# Neural Latching Switch Circuits for temporally structured behavior

Alexis Dubreuil[1,*], Arthur Leblois[1], Rémi Monasson [2]

[1]Université de Bordeaux, CNRS, IMN, UMR 5293, Bordeaux, France.
[2]Ecole Normale Supérieure, PSL Research, CNRS UMR8023, Sorbonne Université, Paris, France.
[*] alexis.dubreuil@cnrs.fr

## Abstract

Elaborated temporal structures in behavior have been recognized as a hall-mark of high-level cognitive processes, such as planning or natural language. Systems neuroscience experiments in behaving animals have validated cell-assemblies as a fundamental piece of neural circuitry underlying lower-level cognitive processes. Although recent research has identified neural bases for certain temporally structured behaviors, it is still an open question whether a general-purpose circuit - akin to cell assemblies - supports such behaviors. Here we introduce Neural Latching Switch Circuits and show how they can be assembled as lego-bricks to create neural architectures suited for the implementation of *any* procedural behavior. By focusing on increasingly complex temporal structures and leveraging a combination of theoretical tools, we demonstrate how these circuits can be mapped onto the fly's head-direction system and propose quantitative testable predictions for identifying analogous structures in mammalian brains, for instance in cortical columns. By interpreting neural architectures as computing automata we reveal surprising relationships between behavior and computation.

# Introduction

Early works in psychology and neuroscience recognized that behavior, and especially high-level cognitive processes such as planning, language or reasoning rely on temporally structured sequences of mental events (1). These works were accompanied by speculations about potential underlying neural substrates, but these remained vague given the amount of experimental knowledge available at the time (2). Looking to propose such a neural substrate, one is facing two challenges.

The first one is to define relevant temporal structures to model behavior. Temporal structures have been introduced to model animal behavior under laboratory conditions, such as the swimming patterns of zebrafish larvae (3), locomotion of C. Elegans (4), courtship songs of birds (5) or grooming patterns of rodents (6). Another more systematic attempt of such a definition is Chomsky's generative grammar (7) which defined classes of languages organized in a hierarchy, with qualitatively different forms of temporal structure at each complexity level (8). However, the relevance of this classification for modeling behavior has been questioned (9; 10; 11).

A second challenge is to relate temporal structures to the anatomical and physiological properties of brain circuits. Oscillations in electrical activity for instance have been deemed important to support cognition (12). Another set of experimental results has demonstrated that the concept of attractor neural network is well suited to account for the emergence of elementary cognitive processes involved in working memory, spatial navigation or decision making (13). They formalize Hebb's ideas (14), with a synaptic connectivity structure characterized by positive feedback loops, allowing electrical activity to build-up among neurons. This leads to the activation of groups of neurons, or cell-assemblies, that encode behavioral variables such as the identity of a remembered image (15), a location in space (16) or a choice option (17). These attracting neural states in principle persist over arbitrarily long durations and therefore do not seem well suited to account for the emergence of high-level, temporally structured, cognitive processes. Systems neuroscience studies have focused on specific temporally structured behaviors (see e.g. (4; 6; 18; 19; 20; 21)), but whether or not a generic circuit motif, akin to cell-assemblies, sustains such behaviors remains to be explored (13; 22).

In order to address these two challenges, we trained artificial neural networks (**ANN**) (23) to perform a set of tasks exhibiting primitive temporal structures which have been proposed to underlie behavior (24). Analyzing these networks, we found an elementary neural circuit, the Neural Latching Switch Circuit (**NLSC**), that can be leveraged to compose efficient neural architectures for all tasks. This elementary circuit consists of a dictionary of cell-assemblies augmented with sets of gate neurons whose properties we characterized quantitatively. We first introduce the NLSC by considering simple behaviors and show how it can be mapped onto the head-direction system described in the fly's brain (25; 26). We then show how to assemble NLSC to implement complex behaviors studied in systems neuroscience, and conclude by modeling the sensory-motor hierarchy (27; 28), whereby interacting NLSC model cortical columns (29). To describe the relationship between neural architectures and the temporal structures they support, cell-assembly activations are naturally interpreted as automata states (30), with different forms of temporal structures mapping onto different automata abilities. Picturing neural networks as com-

puting machines manipulating symbols allows us to outline temporal structures fundamental to behavior.

## Scope of the work and relationship with previous works

In this study we trained artificial neural networks and reverse-engineer them (23). This allows us to describe under which conditions properties of NLSC emerge upon training networks on tasks with simple temporal structures (section 1). Later on, in sections 2 and 3, we limit the use artificial neural networks to putting forward neural architectures, composed of NLSC, that solve tasks with more complex temporal structures.

In order to assess the computational efficiency of the neural architectures, we introduced minimal models of binary neurons that capture the circuit properties of NLSC. This allows us to argue that NLSC-based neural architectures are efficient, in the sense that the number of neurons they require scales favorably with computational demand. To do so we extended previous works on the storage capacity of perceptrons and attractor neural networks that have been developed using tools from statistical physics of neural networks (SI Discussion 6.10.1). To propose a meaningful definition of computational demand and relate that to the temporal structures of the tasks we considered, we take inspiration from the correspondance between the hierarchy of generative grammar rules, studied in linguistic, and the associated hierarchy of automata, studied in computer science (8).

More specifically, the set of temporal structures on which we focus is a subset of those put forward by Dehaene and colleagues (24). This subset has been chosen so as to reflect the computational properties of NLSC-based architectures. We interpret NLSC as finite-state automata (Fig. 2a-d, section 1) and show how combining NLSC allows to augment finite-state automata (Fig. 2a-d) with an external memory (Fig. 3g) on which they can read and write (Fig. 4e) or move (Fig. 4l) (30). Such computational features are basic ingredients to Turing Machines (Fig. 5c), which despite their simplicity are able to implement any procedural behavior.

The relationship with Turing machines and the variety of tasks that can be solved by NLSC-based architectures (motor generation, sensory processing or cognitive tasks), lead us to hypothesize that NLSC is an essential piece of brain circuitry. We show that NLSC can be mapped onto the head direction system of the fly's brain (section 1). The example tasks we considered are inspired by tasks performed by animals in systems neuroscience experiments. When analyzing the associated neural architectures, we specifically focus on characterizing properties of gate neurons and their population structure. This allows us to highlight salient features that can guide experimental work aiming at identifying gate neurons and NLSC in mammalian brains and their cortical columns. The models we studied are limited in their biological realism, but allow nonetheless to relate to a variety of experimentally accessible measurements (Discussion, SI Discussion 6.10.3).

In discussion we contrast this new perspective on neural computation with other existing theoretical frameworks accounting for the neural implementation of procedural behaviors. In particular we show how it expands the computation through neural dynamics point of view (31) (SI Discus-

sion 6.10.1 and SI Fig. 1). We also discuss how state-updates through opening of gate neurons, either with external go cues or with oscillatory dynamics (section 4), contrasts with the more elaborated oscillatory dynamics postulated to underlie binding by synchrony.
We discuss the implications of the present work for the characterization of behavior through the lens of temporal structures. We propose that some of the temporal structures put forward by Dehaene and colleagues reflect different computational abilities associated with the neural substrate underlying behavior, providing a computationally meaningful definition of temporal structures.

In Supplementary Discussion we highlight relationships with previous modeling works focused on sequential behavior, for instance the works of Treves and colleagues about latching networks (32),(33), or works putting forward the role of gate-like neurons in triggering transitions between network states (34; 35; 21) (SI Discussion ). We also discuss the relationships with previous studies of population-structured neural networks and studies of continuous attractor networks. Finally, in SI Discussion we discuss the relationships between ancient and recent works treating neural networks as computing machines (30), for instance a work interpreting attractor states as machine states of finite-state automata (36), or a work using cell-assemblies for assembling Turing machines (37).

# 1 NLSC for behaviors with simple temporal structures

**Programming transitions between neural states using gate neurons**. Brain circuits do not simply implement stimulus-response reflexes, with inputs from external sensors directly transformed into motor actions executed by effectors. Rather, the effect of sensory inputs depends on the current internal state of the brain. In order to define a minimal form of temporal structure capturing this observation we considered a behavioral task designed to study the neural basis of sensory-motor transformations in rodents (38). In this task, one of two sensory stimuli is presented and after a variable delay period the animal is asked to respond to a go cue by performing one of two motor actions (Fig. 1a).

We modeled results of this experiment with a neural latching switch circuit (NLSC). It is composed of an attractor neural network with four cell-assemblies (Fig. 1b, blue) whose activation encodes the two sensory and the two motor states, below we refer to this part of the circuit as a dictionary of cell-assemblies. Similarly to latching switch in electrical circuits, the attracting property of cell-assemblies leads the circuit to latch, i.e. it remains in one of the states in the absence of inputs. Presentation of the go cue (Fig. 1b, green) allows for the activation of gate neurons through a modulatory input (Fig. 1b, yellow), leading the circuit to switch to a different state. To build NLSC, we first trained a rate network to implement the dictionary, then we added gate neurons, which are not recurrently connected with each others, and trained connections from dictionary to gate neurons and back to program task-relevant transitions (Method 5.2.1). The connectivity structure of the NLSC reflects its function, with positive feed-back loops in the dictionary supporting stable cell-assemblies (Fig. 1c, blue squares), and connectivity through gate neurons linking sensory and motor states (Fig. 1c, bottom). Throughout task execution, individual dictionary neurons show persistent activity selective to one of the four states, while gate neurons show non-linear mixed selectivity (39), responding only to conjunction of the go cue and one of the two

sensory states (Fig. 1d,e). Neural activity in the NLSC can also be described at the population level (SI Fig. 1a-d, Methods 5.2.3,5.4), with inputs from the go cue modulating the gain of gate neurons and controlling interactions between cell-assemblies (40).

Recordings of neural activity while rats perform this task showed that sensory and motor states are encoded in pre-motor cortex, and that thalamic neurons respond to presentations of the go cue (38). It would be interesting to see whether the current pre-motor cortical state can be decoded in the associated motor thalamus, with individual neurons exhibiting non-linear mixed selectivity as in Fig. 1e-right. This does not have to be the case and thalamic circuits could also simply relay the go cue. Indeed, by analyzing 50 ANN trained on this task without a priori architectural constraints (Method 5.2.2), we found that transitions from sensory to motor states can in principle be programmed by directly shaping connectivity in-between cell-assemblies of the dictionary, without resorting to gate neurons (Method 5.3.2, SI Fig. 2 g-j). In these trained ANN, a cell-assembly encoding a motor variable is correctly activated, e.g. C, if it receives more current than the other motor variable, e.g. D. These currents have two sources: direct currents coming from the preceding sensory cell-assembly activation ($83\%$ of the difference in received currents, SI Fig. 2h, top-right), and indirect currents coming from gate-like neurons transiently active at the go cue ($17\%$ of the difference in received currents, SI Fig. 2h, bottom-right). While these two network mechanisms are known to be able to implement transitions between neural patterns, modulating effective interactions between dictionary neurons through gate neurons, as in the NLSC (SI Fig. 1b,c), has been shown to be more robust for triggering many transitions (41).

**Hetero-associative connections through gate neurons is more efficient than direct connections between cell-assemblies**. Consistently, when training unconstrained ANNs to implement a permutation among 20 states, as illustrated in Fig.1f (Method 5.3.3), the difference in received currents onto the two motor cell-assemblies dropped at $22\%$ for direct currents and increased to $78\%$ for indirect currents (SI Fig. 2r). In Fig.1g we show the activity of a trained NLSC producing a sequence of 20 neural states. We focused on neural activation patterns of gate neurons in response to the go cue. Gate activation patterns were distributed across multiple neurons: each transition involved the activation of $129$ out of the $820$ gate neurons and each gate neuron was involved in $3$ transitions on average (SI Fig. 2l). In addition they were orthogonal to each others (SI Fig. 2m), as expected from the fact that they correspond to the imprinting of orthogonal dictionary states and that they should each point towards one of these orthogonal dictionary states. We noticed that in order to implement the 20 transitions, the number of gate neurons could be reduced from 820 to 82 without loss in performance (SI Fig. 2n). This lead us to wonder how many neurons are required to program transitions between dictionary states. In principle, having only 20 gate neurons, one per transition is feasible (41). But given that patterns of activation of gate neurons are distributed it could be possible to lower this number.

