1, 2020, Art. no. 011302.

[2] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Comput.*, vol. 14, no. 11, pp. 2531–2560, Nov. 2002.

[3] H. Jaeger, "Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the echo state network approach," German Nat. Res. Center Inf. Technol., Bonn, Germany, Tech. Rep. 159, 2002.

[4] A. Rodan and P. Tino, "Minimum complexity echo state network," *IEEE Trans. Neural Netw.*, vol. 22, no. 1, pp. 131–144, Jan. 2011.

[5] E. P. Frady, D. Kleyko, and F. T. Sommer, "A theory of sequence indexing and working memory in recurrent neural networks," *Neural Comput.*, vol. 30, no. 6, pp. 1449–1513, 2018.

[6] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci. USA*, vol. 79, no. 8, pp. 2554–2558, 1982.

[7] J. J. Hopfield and D. W. Tank, "'Neural' computation of decisions in optimization problems," *Biol. Cybern.*, vol. 52, no. 3, pp. 141–152, 1985.

[8] D. Achlioptas, "Database-friendly random projections: Johnson-Lindenstrauss with binary coins," *J. Comput. Syst. Sci.*, vol. 66, no. 4, pp. 671–687, 2003.

[9] D. L. Donoho, "Compressed sensing," *IEEE Trans. Inf. Theory*, vol. 52, no. 4, pp. 1289–1306, Apr. 2006.

[10] A. Amini and F. Marvasti, "Deterministic construction of binary, bipolar, and ternary compressed sensing matrices," *IEEE Trans. Inf. Theory*, vol. 57, no. 4, pp. 2360–2370, Apr. 2011.

[11] B. Igelnik and Y.-H. Pao, "Stochastic choice of basis functions in adaptive function approximation and the functional-link net," *IEEE Trans. Neural Netw.*, vol. 6, no. 6, pp. 1320–1329, Nov. 1995.

[12] D. Kleyko, M. Kheffache, E. P. Frady, U. Wiklund, and E. Osipov, "Density encoding enables resource-efficient randomly connected neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 8, pp. 3777–3783, Aug. 2021.

[13] T. A. Plate, "Holographic reduced representations," *IEEE Trans. Neural Netw.*, vol. 6, no. 3, pp. 623–641, May 1995.

[14] D. A. Rachkovskij, "Representation and processing of structures with binary sparse distributed codes," *IEEE Trans. Knowl. Data Eng.*, vol. 13, no. 2, pp. 261–276, Mar./Apr. 2001.

[15] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognit. Comput.*, vol. 1, no. 2, pp. 139–159, Oct. 2009.

[16] M. Schmuck, L. Benini, and A. Rahimi, "Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 4, pp. 1–25, Dec. 2019.

[17] S. Wolfram, *A New Kind of Science*. Champaign, IL, USA: Wolfram Media, 2002.

[18] O. Yilmaz, "Symbolic computation using cellular automata-based hyperdimensional computing," *Neural Comput.*, vol. 27, no. 12, pp. 2661–2692, 2015.

[19] S. I. Gallant and T. W. Okaywe, "Representing objects, relations, and sequences," *Neural Comput.*, vol. 25, no. 8, pp. 2038–2078, 2013.

[20] A. Rahimi *et al.*, "High-dimensional computing as a nanoscalable paradigm," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 9, pp. 2508–2521, Sep. 2017.

[21] E. P. Frady, D. Kleyko, and F. T. Sommer, "Variable binding for sparse distributed representations: Theory and applications," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Sep. 3, 2021, doi: 10.1109/TNNLS.2021.3105949.

[22] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, nos. 1–3, pp. 489–501, Dec. 2006.

[23] P. Kanerva, "Fully distributed representation," in *Proc. Real World Computing Symp. (RWC)*. 1997, pp. 358–365.

[24] R. W. Gayler, "Multiplicative binding, representation operators & analogy," in *Advances in Analogy Research: Integration of Theory and Data From the Cognitive, Computational, and Neural Sciences*, D. Gentner, K. J. Holyoak, and B. N. Kokinov, Eds. Sofia, Bulgaria: New Bulgarian Univ., 1998, pp. 1–4.

[25] D. Kleyko, E. Osipov, A. Senior, A. I. Khan, and Y. A. Şekerciogğlu, "Holographic graph neuron: A bioinspired architecture for pattern processing," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 6, pp. 1250–1262, Jun. 2017.

[26] O. J. Räsänen and J. P. Saarinen, "Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 9, pp. 1878–1889, Sep. 2016.

[27] A. Rosato, M. Panella, and D. Kleyko, "Hyperdimensional computing for efficient distributed classification with randomized neural networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–10.

[28] C. Diao, D. Kleyko, J. M. Rabaey, and B. A. Olshausen, "Generalized learning vector quantization for classification in randomized neural networks and hyperdimensional computing," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–9.

[29] D. Kleyko *et al.*, "Vector symbolic architectures as a computing framework for nanoscale hardware," 2021, *arXiv:2106.05268*. [Online]. Available: http://arxiv.org/abs/2106.05268

[30] A. Joshi, J. T. Halseth, and P. Kanerva, "Language geometry using random indexing," in *Proc. Int. Symp. Quantum Interact.* Cham, Switzerland: Springer, 2016, pp. 265–274.

[31] E. Osipov, D. Kleyko, and A. Legalov, "Associative synthesis of finite state automata model of a controlled object with hyperdimensional computing," in *Proc. 43rd Annu. Conf. IEEE Ind. Electron. Soc. (IECON)*, Oct. 2017, pp. 3276–3281.

[32] T. Yerxa, A. Anderson, and E. Weiss, "The hyperdimensional stack machine," in *Proc. Cogn. Comput., Merging Concepts Hardw.*, Hannover, Germany, Dec. 2018, pp. 1–2.

[33] D. V. Pashchenko, D. A. Trokoz, A. I. Martyshkin, M. P. Sinev, and B. L. Svistunov, "Search for a substring of characters using the theory of non-deterministic finite automata and vector-character architecture," *Bull. Electr. Eng. Informat.*, vol. 9, no. 3, pp. 1238–1250, Jun. 2020.

[34] D. Kleyko, E. Osipov, and R. W. Gayler, "Recognizing permuted words with vector symbolic architectures: A Cambridge test for machines," *Proc. Comput. Sci.*, vol. 88, pp. 169–175, Dec. 2016.

[35] D. V. Buonomano and W. Maass, "State-dependent computations: Spatiotemporal processing in cortical networks," *Nature Rev. Neurosci.*, vol. 10, no. 2, pp. 113–125, Feb. 2009.

[36] D. Kleyko, A. Rosato, E. Paxon Frady, M. Panella, and F. T. Sommer, "Perceptron theory for predicting the accuracy of neural networks," 2020, *arXiv:2012.07881*. [Online]. Available: http://arxiv.org/abs/2012.07881

[37] T. A. Plate, "Holographic recurrent networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 1992, pp. 34–41.

[38] T. A. Plate, *Holographic Reduced Representation: Distributed Representation for Cognitive Structures*. Stanford, CA, USA: CSLI, 2003.

[39] M. Lukosevicius, "A practical guide to applying echo state networks," in *Neural Networks: Tricks Trade* (Lecture Notes in Computer Science), vol. 7700. Berlin, Germany: Springer, 2012, pp. 659–686.

[40] D. Kleyko, E. P. Frady, M. Kheffache, and E. Osipov, "Integer echo state networks: Efficient reservoir computing for digital hardware," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Dec. 22, 2021, doi: 10.1109/TNNLS.2020.3043309.

[41] A. Thomas, S. Dasgupta, and T. Rosing, "A theoretical perspective on hyperdimensional computing," *J. Artif. Intell. Res.*, vol. 72, pp. 215–249, Oct. 2021.

[42] E. P. Frady, S. J. Kent, B. A. Olshausen, and F. T. Sommer, "Resonator networks, 1: An efficient solution for factoring high-dimensional, distributed representations of data structures," *Neural Comput.*, vol. 32, no. 12, pp. 2311–2331, Dec. 2020.

[43] S. J. Kent, E. P. Frady, F. T. Sommer, and B. A. Olshausen, "Resonator networks, 2: Factorization performance and capacity compared to optimization-based methods," *Neural Comput.*, vol. 32, no. 12, pp. 2332–2388, Dec. 2020.

[44] C. G. Langton, "Computation at the edge of chaos: Phase transitions and emergent computation," *Phys. D, Nonlinear Phenomena*, vol. 42, nos. 1–3, pp. 12–37, 1990.

[45] M. Cook, "Universality in elementary cellular automata," *Complex Syst.*, vol. 15, no. 1, pp. 1–40, 2004.

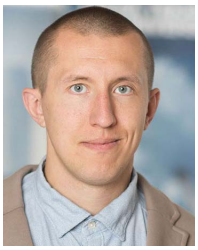
[46] R. W. Gayler, "Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience," in *Proc. Joint Int. Conf. Cognit. Sci. (ICCS/ASCS)*, 2003, pp. 133–138.

[47] O. Yilmaz, "Machine learning using cellular automata based feature expansion and reservoir computing," *J. Cellular Automata*, vol. 10, nos. 5–6, pp. 435–472, 2015.

[48] S. Nichele and A. Molund, "Deep learning with cellular automaton-based reservoir computing," *Complex Syst.*, vol. 26, no. 4, pp. 319–339, 2017.

[49] N. McDonald, "Reservoir computing & extreme learning machines using pairs of cellular automata rules," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 2429–2436.