We resorted to previously developed mathematical techniques to estimate the storage capacity of attractor neural networks (Method 5.5), this allows us to estimate $N_{min}$, the minimal number of neurons that the NLSC needs to implement a permutation among $P$ dictionary states. To do so we treated the different types of connections of the NLSC as three perceptron problems (Fig. 1h). As for attractor neural networks we found this minimal number to scale as $\sqrt{P}$ (Fig. 1i). A total of

$N_{min} \simeq 10,000$ neurons, the approximate size of a cortical column, allows the NLSC to implement a permutation on $P \simeq 22,000$ dictionary states, with dictionary and gate patterns characterized by a coding level $f \simeq 0.003$, i.e. $0.3\%$ of dictionary or gate neurons are activated in each pattern of activity, and a fraction of activated synapses inside the dictionary, or in between dictionary and gates, of $0.15$ (SI Fig. 7e). By contrast, the network mechanism without gate neurons is less efficient (SI Fig. 7i), as expected from previous numerical estimates (41).

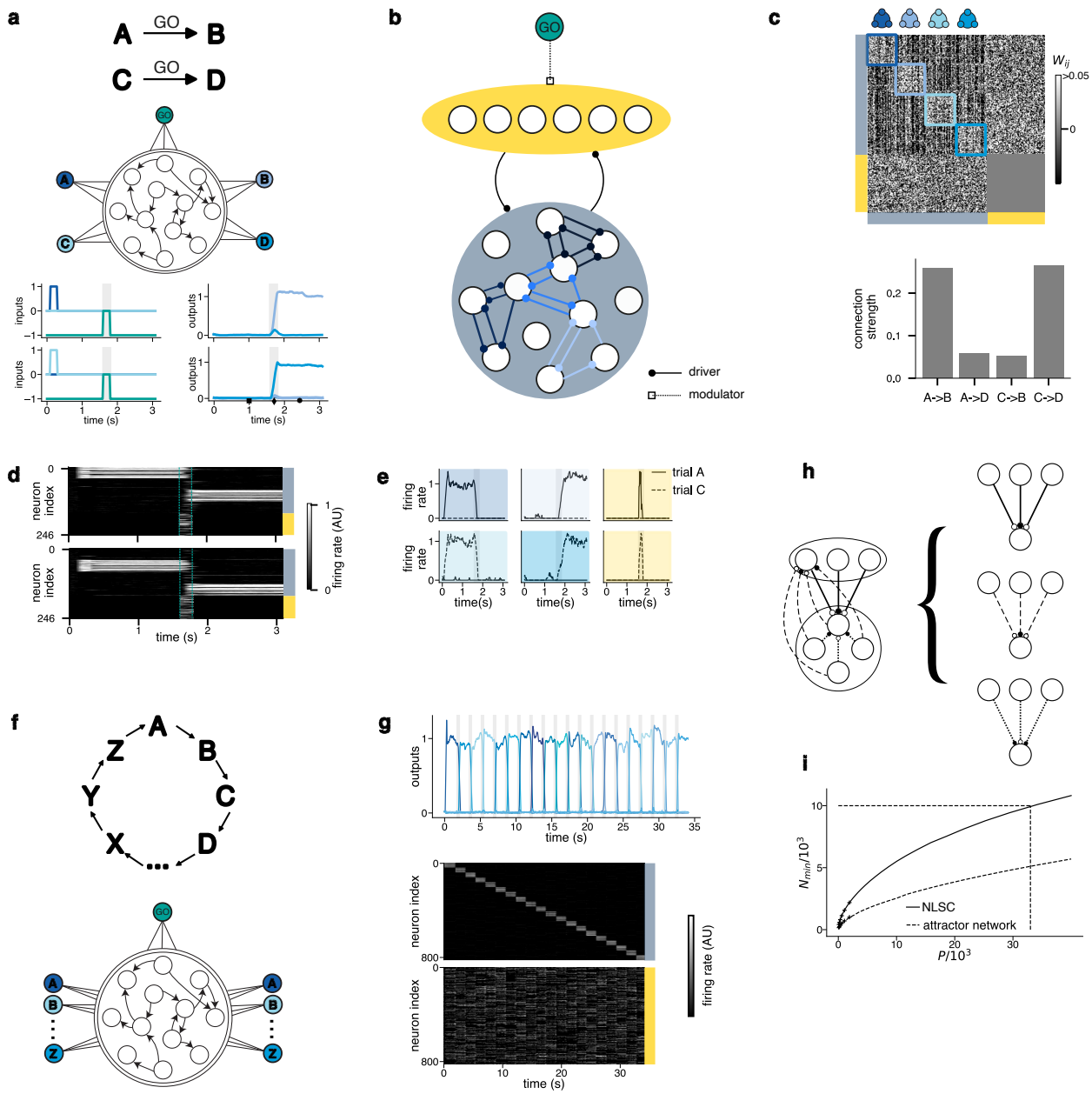


**Fig. 1:** ***Neural Latching Switch Circuit: programming transitions between cell-assembly activations with gate neurons.*** ***a****, Top: illustration of a minimal form of temporal structure. In this sensory-motor task, upon presentation of a go cue the system transits to a new state that depends only on the previous state. Bottom: ANN trained to perform the task.* ***b****, Cartoon of the NLSC. Circles represent neurons. Round and squared connections represent driving and modulatory inputs respectively. A dictionary, with positive feedback loops in its connectivity implement stable cell-assemblies, while gate neurons, under the modulatory control of a go cue, control transitions between cell-assemblies.* ***c****, Left: connectivity matrix of the trained NLSC. Right: connectivity through gate neurons, bar size for a transition $X \rightarrow Y$ represents the averaged product of the weights from cell-assembly X to gate neurons and the weights from gate to cell-assembly Y.* ***d****, Raster of neural activity in trials with A to B (top) and C to D (bottom) transitions.* ***e****, Firing rate of six neurons for the two transitions.* ***f****, Permutation of dictionary states.* ***g****, Sequential activity of an ANN trained to implement a permutation of state.* ***h****, Calculation of $N_{min}$, the minimal number of neurons for an architecture, relies on treating the dictionary and gate neurons as inputs and/or outputs of perceptrons.* ***i****, $N_{min}$ for an attractor neural network storing $P$ patterns, or a NLSC implementing a permutation among $P$ dictionary states.*

**NLSC as finite-state-automata**. The sensory-motor transformation task (Fig.1a), or the production of sequences of motor states (Fig.1h), can be modeled by finite-state-automata (**FSA**, Fig.2a). FSA describe machines programmed to carry simple tasks such as running an elevator. The computations they perform are depicted as a succession of machine states, with sensory inputs from the environment triggering state transitions and some of the states acting on the environment through activation of effectors (30) (SI Section 6.7.1). Interestingly FSA can also be used to define a simple form of temporally structured behavior, namely regular-languages, which constitutes the lowest complexity-level of Chomsky's hierarchy (SI Section 6.8). In this context, a FSA implements a machine, called a recognizer, that recognizes whether or not a presented string of symbols belongs to a given language (Fig.2b). The behavioral tasks we have thus far implemented with a NLSC are modeled by another specific type of finite-state-automata, transducers, which are characterized by a single sensory input playing the role of a clock, as in an electrical circuit running a traffic light.

To go beyond behaviors modeled by these simple FSA we now focus on the task of head-direction encoding, modeled by the FSA in Fig. 2c. Such a FSA is called a categorizer: all sequences of the two clockwise/counter-clockwise vestibular inputs (e.g. $++--++$, green dots Fig. 2e) are categorized into an angular position represented by one of the state of the automaton. We trained ANN to maintain a representation of their current angular position, and to update this position in response to activation of one of the vestibular velocity inputs (Fig. 2e, Method 5.3.4). We first trained unconstrained ANN on this task and confirmed that networks resorted to gate neurons to trigger transitions between cell-assemblies encoding head-directions (SI Fig. 3f,g). We then trained a NLSC and examined the emerging structure of activity in gate neurons. Examination of the patterns of activations (Fig. 2f green delimiters) for all transitions revealed that gate neurons segregated into two populations, one population of neurons active during clockwise turns and another one active during counter-clockwise turns (Fig. 2g, SI Fig. 3c,d,e). This maps well onto the two populations of the protocerebral bridge in the central complex of the fly's brain, with

neurons exhibiting non-linear mixed selectivity, mixing angular position and vestibular velocity in one direction (26). The emergence of these two populations of gate neurons can be understood as follows. The mixing should be done to have gate patterns that are linearly separable so that the connectivity from gate to dictionary neurons can be shaped to point towards the desired next dictionary pattern (SI Fig. 7l-n). Splitting gate neurons into two populations, each responding to a single vestibular cue, is a way to obtain linearly separable neural representations of position and velocity.

In order to better understand how NLSC can implement FSA, we explored another case with multiple external inputs (Fig. 2d,h,i), but this time their number $P_c$ was much higher than the number $P$ of dictionary states (while $P_c = 2$ and $P = 20$ in the head-direction example). Training a NLSC to implement all $P_c = 24$ permutations on a dictionary with $P = 4$ states required to correctly scale input connectivity weights with respect to dictionary-gates weights, as well as to increase the number of gate neurons. We found this time that gate neurons segregated into 4 populations (Fig. 2j-left, SI Fig. 3j-l, Method 5.3.4). In the head-direction case, external inputs were modulating gate neurons by controlling their gain (maintaining them below firing threshold) and dictionary neurons were driving them (42) (Fig. 2g-right). Here we observe the opposite with dictionary states controlling the gain of gate neurons and external inputs driving them (Fig. 2j-right). This other form of mixing is another way to obtain linearly separable gate patterns.

**Computational capacity of NLSC**. To understand when it might be better to structure gate neurons of the NLSC as in Fig. 2g or Fig. 2j, we compared values of $N_{min}$ for these two implementations, under different values of $P$ and $P_c$. Consistent with the organization emerging in ANN, we found that when $P_c \ll P$, splitting gate neurons into $P_c$ populations leads to smaller values of $N_{min}$ compared to splitting into $P$ populations. The converse was true for $P_c \gg P$ (Fig. 2k, dashed lines). This can be understood by considering the need to demix the two sources of inputs onto gate neurons and the need to use sparse-coded neural patterns to minimize ressource use (43) (SI Section 6.1.5).

Studies on attractor neural networks defined the notion of capacity of a neural architecture, which measures how the minimal amount of ressources (e.g. number of neurons) depends on computational demand (e.g. number of stable cell-assemblies) (44). We characterized the computational capacity of the NLSC by the minimal number $N_{min}$ of neurons required to implement a FSA specified by its number of machine states $P$, number of possible input symbols $P_c$ and fraction $\epsilon$ of existing transitions from one machine state among the $P_c$ possible ones (SI Section 6.1.3). The full line in Fig. 2k shows $N_{min}$ as a function of the number of input symbols $P_c$ for $P = 1000$ machine states and $\epsilon = 1$. Our analytical approach allows to characterize optimal neural architectures by their number of dictionary and gate neurons, number of gate populations, organisation in terms of driver and modulator inputs, coding levels of activity patterns in dictionary and gate neurons, and average fraction of active synapses between structures (SI Fig. 7j,k).

We illustrate this by considering NLSC optimized to implement head-direction systems. Head-direction systems have been described in various animal models (45; 25; 46), with different angular resolutions, as suggested by the observation of wider bumps of activity in zebrafish larvae compared to flies. For decreasing angular resolutions $\Delta\theta$ ($P = \frac{360}{\Delta\theta}$, $P_c = 2$, $\epsilon = 1$), $N_{min}$ goes from

193 to 805 for $\Delta\theta = 10°$ to $\Delta\theta = 1°$. The coding levels of dictionary and gate patterns decrease with decreasing resolution, the average connectivities remain constant, and a ratio of 1.8 between quantities for dictionary and gates is kept constant (Fig. 2l). It would be interesting to see whether the head-direction system of the zebrafish larvae (46) also relies on gate neurons, and to compare their properties with those of flies (25).

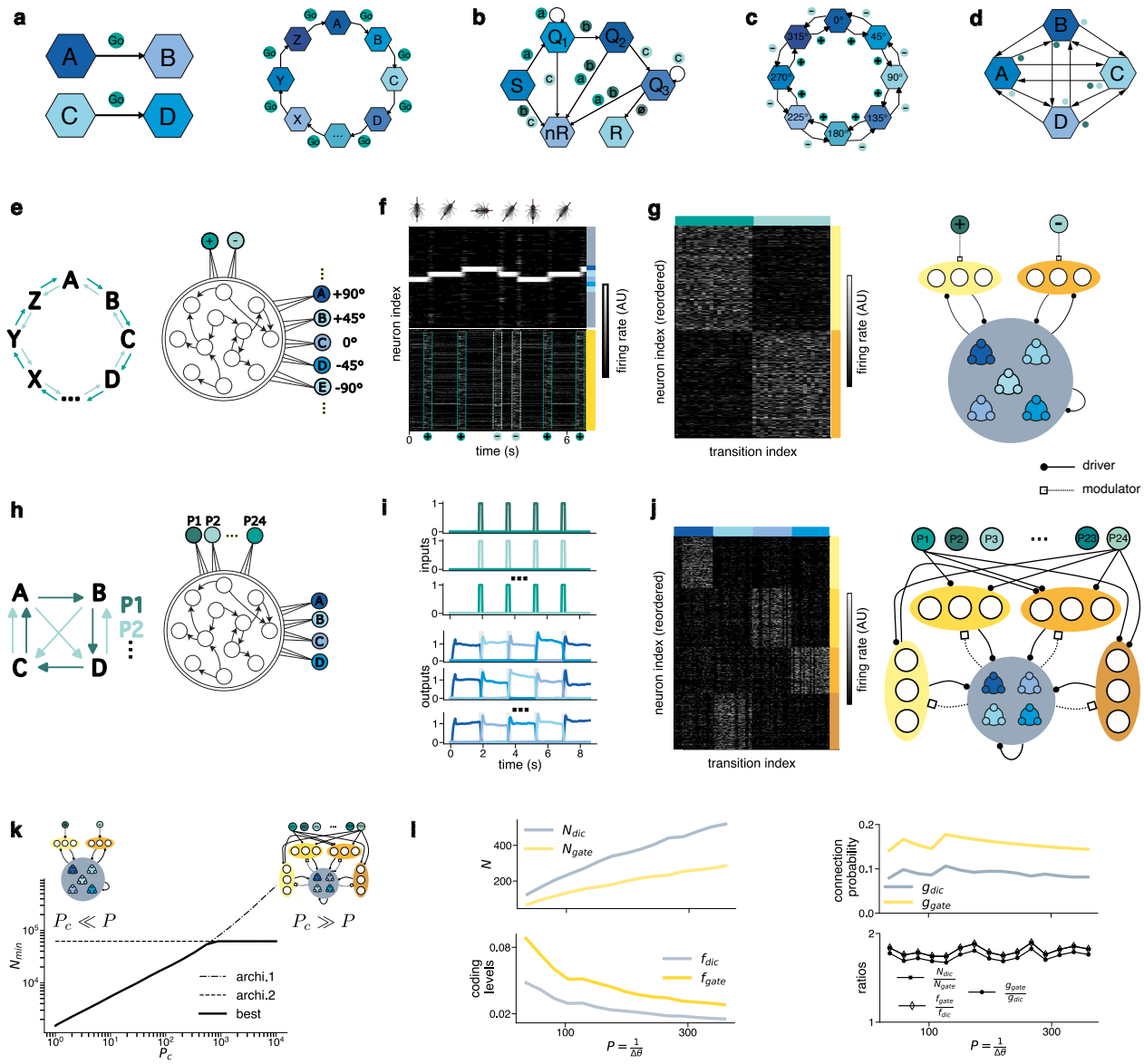
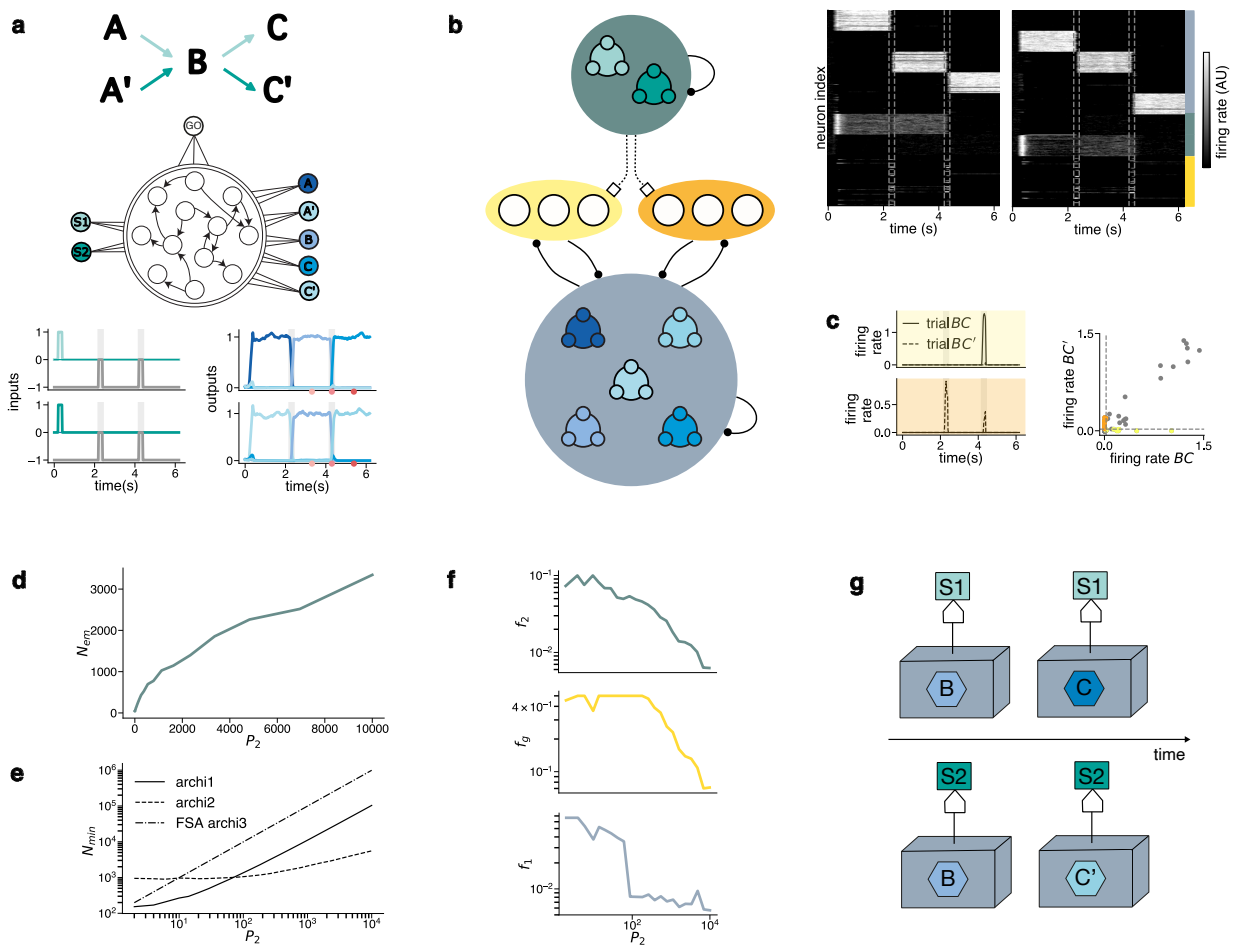


**Fig. 2:** ***NLSC implement Finite-State-Automata.*** ***a-d*** *Finite-state automata representations: blue hexagons represent machine states, arrows represent states transitions in response to external inputs, green dots.* ***a****, Transducers for solving the sensory-motor transformation task (left) and generating sequences of 20 states (right).* ***b****, FSA recognizing a language composed of all strings of words of the form $a^*bc^*$ ($*$ means that $a$ and $c$ can appear arbitrary numbers of times), the automaton ends up in $R$, or $nR$, when the presented string belongs, or not, to the language.* ***c****, Categorizer implementing a head-direction system.* ***d****, FSA describing a machine with 24 external inputs, each driving a permutation on 4 states.* ***e****, Modeling head-direction encoding as two inverse permutations among dictionary states, under the control of clockwise and counter-clockwise vestibular velocity inputs.* ***f****, Raster of neural activity in a trained NLSC with 20 dictionary states for a sequence of turns (green dots). Top: activation of cell-assemblies in dictionary neurons, bottom: activations of gate neurons.* ***g****, Raster of patterns of activation of the 820 gate neurons, for the 20 clockwise and 20 counter-clockwise transitions.* ***h****, Training networks to implement 24 permutations on a set of 4 dictionary states.* ***i****, Activation of readout neurons (bottom, blue) upon 4 applications of a single go cue (top, green) (3 examples).* ***j****, Patterns of activation of the 1640 gate neurons. Transitions are ordered on the X-axis to first display the 24 transitions starting from state A, then those starting from states B, C and D.* ***k****, Computational capacity of NLSC implementing FSA.* ***l****, Physiological and anatomical characteristics of NLSC optimized to implement head-direction systems with improving resolutions.*

# 2 Assembling NLSC for temporally structured behavior

**External memory for behaviors with long-distance dependencies**. In order to depart from the simple form of temporal structures associated with FSA, we considered long-distance dependencies which are commonly described in behavior. Examples are subject-verb agreement in sentences where the subject is separated from the verb by other words, or canari's courtship songs where the probability to produce a phrase depends not only on the currently sung phrase, but also on preceding phrases sung several steps in the past (5). Following what we previously did to model canari's song production (47), we trained ANN on a minimal task involving long-distance dependencies (Fig. 3a, Method 5.3.5). Networks are presented with one of two initializing cues and are asked to produce one of two sequences that share a common path of production states, and then diverge. Training a NLSC with a 5-states dictionary, one per symbol to be produced, did not allow to implement the task, but augmenting the architecture with extra neurons we obtained good performance. In such an architecture the NLSC is augmented with an external memory that maintains a contextual neural representation of the initializing cue (Fig. 3b), allowing to bias the transitions towards the correct dictionary state at the point where the two sequences diverge. This routing is accomplished by the gate neurons of the NLSC that activate upon conjunction of the go cue and one of the contextual neural representations (Fig. 3c, SI Fig. 4a,d). At the population level, the state of the external memory modulates the gain of two populations of gate neurons, controlling the interactions between cell-assemblies to correctly route neural activity (SI Fig. 1e-g).

Long-distance dependencies are present in various behavioral tasks used in neuroscience experiments, neural representations for dictionary (48; 49) or external memory (50; 51) states have been exhibited in different cortical networks (SI Discussion 6.10.3). It would be interesting to see whether gate neurons mediating communication between cortical areas could be described.




We computed $N_{min}$ for such architectures and analyzed their characteristics (SI Section 6.1.6). Sparse but distributed representations of $P_c$ external memory states are stored as stable states in an attractor neural network whose size $N_c$ scales as $N_c \propto \sqrt{P_c}$ (Fig. 3d). As for the case of FSA, using a NLSC with $P_c$ gate populations (Fig. 2g) was more efficient for $P_c \ll P$ (Fig. 3e, full line), compared to using a NLSC with $P$ populations (Fig. 3e, dashed line). The converse being true for $P_c \gg P$. Such an architecture storing $P_c = 10,000$ contextual representations, here thought of as words piloting a dictionary of phonemes of size $P = 30$, requires $N_{min} \simeq 6,000$ neurons. Neurons inside a cortical column could thus implement an articulatory apparatus mapping word representations into their motor production, with sparse neural patterns in external memory, dictionary and gate populations (Fig. 3f).