[50] D. Kleyko and E. Osipov, "No two brains are alike: Cloning a hyperdimensional associative memory using cellular automata computations," in *Biologically Inspired Cognitive Architectures (BICA)* (Advances in Intelligent Systems and Computing), vol. 636. Cham, Switzerland: Springer, 2017, pp. 91–100.
[51] S. Nichele and M. Gundersen, "Reservoir computing using non-uniform binary cellular automata," *Complex Syst.*, vol. 26, no. 3, pp. 225–245, 2017.
[52] N. Babson and C. Teuscher, "Reservoir computing with complex cellular automata," *Complex Syst.*, vol. 28, no. 4, pp. 433–455, Dec. 2019.
[53] J. Daugman, "The importance of being random: Statistical principles of iris recognition," *Pattern Recognit.*, vol. 36, no. 2, pp. 279–291, 2003.
[54] O. Martin, A. M. Odlyzko, and S. Wolfram, "Algebraic properties of cellular automata," *Commun. Math. Phys.*, vol. 93, no. 2, pp. 219–258, Jun. 1984.
[55] S. Wolfram, "Random sequence generation by cellular automata," *Adv. Appl. Math.*, vol. 7, no. 2, pp. 123–169, 1986.
[56] R. Santoro, S. Roy, and O. Sentieys, "Search for optimal five-neighbor FPGA-based cellular automata random number generators," in *Proc. Int. Symp. Signals, Syst. Electron.*, Jul. 2007, pp. 343–346.
[57] M. Dascalu, "Cellular automata and randomization: A structural overview," in *From Natural to Artificial Intelligence—Algorithms and Applications*, R. Lopez-Ruiz, Ed. Rijeka, Croatia: InTechOpen, 2018, ch. 9, pp. 165–183.
[58] N. Karvonen, J. Nilsson, D. Kleyko, and L. L. Jimenez, "Low-power classification using FPGA—An approach based on cellular automata, neural networks, and hyperdimensional computing," in *Proc. 18th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2019, pp. 370–375.
[59] D. Kleyko, S. Khan, E. Osipov, and S.-P. Yong, "Modality classification of medical images with distributed representations based on cellular automata reservoir computing," in *Proc. IEEE 14th Int. Symp. Biomed. Imag. (ISBI)*, Apr. 2017, pp. 1053–1056.
[60] N. McDonald and R. Davis, "Complete & orthogonal replication of hyperdimensional memory via elementary cellular automata," in *Proc. Unconventional Comput. Natural Comput. (UCNC)*, Tokyo, Japan, Jun. 2019, p. 1.
[61] G. Orbán, P. Berkes, J. Fiser, and M. Lengyel, "Neural variability and sampling-based probabilistic representations in the visual cortex," *Neuron*, vol. 92, no. 2, pp. 530–543, Oct. 2016.
[62] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2003.
[63] A. Menon, D. Sun, M. Aristio, H. Liew, K. Lee, and J. M. Rabaey, "A highly energy-efficient hyperdimensional computing processor for wearable multi-modal classification," in *Proc. IEEE Biomed. Circuits Syst. Conf. (BioCAS)*, Oct. 2021, pp. 1–4.
[64] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.
[65] V. I. Gritsenko, D. A. Rachkovskij, A. A. Frolov, R. Gayler, D. Kleyko, and E. Osipov, "Neural distributed autoassociative memories: A survey," *Cybern. Comput. Eng.*, vol. 2, no. 188, pp. 5–35, 2017.
[66] A. Knoblauch, "Zip nets: Efficient associative computation with binary synapses," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2010, pp. 1–8.



Denis Kleyko (Member, IEEE) received the B.S. degree (Hons.) in telecommunication systems and the M.S. degree (Hons.) in information systems from the Siberian State University of Telecommunications and Information Sciences, Novosibirsk, Russia, in 2011 and 2013, respectively, and the Ph.D. degree in computer science from the Luleå University of Technology, Luleå, Sweden, in 2018.

He is currently a Post-Doctoral Researcher on a joint appointment between the Redwood Center for Theoretical Neuroscience, University of California at Berkeley, Berkeley, CA, USA, and the Intelligent Systems Laboratory, Research Institutes of Sweden, Kista, Sweden. His current research interests include machine learning, reservoir computing, and vector symbolic architectures/hyperdimensional computing.

Edward Paxon Frady received the B.S. degree in computation and neural systems from the California Institute of Technology, Pasadena, CA, USA, in 2008, and the Ph.D. degree in neuroscience from the University of California at San Diego, La Jolla, CA, USA, in 2014.

He is currently a Researcher in residence with the Neuromorphic Computing Laboratory, Intel Labs, Santa Clara, CA, USA, and a Visiting Scholar with the Redwood Center for Theoretical Neuroscience, University of California at Berkeley, Berkeley, CA, USA. His research interests include neuromorphic engineering, vector symbolic architectures/hyperdimensional computing, and machine learning.

Friedrich T. Sommer received the Diploma degree in physics from the University of Tuebingen, Tuebingen, Germany, in 1987, the Ph.D. degree from the University of Duesseldorf, Duesseldorf, Germany, in 1993, and the Habilitation degree in computer science from the University of Ulm, Ulm, Germany, in 2002.

He is currently a Researcher in residence with the Neuromorphic Computing Laboratory, Intel Labs, Santa Clara, CA, USA, and an Adjunct Professor with the Redwood Center for Theoretical Neuroscience, University of California at Berkeley, Berkeley, CA, USA. His research interests include neuromorphic engineering, vector symbolic architectures/hyperdimensional computing, and machine learning.