Training ANN with different constraints we found other network mechanisms that could support the production of sequences with long-distance dependencies (SI Section 5.3.5). A different solution is one where symbol B is produced by different neural states depending on sequence identity (SI Fig. 4k). Computing $N_{min}$ for this neural architecture allows to show it is not advantageous compared to the architecture with external memory we presented (Fig. 3e, dotted-dashed line, SI Section 6.1.6).

**Fig. 3:** ***NLSC augmented with an external memory.*** ***a****, ANN trained to produce a minimal set of two sequences S1 and S2 with long-distance dependencies.* ***b****, Left: Cartoon of NLSC with an external memory (green). Right: Raster of neural activity underlying the production of the two sequences.* ***c****, Left: Firing rate of two exemple gate neurons driving transitions from B to C (top) and from B to C′ (bottom). Right: Firing rate of all gate neurons in transition B to C (X-axis) and B to C′ (Y-axis).* ***d****, Number of neurons in the external memory for optimized architecture.* ***e****,* $N_{min}$ *for various neural architectures implementing a set of* $P_2$ *sequences on a dictionary of* $P = 30$ *states with* $\epsilon = 0.1$*.* ***f****, Coding levels in external memory (top), gates (middle) and dictionary (bottom) for the optimized architecture.* ***g****, Time evolution of a FSA (box) augmented with an external memory (square) during the production of S1 and S2. The front-side of the box displays the state in which the automaton is in.*

**NLSC as a controllable external memory for behaviors organized into chunks**. Behavior can often be described as hierarchically organized. For example, the different locomotive behaviors of zebrafish larvae, such as in hunting or counter-stream swimming, can be decomposed into temporal sequences of thirteen elementary tail movements, which are themselves a sequential combination of muscle activations (3). Here we assemble NLSC (Method 5.2.4) to build a neural architecture that implements such a motor hierarchy, producing sequences organized into chunks (24) (Fig. 4a, Method 5.3.6). To do so we built upon the architecture above (Fig. 3b) by adding an extra state in the dictionary, to be played at the end of each chunk (Fig. 4b, light blue cell-assembly). This state triggered transitions in the external memory through a new set of gate neurons (Fig. 4b, orange), which are bidirectionally coupled with the external memory and that receive connections from the low-level dictionary (Fig. 4b, blue). This leads them to exhibit the following non-linear mixed selectivity: neurons respond to conjunction of the middle-level dictionary states and the added stop state (SI Fig. 5a,b). Thus piling-up two NLSC allowed to produce sequences organized into chunks (Fig. 4c). This assemblage can be iterated to produce sequences with temporal structures with multiple hierarchical levels (SI Section 6.1.7).

We introduced a stop state in the dictionary of the lower level, it instructs the higher-level NLSC, which encodes chunk identity, to transit once the content of a chunk has been produced by the lower-level. By computing $N_{min}$ for architectures with and without stop states (SI Section 6.1.7), we found that using a stop state is beneficial: for a motor apparatus wired to produce $10,000$ words and $5$ sentences of $100$ words, the number of required neurons goes from $N_{min} = 5,500$ with a stop state to $N_{min} = 10,500$ without. This advantage becomes even more pronounced when the number of sentences to be produced increases. For 20 sentences, it goes from $N_{min} = 5,900$ with a stop state to $N_{min} = 26,300$ without (Fig. 4d).

Such interacting NLSC share similarities with the large-scale organization of motor cortex. Human motor cortex has been described as an intertwining of functionally low-level motor-effectors regions with functionally higher-level regions (52). Accordingly, electrical stimulations in monkeys' motor cortex showed that at some location, brief stimulations elicited activation of single motor effectors such as finger moves. By contrast, in nearby regions, brief stimulations were inoperant but more prolonged ones triggered the execution of whole motor programs such as moving upper limbs to direct the hand towards the mouth (53). We expect neural activity in the

high-level regions to evolve more slowly than in the low-level ones (54) (Fig. 4c).

The automaton corresponding to the two NLSC piled-up (Fig. 4e) has an extra feature compared to the one we proposed for the minimal task with long-distance dependencies (Fig. 3g): machine states can now update the state of, or write on, the external memory. Writing on the external memory corresponds to the low-level dictionary signaling the upper-level dictionary of its current state, and triggering transitions when the low-level state goes out of the null-space of the high-level NLSC (55), similar to corollary discharges or efference copies observed in motor systems (56).

**NLSC as an attentional focus for behaviors with algebraic patterns**. Another form of temporal structure that has been highlighted by cognitive neuroscience is algebraic patterns (24), characterized by the ability to recognize or produce patterned data, e.g. XYX, where X and Y are roles that can be filled from an alphabet of symbols (Fig. 4f, Method 5.3.7). In order to assemble an architecture suitable for such a production, we took inspiration from prefrontal cortex neural activity recordings during a task in which monkeys have to memorize and produce temporal sequences XYZ of three roles, with an alphabet of fillers constituted of eight spatial locations (18). The neural patterns associated to the memory of spatial locations for each role were carried by distinct groups of neurons (18; 57). In Fig. 4g,h we present a neural architecture producing temporal sequences with an algebraic pattern structure. Two dictionaries (green) form an external memory and are used to store a representation of the content of roles X and Y, they project to neurons responsible for the production of the symbols (blue). Interactions between the external memory and the production network are dynamically modulated by the state of a NLSC responsible for producing the abstract sequence XYX (purple). Modulation is performed through a new type of *output* gate neurons associated with dictionaries of the external memory (orange). As opposed to the *transition* gate neurons that we have discussed thus far, they do not feedback to their dictionaries.

Using the perceptron analysis we computed how the number of *output* gate neurons depends on task parameters (SI Section 6.1.8), it has a term roughly proportional to the number of roles (Fig. 4i) and it scales as $\sqrt{P}$, the number of symbol each role can be filled with (Fig. 4j). From our theory, we can also show that splitting neural representations for each role into segregated networks, as observed in prefrontal cortex (18), is more efficient than storing distributed representations of all role-filler pairs in a single attractor network with multi-item working memory (Fig. 4k, SI Section 6.1.8).

Our automaton interpretation for this neural architecture is the following: the NLSC encoding the abstract sequence is a FSA that implements a reading head which couples the relevant cell of the external memory with a machine printing the currently represented symbol (Fig. 4l). From this perspective PFC circuits of the monkeys implement an external memory accessed by e.g. lower motor centers. It would be interesting to see whether neurons implementing the head NLSC for the abstract sequence XYX, or *output* gate neurons with a mixed selectivity to roles and fillers, could be observed while animals perform such a task.

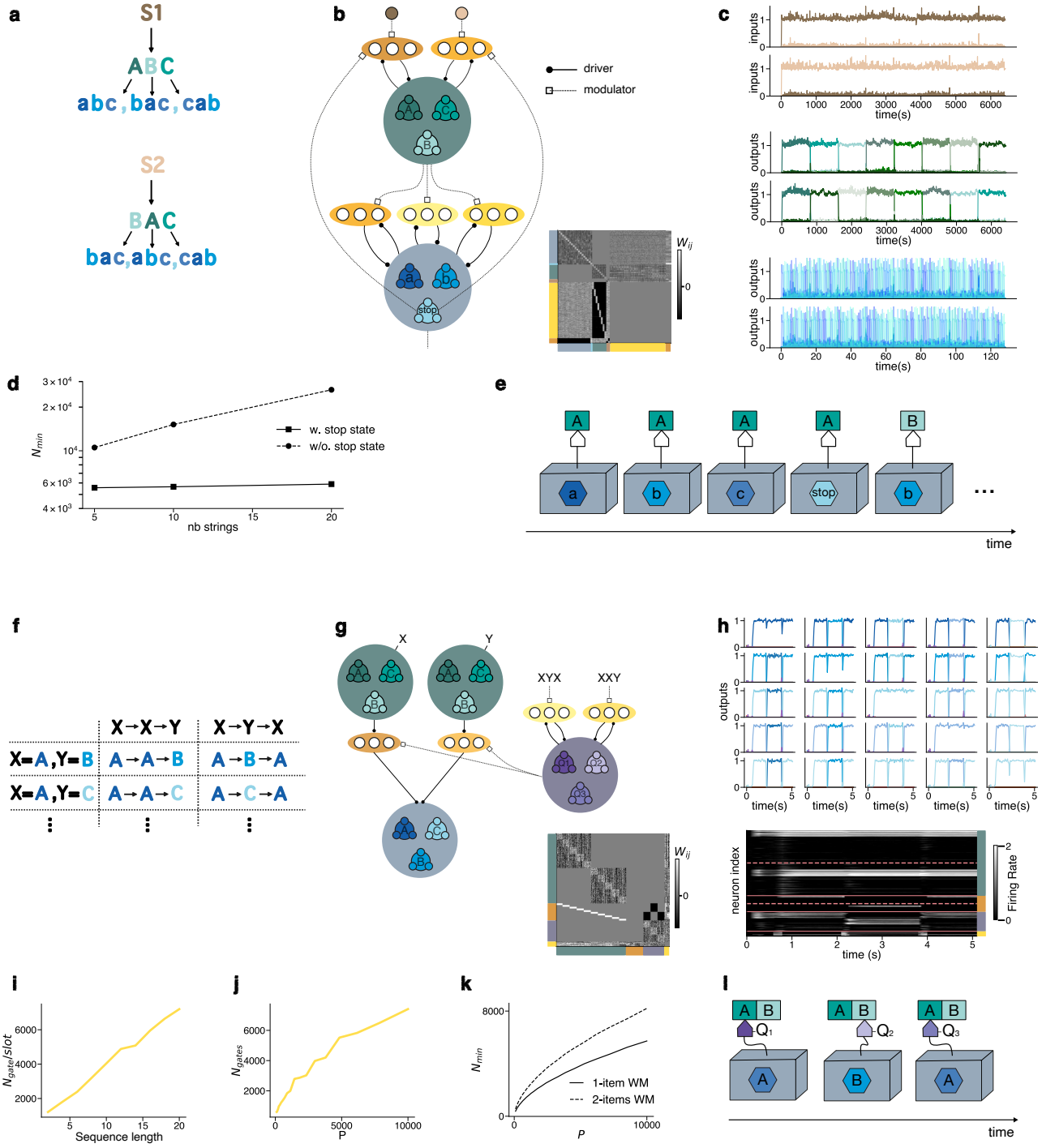

a
S1
ABC
abc, bac, cab
S2
BAC
bac, abc, cab
b
driver
modulator
stop
$W_{ij}$
c
inputs
outputs
time(s)
d
$N_{min}$
w. stop state
w/o. stop state
nb strings
e
A
B
a
b
c
stop
time
f
X→X→Y
X→Y→X
X=A,Y=B
A→A→B
A→B→A
X=A,Y=C
A→A→C
A→C→A
g
X
Y
XYX
XXY
$W_{ij}$
h
outputs
time(s)
neuron index
Firing Rate
time (s)
i
$N_{gate}/slot$
Sequence length
j
$N_{gates}$
P
k
$N_{min}$
1-item WM
2-items WM
P
l
A
B
$Q_1$
$Q_2$
$Q_3$
time

**Fig. 4:** ***Assembling NLSC for complex temporal structures.*** ***a****, Temporal structure of sequences organized into chunks.* ***b****, Piling-up two NLSC. Green cell-assemblies represent chunk identity, blue cell-assemblies represent terminal symbols to be produced. Gate neurons (yellow shades) program transitions between cell-assembly activations.* ***c****, Cell-assembly activations in the different levels of the architecture during the production of S1 (top) and S2 (bottom).* ***d****,* $N_{min}$ *for neural architectures with and without stop states in the low-level NLSC.* ***e****, Time evolution of an automaton summarizing the behavior of the proposed neural architecture. A FSA reads from the external memory to determine its next state. When in the stop state, the FSA writes a new symbol on the external memory, triggering a new set of transitions in the FSA.* ***f****, Temporal structure of sequences with algebraic patterns.* ***g****, Neural architecture for producing sequences with an algebraic patterns temporal structure. An output dictionary (blue), reads the state of one of two dictionaries playing the role of two external memory cells. The current dictionary state of the head NLSC (purple) decides from which cell the output is read, trough modulation of populations of output gate neurons (orange) associated with external memory cells.* ***h****, Top: readout activations (blue cell assemblies) for a network assembled to produce sequences XYX with 5 possible different fillers. Bottom: raster of neural activations in the purple and green NLSC during the production of sequence ABA (dashed square on Top).* ***i,j****, Number of output gate neurons for an optimized architecture as a function of the number of slots in the external memory (****i****) and the number of possible symbol in each memory slot (****j****).* ***k****, Minimal number of dictionary neurons for an external memory implemented by a single attractor network operating in a two-items working memory regime (dashed) or two attractor networks with single item working memory (full).* ***l****, Automaton interpretation of the neural architecture. A head switches locations on the external memory throughout sequence production.*

# 3 Assembling NLSC for behavioral programs

**Neural Turing machines**. Here we show how this ability to move on the external memory (Fig. 4l) can be combined with the ability to write on it (Fig. 4e). The external memory is now composed of multiple NLSC (Fig. 5a shows a full NLSC with both populations of transition and output gate neurons), each implementing a memory cell (Fig. 5b green). A NLSC plays the role of a FSA, or machine, reading and writing on the external memory (Fig. 5b blue). A NLSC plays the role of the head, deciding from which cell the machine is reading and writing (Fig. 5b purple). Thus, combining NLSC allows us to assemble Turing machines (Fig. 5c, SI Section 6.7). Turing machines are fundamental to computer science. They consist of simple automata, which manipulate symbols according to sets of rules. Yet, they are computationally universal, meaning that such simple machines can implement any definable procedure, algorithm, or program. For systems neuroscience, it implies that NLSC-based architectures can support any pre-defined behavioral program performed by an animal in an experiment.

**Assembling NLSC for a specific behavioral program**. To illustrate how to assemble NLSC to build such an architecture, with both moving and writing on an external memory, we consider a foraging task with a planning component. An agent is navigating on a map of 10 spatial locations linked through 3 different paths, a value, representing e.g. food quantity, is associated to each location, and the planing algorithm identifies the shortest path leading to the collection of food to satisfy a predefined hunger level (Fig. 5d). In order to design NLSC and assemble them (Method 5.2.4, 5.3.8, SI Section 6.4) we follow the associated pseudo-code SI Section 6.3, the final neural

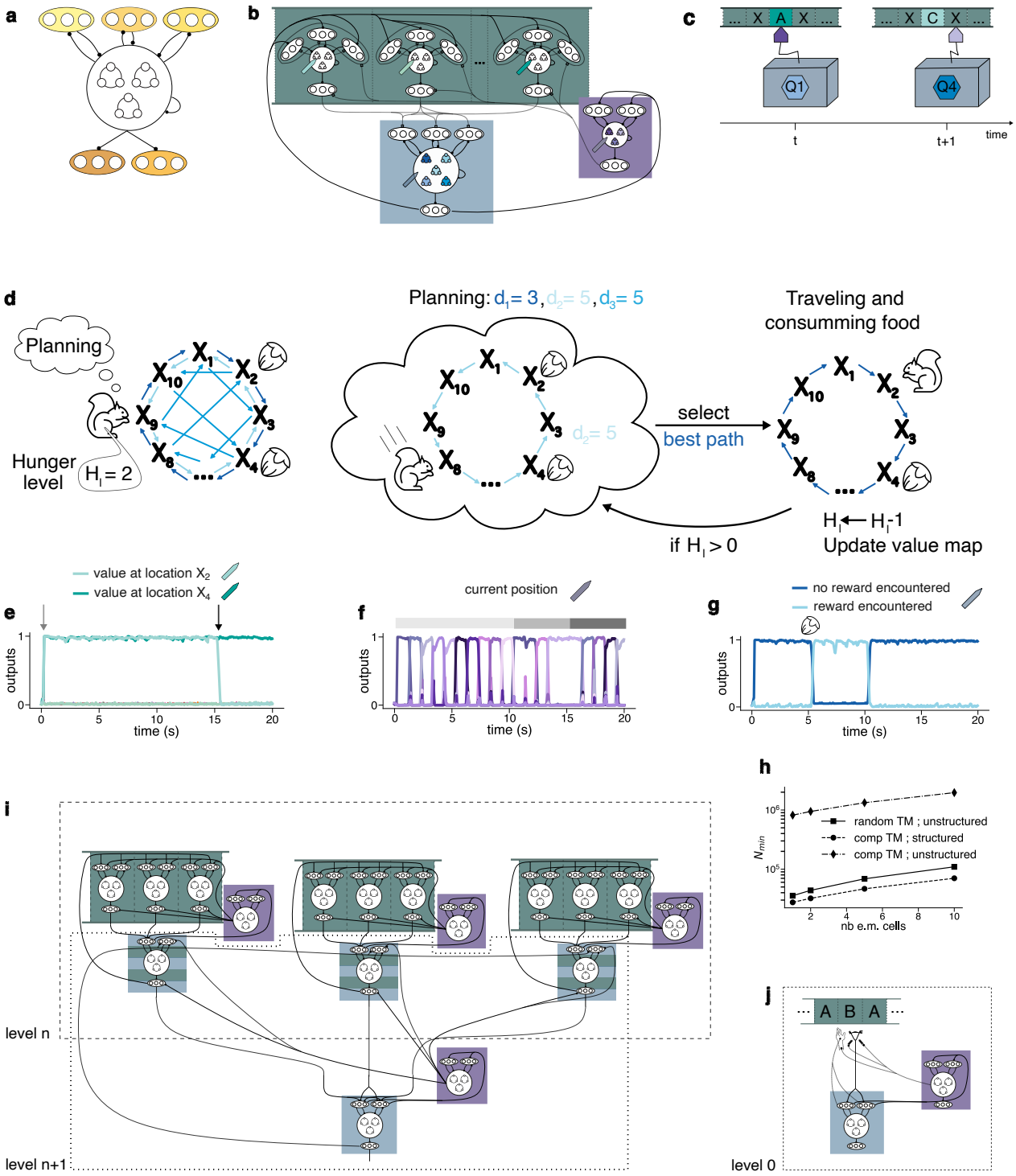

a
b
c
... X A X ...
... X C X ...
Q1
Q4
t
t+1
time
d
Planning
Hunger level
$H_l = 2$
Planning: $d_1 = 3$, $d_2 = 5$, $d_3 = 5$
$d_2 = 5$
select best path
Traveling and consumming food
if $H_l > 0$
$H_l \leftarrow H_l - 1$
Update value map
e
value at location $X_2$
value at location $X_4$
outputs
time (s)
f
current position
outputs
time (s)
g
no reward encountered
reward encountered
outputs
time (s)
h
random TM ; unstructured
comp TM ; structured
comp TM ; unstructured
$N_{min}$
nb e.m. cells
i
level n
level n+1
j
A B A
level 0

**Fig. 5:** ***Assembling NLSC for the implementation of behavioral programs.*** ***a****, Full NLSC with both populations of transition gate neurons (yellow) implementing multiple permutations over dictionary states, and populations of output gate neurons (orange) allowing to route information transmission to downstream circuits.* ***b****, Neural Turing machine,* ***c*** *mapped onto its conventional automata representation.* ***d****, Foraging task. Left: environment on which an agent evolves to satisfy a hunger level, 10 locations are linked by three paths (blue shaded arrows). Middle: planing step of the task whereby the agent mentally travels on the light-blue path 2 to estimate the distance from a location with a food item. Right: After selecting the path with shortest distance from food, the agent travels, consumes food, updates its hunger level as well as its mental representation of where food is available. If still hungry it goes back to planing a route towards a new food location.* ***e****, Dictionary activations for the 10 NLSC of the external memory circuits (green electrodes in (B), readouts that signal 0 value are not shown): at the time of initialization (light arrow) food items are present at two locations, and once a food item is consumed the external memory cell at this location is updated (dark arrow).* ***f****, Cell-assembly activations for the head circuit (purple electrode) while planing path 3 (light gray), traveling to consume food (middle-light gray) and start planing again (dark gray).* ***g****, Cell-assembly activations for the machine (blue electrode) updating its state based on the currently read cell content, the particular NLSC we show keeps a memory that food has been encountered in planing a path.* ***h****,* $N_{min}$ *for NLSC-based Turing machines implementing random transitions with non-compositional representations (full square), structured transitions with compositional representations (dashed circles), and random transitions with compositional representations (dashed diamonds).* ***i****, Recursively piling-up neural Turing machines to build hierarchies of sensory-motor programs.* ***j****, Initialization of the recursion, with a first NLSC sampling the state of a subpart of the environment, through input sensors, and acting on the this subpart of the environment through motor effectors.*

architectures is represented in SI Fig. 8e. We now describe cell-assembly activations throughout running the behavioral program, in the different circuits. The variables storing the food value associated to each spatial location are represented in the external memory using 10 NLSC with a 2-states dictionary (value 0 or 1). At the beginning of the task, the external memory is loaded with values, mimicking an agent recollecting memories of where food has been cached (Fig. 5e, light arrow, two locations with food). When, at some point during the behavioral program, food is consumed at a location X, the associated value is decreased by switching the dictionary state of the Xth NLSC of the external memory (Fig. 5e, dark arrow). In order to navigate on the 10 spatial locations, either during the planing components or while travelling towards food consumption, the neural architecture is equipped with a head NLSC with a 10-states dictionary (one per location) and three populations of gate neurons, each programing one of the path. Fig. 5f shows cell-assembly activations in the head circuit for planing path 3 (light gray), traveling to consume food (medium gray) and planing again on path 1 (dark gray). Each cell-assembly activation in this circuit enables communication between one of the external memory cell and the machine circuit (connections from purple to green circuits Fig. 5b, SI Fig. 8e). The machine circuit is composed of multiple NLSC that coordinate the running of the behavioral algorithm (Method 5.3.8) or that hold variables in memory to program transitions. Fig. 5g shows the activity in such a NLSC, it encodes whether food has been encountered while planing a path, allowing the rest of the machine circuit to evaluate distance from food on each path (Method 5.3.8, SI Section 6.4). The posterior parietal cortex (58) or dorsolateral prefrontal cortex (59) could implement the NLSC of the head circuit representing and updating the (mental) position of the animal, while the external memory representing values for each position could be implemented by NLSC in the orbitofrontal cortex (60).

**Capacity of neural Turing machines**. In practice, as in any machine, the cells of the external memory are in finite number. The number of neurons required to build a defined NLSC-based Turing machine can be computed (SI Section 6.1.9, SI Fig. 8d) and depends: for the external memory, on its number of cells and the number of symbols supported on each cell; for the machine, on the number of machine states ; for the head system, on the number of cells. The size and number of gate populations depend on the number of dictionary transitions that should be coordinated among external memory, machine and head NLSC. In Fig. 5h, we show $N_{min}$ for a random Turing machine with $P = 1000$ machine states and $P_{e.m.} = 100$ external memory symbols (SI Section 6.1.9). Extending the tape from 1 to 10 cells increases $N_{min}$ from $36,000$ to $110,000$ (full line). In this modeling, machine states are carried by a single NLSC with $P$ states $\{X^\mu\}_{\mu=1,\dots,P}$. This is not the case for the neural architecture we proposed for the foraging task, which has a machine circuit composed of multiple NLSC (SI Fig. 8e, blue), such that machine states are indexed not by P symbols but by tuplets of symbols $(path, hunger\,level, distance\,counter...)$: representations are compositional, or factored. The dashed line in Fig. 5h shows that using compositional representations $\{(X^\mu, Y^\nu)\}_{\mu,\nu=1,\dots,\sqrt{P}}$ supported by 2 NLSC allows to reduce $N_{min}$ from $110,000$ to $70,000$ for implementing a program with a structured set of transitions between dictionary states (SI Section 6.1.9). In general compositional representations are not beneficial for unstructured sets of transitions (Fig. 5h, dashed-point line) since the number of gate neurons increases faster than the number of dictionary neurons decreases.

**Sensory-motor programs hierarchy with NLSC**. In the previous section we have seen how NLSC

can be combined to implement programs. Neuroscience (28) and artificial intelligence (61) research lead to the hypothesis that cortical networks implement a hierarchy of sensory-motor programs, which allows central nervous systems to perceive and act upon their environment. We show here how NLSC can be combined to instantiate such a hierarchy. The construction relies on two ingredients. First, a recursion principle in which a NLSC is interpreted both as the cell of an external memory of a Turing machine at level $n$, and, the FSA of a Turing machine at level $n+1$ (Fig. 5i). Second, for the initialization of this recursion, the Turing machine-0 is composed of a NLSC to implement a FSA-0 and the environment is treated as the external memory (Fig. 5j). The dictionary states of the NLSC encodes the states of FSA-0. Gate neurons encode the conjunction of the current state of the FSA-0 and inputs coming from sensing the external world. In addition gate neurons control effectors that update the state of the currently visited cell of the environment. In SI Fig. 8f we show the connectivity structure of a network assembled to implement an agent equipped with a routine -*eating*- calling multiple sub-routines - *setting a table, serving from a dish, consuming food and clearing the table*- to satisfy a goal (Method 5.3.9).

Together, this highlight properties of neural circuits one could expect to find at the basis of a sensory-motor hierarchy. Cortical columns could be well suited to implement the NLSC composing the various stages of the hierarchy (see Discussion). A brain structure that would fit well with the *transition* gate neurons of level-0 (Fig. 5j) is the superior colliculus: it receives inputs from sensors, e.g. the retina, and drives effectors through the spinal cord, e.g. eye muscles (62). It is also bi-directionally coupled with e.g. the frontal eye field which could play the role of a head circuit encoding the currently visited location. Mixing frontal eye field inputs with the current visual content from retinal inputs would provide neural representations to trigger ocular saccades (63) to implement e.g. a recognition algorithm.

## 4 Oscillatory activity driving state update

In the architectures presented so far, transitions between machine states are triggered by an external go cue. We wondered how such transitions could be triggered internally. We trained a NLSC to autonomously switch between its dictionary states every one second (Fig. 6a). To do so, the network developed an oscillatory mode (Fig. 6b,c), with patterns of gate activations triggering transitions among dictionary states (SI Fig.6a,b). At the network level, dictionary states became meta-stables (64). While at the single cell level, this oscillatory mode manifested as dictionary neurons having ramping up or ramping down activations (Fig. 6d,e,f), as observed in singing birds (65) or in working memory tasks with delays with a characteristic time scale (66; 67; 68).

# Discussion

We analyzed artificial neural networks trained on a set of tasks with primitive temporal structures that have been proposed to underly behavior (24). This lead us to highlight the NLSC as an elementary neural circuit (Fig. 5a) that can be combined to assemble neural architectures suited for all these tasks.

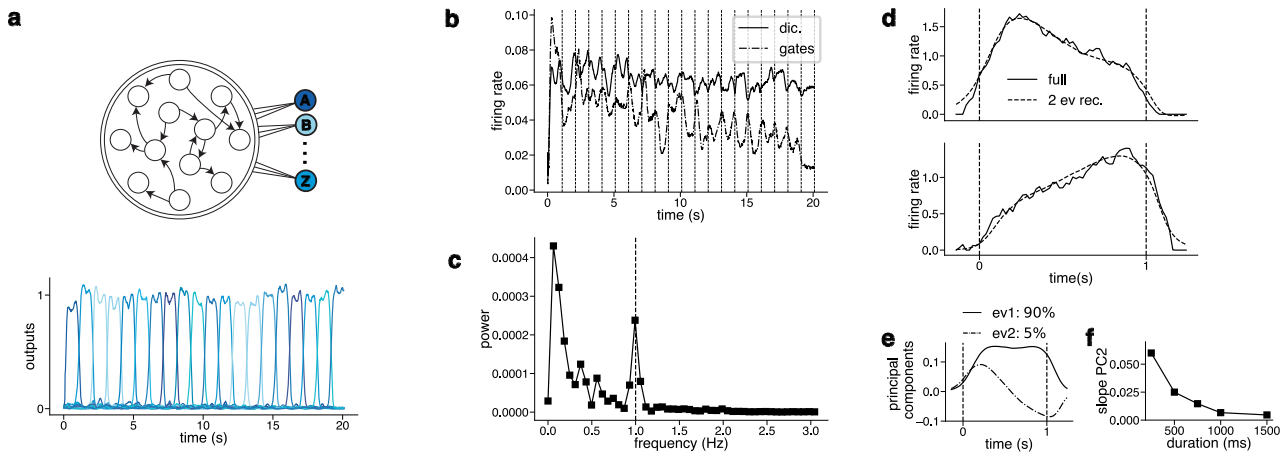



**Fig. 6:** ***Oscillatory activity driving state update.*** ***a****, ANN trained to autonomously switch states every one second.* ***b****, Average firing rate of dictionary and gate neurons.* ***c****, Power spectrum of the average activity of gate neurons.* ***d****, Firing rate of two example dictionary neurons.* ***e****, Temporal profiles of dictionary neurons are well explained by two components.* ***f****, Slope of the second dashed component as a function of state duration.*

**Relationship between neural and symbolic computations**. To characterize these neural architectures we leveraged neural network theory tools developed for the study of attractor neural networks (SI Discussion 6.10.1). Attractor neural networks (14) are described as dynamical systems in which trajectories of neural activity converge towards fixed-points, a feature that has been proposed to support various computations such as working-memory maintenance, stimulus denoising or temporal integration (13). In order to provide sensible definitions for temporal structures and for the computational power of our neural architectures, we resorted to computer science and interpreted attractor states as automata states. In this framework, NLSC can act as the machine, the external memory, or the head of a Turing machine (Fig. 5b,c), showing it can be combined to implement any behavioral program (SI Discussion 6.10.2). From this point of view, neural computations are instantiated by the coordinated transitions between attractor-states of multiple NLSC. This establishes a bridge between neural and symbolic computations, whereby neural network operations are conceptualized as symbol manipulations. We computed how the minimal number of neurons of an architecture depends on its computational power (e.g. number of external memory cells, machine states, head moves). This has allowed us to compare various architectures and to show that optimized NLSC-based architectures, which use distributed and sparse neural patterns (43), are efficient, for instance when comparing our finite-state-automaton implementation with the classic one proposed by McCulloch and Pitts (30) (SI Discussion 6.10.2).

**Relationship with brain circuits**. The concept of computation has proven valuable for understanding the function of various biological entities (69) and may as well offer insights into the workings of brain circuits. The recurrent circuitry of brain networks has been shown to be well suited to implement attractor dynamics, with stable neural states mapped onto activations of cell-assemblies as introduced by Hebb (SI Discussion 6.10.3). By contrast experimental evidence is more scarce for the gate neurons of the NLSC. The most direct evidence comes from studies

of the head-direction system in flies, with the identification of two populations of protocerebral bridge neurons mixing clockwise or counter-clockwise vestibular velocity and current angular position (26). Converging evidence comes from the study of C. Elegans brains and the discovery of neurons involved in coordinating various levels of the motor hierarchy to shape the temporal structure of locomotion (4). Whether and how mammalian brains combine NLSC to implement behaviors with more complex temporal structures remains to be explored. We mentioned a first possibility in SI Discussion 6.10.3, where we explain how the minimalistic set of ingredients required to build NLSC are related to experimental observations about thalamic circuits. Another possibility is that cortical columns (29) can implement NLSC, with e.g. L2/3 neurons implementing transition gate neurons and L5/6 neurons output gate neurons (Fig. 5a yellow and orange, SI Discussion 6.10.3).

Throughout the manuscript we have shown how interactions within large-scale brain circuits can be understood as interactions between NLSC: interactions between subparts of motor-cortex (52) to implement procedures (Fig. 4b), interactions between PFC and down-stream cortical areas for the implementation of rule-based sensory-motor programs (48; 50) (Fig. 3b, Fig. 4g), interactions within the cortical hierarchy (27; 28) (Fig. 5b), or interactions between a brain and its environment (28; 70) (Fig. 5i,j). Further work is required to reach more precise descriptions of these systems and see how well they map onto interacting NLSC, and e.g. see whether dictionary and gate neurons can be assigned to specific cell classes or not.

The quantitative theory we developed allows to characterize the anatomical and physiological properties of efficient neural architectures. For a given behavior, a confrontation with experimental data can be devised. This should aim to characterize the selectivity of recorded neurons to task variables (39), describe whether neurons are driven or modulated by their various inputs (42; 71), characterize the coding level of cells and their cell class dependence (72), but also the trajectories of population recordings during task execution (73), the connectivity statistics between cell classes (74; 75) or the population structure of circuits (76; 77). The examples of behaviors treated here are inspired by animal models currently used in systems neuroscience experiments. Of particular interest are context-dependent tasks (48; 50; 78), which are known to engage interactions between distinct cortical areas and thus seem well suited to confront cortical networks with our theory. As for the fly's head-direction circuit, understanding whether and how mammalian brain circuits implement path integration on continuous attractors could also be interesting, although the current theory might have to be extended to give adequate descriptions (SI Discussion 6.10.1).

**Functional role for neural oscillations**. In most of the neural architectures we discussed, an external go cue acted as a clock signal to trigger transitions among neural states. The last set of experiments we presented suggests that network oscillations (12) could serve as internally generated clocks, leading attractor-states to become metastable (22). This is less of a strong role for oscillations than in binding by synchrony (79; 80), where oscillatory modes are proposed to dynamically establish communication channels between neural circuits. In NLSC architectures, communication channels are pre-wired, with contextual representations in higher-level areas deciding which communication channel to activate, while oscillations pace dictionary state updates. Further studies could assess whether brain oscillations observed at various frequencies (12) reflect updating of neural network states, similar to what has been observed for gamma frequencies (81) (SI Dis-

cussion 6.10.3). The network architectures we presented in this work are highly structured, with a special circuitry organizing transitions between network states, this can also be contrasted with a recent proposal whereby recurrent networks operating at a critical point can be reconfigured with inputs to perform different tasks (82).

**Temporal structures of behavior reflecting computational abilities of neural circuits**. In trying to find a neural substrate underlying high-level cognitive processes, we proposed a set of behavioral temporal structures which reveals computational features of the underlying neural circuitry (Fig. 2c,e,g, Fig. 3a,b,g, Fig. 4a,b,e, Fig. 4f,g,l or Fig. 5b,c). To do so, we took inspiration from (24) which proposed chunking and algebraic patterns as two important forms of temporal structures to model behavior. Interestingly, we found that behaviors associated with these temporal structures rely on the computational ability of neural circuits to write symbols (Fig. 4e) and to move on cells of an external memory (Fig. 4l). Another temporal structure they proposed involves the generation of nested-tree structures, accounting for the combinatorial expressivity of behaviors such as natural language. Here, guided by computational principles, we ended-up putting the focus on the notion of sensory-motor programs instead, which is also associated with a form of combinatorial expressivity (SI Section 6.9). Previous studies have shown that imposing physical constraints on the underlying neural substrate can lead to quantitative predictions of behavioral measurements (83; 84; 85). Further work is required to investigate whether the computational account exposed here could be tested in similar ways, for example by predicting the behavioral circumstances under which compositional mental representations can be expected (86) (Fig. 5h), or by examining features of natural languages (SI Section 6.9).

**Perspectives**. From Marr's computational perspective (87), by showing how to combine NLSC, we propose a piece of hardware that can be used to implement any behavioral program. We can reiterate previous speculations that mammalian cortex, with its columnar architecture (29), could be seen as a set of machines implementing sensory-motor programs (28; 61). Brain activity operating at different frequencies (12) would reflect imbricated machine operations at different time scales, all being relevant for understanding cognition (88). Our examples show how this can be relevant to implement sensory processing, motor programs, or planning algorithms running mental models of the world. While we did not focus on any particular cognitive process defining the first level of Marr's hierarchy, it would be interesting to use the quantitative approach exposed here to focus on specific cognitive abilities such as natural language, or the ability to predict external sensory cues. This could help better delineate the function of brain circuits (89; 90). Towards this goal, it would also be useful to go beyond the notion of sensory-motor program and analyze other cognitive abilities such as learning.

# References


[1] K. S. Lashley, The Problem of Serial Order in Behavior (1951).

[2] D. O. Hebb, *The organization of behavior: A neuropsychological theory* (Psychology press) (2005).

[3] J. C. Marques, S. Lackner, R. Félix, M. B. Orger, Structure of the Zebrafish Locomotor Repertoire Revealed with Unsupervised Behavioral Clustering. *Current Biology* **28** (2), 181–195.e5 (2018), doi:10.1016/j.cub.2017.12.002.

[4] J. Huo, *et al.*, Hierarchical behavior control by a single class of interneurons. *Proceedings of the National Academy of Sciences* **121** (47), e2410789121 (2024).

[5] J. E. Markowitz, E. Ivie, L. Kligler, T. J. Gardner, Long-range Order in Canary Song. *PLoS Computational Biology* **9** (5), e1003052 (2013), doi:10.1371/journal.pcbi.1003052.

[6] J. E. Markowitz, *et al.*, Spontaneous behaviour is structured by reinforcement without explicit reward. *Nature* (2023), doi:10.1038/s41586-022-05611-2.

[7] N. Chomsky, Syntactic structures. *The hague: Mouton* (1957).

[8] D. Jurafsky, J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*, 3rd ed. (2025), `https://web.stanford.edu/ jurafsky/slp3/`, online manuscript released January 12, 2025.

[9] X. Jiang, *et al.*, Production of Supra-regular Spatial Sequences by Macaque Monkeys. *Current Biology* **28** (12), 1851–1859.e4 (2018), doi:10.1016/j.cub.2018.04.047.

[10] R. L. Lewis, S. Vasishth, J. A. Van Dyke, Computational principles of working memory in sentence comprehension. *Trends in Cognitive Sciences* **10** (10), 447–454 (2006), doi: 10.1016/j.tics.2006.08.007.

[11] A. Alamia, V. Gauducheau, D. Paisios, R. VanRullen, Comparing feedforward and recurrent neural network architectures with human behavior in artificial grammar learning. *Scientific Reports* **10** (1), 22172 (2020), doi:10.1038/s41598-020-79127-y.

[12] G. Buzsaki, A. Draguhn, Neuronal oscillations in cortical networks. *science* **304** (5679), 1926–1929 (2004).

[13] M. Khona, I. R. Fiete, Attractor and integrator networks in the brain. *Nature Reviews Neuroscience* (2022), doi:10.1038/s41583-022-00642-0.

[14] J. J. Hopfield, Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proceedings of the National Academy of Sciences* **79** (8), 2554–2558 (1982), doi:10.1073/pnas.79.8.2554.

[15] Y. Miyashita, Neuronal correlate of visual associative long-term memory in the primate temporal cortex. *Nature* **335** (6193), 817–820 (1988).

[16] J. OḰeefe, Place units in the hippocampus of the freely moving rat. *Experimental Neurology* **51** (1), 78–109 (1976), doi:https://doi.org/10.1016/0014-4886(76)90055-8.

[17] X.-J. Wang, Decision Making in Recurrent Neuronal Circuits. *Neuron* **60** (2), 215–234 (2008), doi:10.1016/j.neuron.2008.09.034.

[18] Y. Xie, *et al.*, Geometry of sequence working memory in macaque prefrontal cortex. *Science* **375** (6581), 632–639 (2022), doi:10.1126/science.abm0204.

[19] D. Xu, *et al.*, Cortical processing of flexible and context-dependent sensorimotor sequences. *Nature* (2022), doi:10.1038/s41586-022-04478-7.

[20] K. M. Cury, R. Axel, Flexible neural control of transition points within the egg-laying behavioral sequence in Drosophila. *Nature Neuroscience* **26** (6), 1054–1067 (2023), doi: 10.1038/s41593-023-01332-5.

[21] S. Recanatesi, U. Pereira-Obilinovic, M. Murakami, Z. Mainen, L. Mazzucato, Metastable attractors explain the variable timing of stable behavioral action sequences. *Neuron* **110** (1), 139–153.e9 (2022), doi:10.1016/j.neuron.2021.10.011.

[22] L. Mazzucato, Neural mechanisms underlying the temporal organization of naturalistic animal behavior. *eLife* **11**, e76577 (2022), doi:10.7554/eLife.76577.

[23] O. Barak, Recurrent neural networks as versatile tools of neuroscience research. *Current Opinion in Neurobiology* **46**, 1–6 (2017), doi:10.1016/j.conb.2017.06.003.

[24] S. Dehaene, F. Meyniel, C. Wacongne, L. Wang, C. Pallier, The Neural Representation of Sequences: From Transition Probabilities to Algebraic Patterns and Linguistic Trees. *Neuron* **88** (1), 2–19 (2015), doi:10.1016/j.neuron.2015.09.019.

[25] J. D. Seelig, V. Jayaraman, Neural dynamics for landmark orientation and angular path integration. *Nature* **521** (7551), 186–191 (2015), doi:10.1038/nature14446.

[26] D. Turner-Evans, *et al.*, Angular velocity integration in a fly heading circuit. *eLife* **6**, e23496 (2017), doi:10.7554/eLife.23496.

[27] D. J. Felleman, D. C. Van Essen, Distributed hierarchical processing in the primate cerebral cortex. *Cerebral cortex (New York, NY: 1991)* **1** (1), 1–47 (1991).

[28] J. Fuster, *The prefrontal cortex* (Academic press) (2015).

[29] V. B. Mountcastle, The columnar organization of the neocortex. *Brain: a journal of neurology* **120** (4), 701–722 (1997).

[30] M. L. Minsky, *Computation: finite and infinite machines* (Prentice-Hall Englewood Cliffs) (1967).

[31] S. Vyas, M. D. Golub, D. Sussillo, K. V. Shenoy, Computation Through Neural Population Dynamics. *Annual Review of Neuroscience* **43** (1), 249?275 (2020), doi:10.1146/annurev-neuro-092619-094115.

[32] E. Russo, V. M. K. Namboodiri, A. Treves, E. Kropff, Free association transitions in models of cortical latching dynamics. *New Journal of Physics* **10** (1), 015008 (2008), doi:10.1088/1367-2630/10/1/015008.

[33] E. Russo, A. Treves, Cortical free-association dynamics: Distinct phases of a latching network. *Physical Review E?Statistical, Nonlinear, and Soft Matter Physics* **85** (5), 051920 (2012).

[34] M. Rigotti, Internal representation of task rules by recurrent dynamics: the importance of the diversity of neural responses. *Frontiers in Computational Neuroscience* **4** (2010), doi: 10.3389/fncom.2010.00024.

[35] L. Logiaco, L. Abbott, S. Escola, Thalamic control of cortical dynamics in a model of flexible motor sequencing. *Cell Reports* **35** (9), 109090 (2021), doi:10.1016/j.celrep.2021.109090.

[36] M. Cotteret, H. Greatorex, M. Ziegler, E. Chicca, Vector Symbolic Finite State Machines in Attractor Neural Networks. *Neural Computation* **36** (4), 549–595 (2024).

[37] M. Dabagia, C. Papadimitriou, S. Vempala, Computation with Sequences of Assemblies in a Model of the Brain, in *International Conference on Algorithmic Learning Theory* (PMLR) (2024), pp. 499–504.

[38] H. K. Inagaki, *et al.*, A midbrain-thalamus-cortex circuit reorganizes cortical dynamics to initiate movement. *Cell* p. S0092867422001465 (2022), doi:10.1016/j.cell.2022.02.006.

[39] M. Rigotti, *et al.*, The importance of mixed selectivity in complex cognitive tasks. *Nature* **497** (7451), 585–590 (2013), doi:10.1038/nature12160.

[40] A. Dubreuil, A. Valente, M. Beiran, F. Mastrogiuseppe, S. Ostojic, The role of population structure in computations through neural dynamics. *Nature Neuroscience* **25** (6), 783–794 (2022), doi:10.1038/s41593-022-01088-4.

[41] L. Herron, P. Sartori, B. Xue, Robust Retrieval of Dynamic Sequences through Interaction Modulation. *PRX Life* **1** (2), 023012 (2023), doi:10.1103/PRXLife.1.023012.

[42] S. M. Sherman, R. W. Guillery, On the actions that one nerve cell can have on another: Distinguishing drivers from modulators. *Proceedings of the National Academy of Sciences* **95** (12), 7121–7126 (1998), doi:10.1073/pnas.95.12.7121.

[43] M. Tsodyks, M. FeigelḾan, The enhanced storage capacity in neural networks with low activity level. *EPL (Europhysics Letters)* **6**, 101 (1988).

[44] D. Amit, H. Gutfreund, H. Sompolinksy, Storing Infinite Numbers of Patterns in a Spin-Glass Model of Neural Networks. *Physical Review Letters* **55** (14), 1530–1533 (1985).

[45] J. S. Taube, R. U. Muller, J. B. Ranck, Head-direction cells recorded from the postsubiculum in freely moving rats. I. Description and quantitative analysis. *Journal of Neuroscience* **10** (2), 420–435 (1990).

[46] L. Petrucco, *et al.*, Neural dynamics and architecture of the heading direction circuit in zebrafish. *Nature Neuroscience* **26** (5), 765–773 (2023), doi:10.1038/s41593-023-01308-5.

[47] A. Dubreuil, A. Leblois, Network Mechanism Supporting Long-Distance-Dependencies, in *2021 International Joint Conference on Neural Networks (IJCNN)* (IEEE, Shenzhen, China) (2021), pp. 1–6, doi:10.1109/IJCNN52387.2021.9534151, `https://ieeexplore.ieee.org/document/9534151/`.

[48] V. Mante, D. Sussillo, K. V. Shenoy, W. T. Newsome, Context-dependent computation by recurrent dynamics in prefrontal cortex. *Nature* **503** (7474), 78–84 (2013), doi:10.1038/nature12742.

[49] N. Steinemann, *et al.*, Direct observation of the neural computations underlying a single decision. *eLife* **12**, RP90859 (2024), doi:10.7554/eLife.90859.

[50] L. I. Schmitt, *et al.*, Thalamic amplification of cortical connectivity sustains attentional control. *Nature* **545** (7653), 219–223 (2017), doi:10.1038/nature22073.

[51] Y. Cohen, *et al.*, Hidden neural states underlie canary song syntax. *Nature* **582** (7813), 539–544 (2020), doi:10.1038/s41586-020-2397-3.

[52] E. M. Gordon, *et al.*, A somato-cognitive action network alternates with effector regions in motor cortex. *Nature* (2023), doi:10.1038/s41586-023-05964-2.

[53] M. S. Graziano, Ethological Action Maps: A Paradigm Shift for the Motor Cortex. *Trends in Cognitive Sciences* **20** (2), 121–132 (2016), doi:10.1016/j.tics.2015.10.008.

[54] J. D. Murray, *et al.*, A hierarchy of intrinsic timescales across primate cortex. *Nature Neuroscience* **17** (12), 1661–1663 (2014), doi:10.1038/nn.3862.

[55] J. D. Semedo, A. Zandvakili, C. K. Machens, B. M. Yu, A. Kohn, Cortical Areas Interact through a Communication Subspace. *Neuron* **102** (1), 249–259.e4 (2019), doi:10.1016/j.neuron.2019.01.026.

[56] T. B. Crapse, M. A. Sommer, Corollary discharge across the animal kingdom. *Nature Reviews Neuroscience* **9** (8), 587–600 (2008).

[57] Z. Tian, *et al.*, Mental programming of spatial sequences in working memory in the macaque frontal cortex. *Science* **385** (6716), eadp6091 (2024).

[58] J. W. Gnadt, R. A. Andersen, Memory related motor planning activity in posterior parietal cortex of macaque. *Experimental brain research* **70**, 216–220 (1988).

[59] J. M. Fuster, Unit activity in prefrontal cortex during delayed-response performance: neuronal correlates of transient memory. *Journal of neurophysiology* **36** (1), 61–78 (1973).

[60] E. B. Knudsen, J. D. Wallis, Taking stock of value in the orbitofrontal cortex. *Nature Reviews Neuroscience* **23** (7), 428–438 (2022), doi:10.1038/s41583-022-00589-2.

[61] M. Minsky, *Society of mind* (Simon and Schuster) (1988).

[62] N. L. Benavidez, *et al.*, Organization of the inputs and outputs of the mouse superior colliculus. *Nature Communications* **12** (1), 4004 (2021), doi:10.1038/s41467-021-24241-2.

[63] S. Martinez-Conde, J. Otero-Millan, S. L. Macknik, The impact of microsaccades on vision: towards a unified theory of saccadic function. *Nature Reviews Neuroscience* **14** (2), 83–96 (2013), doi:10.1038/nrn3405.

[64] G. La Camera, A. Fontanini, L. Mazzucato, Cortical computations via metastable activity. *Current Opinion in Neurobiology* **58**, 37–45 (2019), doi:10.1016/j.conb.2019.06.007.

[65] H. Fujimoto, T. Hasegawa, D. Watanabe, Neural Coding of Syntactic Structure in Learned Vocalizations in the Songbird. *Journal of Neuroscience* **31** (27), 10023–10033 (2011), doi: 10.1523/JNEUROSCI.1606-11.2011.

[66] H. K. Inagaki, *et al.*, Neural Algorithms and Circuits for Motor Planning. *Annual Review of Neuroscience* **45** (1), annurev–neuro–092021–121730 (2022), doi:10.1146/annurev-neuro-092021-121730.

[67] J. W. Machens, C. K., Brody, Flexible Control of Mutual Inhibition: A Neural Model of Two-Interval Discrimination. *Science* **307** (5712), 1118–1121 (2005), doi:10.1126/science.1105783.

[68] M. Pagan, *et al.*, Individual variability of neural computations underlying flexible decisions. *Nature* (2024), doi:10.1038/s41586-024-08433-6.

[69] H. M. Al-Hashimi, Turing, von Neumann, and the computational architecture of biological machines. *Proceedings of the National Academy of Sciences* **120** (25), e2220022120 (2023), doi: 10.1073/pnas.2220022120.

[70] C. E. Schroeder, D. A. Wilson, T. Radman, H. Scharfman, P. Lakatos, Dynamics of Active Sensing and perceptual selection. *Current Opinion in Neurobiology* **20** (2), 172–176 (2010), doi: 10.1016/j.conb.2010.02.010.

[71] Zhang, Long-range and local circuits for top-down modulation of visual cortex processing. *Science* **345** (6197), 657–660 (2014), doi:10.1126/science.1251853.

[72] S. Sakata, K. D. Harris, Laminar structure of spontaneous and sensory-evoked population activity in auditory cortex. *Neuron* **64** (3), 404–418 (2009).

[73] J. P. Cunningham, B. M. Yu, Dimensionality reduction for large-scale neural recordings. *Nature Neuroscience* **17** (11), 1500–1509 (2014), doi:10.1038/nn.3776.

[74] X. Jiang, *et al.*, Principles of connectivity among morphologically defined cell types in adult neocortex. *Science* **350** (6264), aac9462 (2015).

[75] T. M. Consortium, *et al.*, Functional connectomics spanning multiple areas of mouse visual cortex. *Nature* **640** (8058), 435?447 (2025), doi:10.1038/s41586-025-08790-w.

[76] W. Yang, S. L. Tipparaju, G. Chen, N. Li, Thalamus-driven functional populations in frontal cortex support decision-making. *Nature Neuroscience* **25** (10), 1339–1352 (2022), doi: 10.1038/s41593-022-01171-w.

[77] J. Hirokawa, A. Vaughan, P. Masset, T. Ott, A. Kepecs, Frontal cortex neuron types categorically encode single decision variables. *Nature* (2019), doi:10.1038/s41586-019-1816-9.

[78] M. Pagan, *et al.*, Individual variability of neural computations underlying flexible decisions. *Nature* **639** (8054), 421?429 (2025), doi:10.1038/s41586-024-08433-6.

[79] W. Singer, Recurrent dynamics in the cerebral cortex: Integration of sensory evidence with stored knowledge. *Proceedings of the National Academy of Sciences* **118** (33), e2101043118 (2021), doi:10.1073/pnas.2101043118.

[80] A. Palmigiano, T. Geisel, F. Wolf, D. Battaglia, Flexible information routing by transient synchrony. *Nature Neuroscience* **20** (7), 1014–1022 (2017), doi:10.1038/nn.4569.

[81] Y.-C. Huang, *et al.*, Dynamic assemblies of parvalbumin interneurons in brain oscillations. *Neuron* **112** (15), 2600–2613 (2024).

[82] M. O. Magnasco, Input-driven circuit reconfiguration in critical recurrent neural networks. *Proceedings of the National Academy of Sciences* **122** (10), e2418818122 (2025), doi: 10.1073/pnas.2418818122.

[83] S. Romani, I. Pinkoviezky, A. Rubin, M. Tsodyks, Scaling Laws of Associative Memory Retrieval. *Neural Computation* **25** (10), 2523–2544 (2013).

[84] W. Zhong, *et al.*, Random Tree Model of Meaningful Memory. *Phys. Rev. Lett.* **134**, 237402 (2025), doi:10.1103/g1cz-wk1l.

[85] L. Lebovich, R. Darshan, Y. Lavi, D. Hansel, Y. Loewenstein, Idiosyncratic choice bias naturally emerges from intrinsic stochasticity in neuronal dynamics. *Nature human behaviour* **3** (11), 1190–1202 (2019).

[86] M. Baroni, Linguistic generalization and compositionality in modern artificial neural networks. *Philosophical Transactions of the Royal Society B: Biological Sciences* **375** (1791), 20190307 (2020), doi:10.1098/rstb.2019.0307.

[87] D. Marr, T. Poggio, From understanding computation to understanding neural circuitry. *A.I. Memo (Artificial Intelligence Laboratory)* **357**.

[88] D. F. Parks, *et al.*, A nonoscillatory, millisecond-scale embedding of brain state provides insight into behavior. *Nature Neuroscience* **27** (9), 1829–1843 (2024), doi:10.1038/s41593-024-01715-2.

[89] K. Kermani Nejad, P. Anastasiades, L. Hertäg, R. P. Costa, Self-supervised predictive learning accounts for cortical layer-specificity. *BioRxiv* (2024), doi:10.1101/2024.04.24.590916.

[90] R. P. Rao, A sensory–motor theory of the neocortex. *Nature neuroscience* **27** (7), 1221–1235 (2024).

[91] D. J. Willshaw, O. P. Buneman, H. C. Longuet-Higgins, Non-Holographic Associative Memory. *Nature* **222**, 960–962 (1969).

[92] J. Nadal, Associative memory: on the (puzzling) sparse coding limit. *Journal of Physics A: Mathematical and General* **24**, 1093 (1991).

[93] A. Knoblauch, G. Palm, F. T. Sommer, Memory capacities for synaptic and structural plasticity. *Neural Computation* **22** (2), 289–341 (2010).

[94] A. M. Dubreuil, Y. Amit, N. Brunel, Memory Capacity of Networks with Stochastic Binary Synapses. *PLoS computational biology* **10** (8), e1003727 (2014).

[95] D. J. Amit, S. Fusi, Learning in neural networks with material synapses. *Neural Computation* **6** (5), 957–982 (1994).

[96] A. Knoblauch, F. T. Sommer, Structural Plasticity, Effectual Connectivity, and Memory in Cortex. *Frontiers in Neuroanatomy* **10** (2016), doi:10.3389/fnana.2016.00063.

[97] H. Sompolinsky, I. Kanter, Temporal association in asymmetric neural networks. *Physical review letters* **57** (22), 2861 (1986).

[98] F. Mastrogiuseppe, S. Ostojic, Linking Connectivity, Dynamics, and Computations in Low-Rank Recurrent Neural Networks. *Neuron* (2018), doi:10.1016/j.neuron.2018.07.003.

[99] N. Parga, M. Virasoro, The ultrametric organization of memories in a neural network. *Journal de Physique* **47** (11), 1857–1864 (1986), doi:10.1051/jphys:0198600470110185700.

[100] D. Krotov, J. J. Hopfield, Dense associative memory for pattern recognition. *Advances in neural information processing systems* **29** (2016).

[101] R. Chaudhuri, I. Fiete, Bipartite expander Hopfield networks as self-decoding high-capacity error correcting codes. *Advances in neural information processing systems* **32** (2019).

[102] B. Derrida, E. Gardner, A. Zippelius, An Exactly Solvable Asymmetric Neural Network Model. *Europhysics Letters (EPL)* **4** (2), 167–173 (1987), doi:10.1209/0295-5075/4/2/007.

[103] F. P. Battaglia, A. Treves, Attractor neural networks storing multiple space representations: A model for hippocampal place fields. *Physical Review E* **58** (6), 7738–7753 (1998), doi:10.1103/PhysRevE.58.7738.

[104] R. Monasson, S. Rosay, Crosstalk and transitions between multiple spatial maps in an attractor neural network model of the hippocampus: Phase diagram. *Physical Review E* **87** (6) (2013), doi:10.1103/PhysRevE.87.062813.

[105] D. Spalla, A. Dubreuil, S. Rosay, R. Monasson, A. Treves, Can Grid Cell Ensembles Represent Multiple Spaces? *Neural Computation* **31** (12), 2324–2347 (2019).

[106] E. Gardner, The space of interactions in neural network models. *Journal of physics A: Mathematical and general* **21**, 257 (1988).

[107] M. Gillett, U. Pereira, N. Brunel, Characteristics of sequential activity in networks with temporally asymmetric Hebbian learning. *Proceedings of the National Academy of Sciences* **117** (47), 29948–29958 (2020).

[108] D. Spalla, I. M. Cornacchia, A. Treves, Continuous attractors for dynamic memories. *eLife* **10**, e69499 (2021), doi:10.7554/eLife.69499.

[109] H. Tahir Chaudhry, J. A. Zavatone-Veth, D. Krotov, C. Pehlevan, Long sequence Hopfield memory. *Journal of Statistical Mechanics: Theory and Experiment* **2024** (10) (2024), doi: 10.1088/1742-5468/ad6427.

[110] A. Maes, M. Barahona, C. Clopath, Learning compositional sequences with multiple time scales through a hierarchical network of spiking neurons. *PLOS Computational Biology* **17** (3), e1008866 (2021), doi:10.1371/journal.pcbi.1008866.

[111] T. Asabuki, P. Kokate, T. Fukai, Neural circuit mechanisms of hierarchical sequence learning tested on large-scale recording data. *PLOS Computational Biology* **18** (6), e1010214 (2022), doi: 10.1371/journal.pcbi.1010214.

[112] H. S. Seung, How the brain keeps the eyes still. *Proceedings of the National Academy of Sciences of the United States of America* **93** (23), 13339–13344 (1996).

[113] J. J. Hopfield, Understanding Emergent Dynamics: Using a Collective Activity Coordinate of a Neural Network to Recognize Time-Varying Patterns. *Neural Computation* **27** (10), 2011?2038 (2015), doi:10.1162/NECO$_{a0}$0768.

[114] J. M. Fuster, G. E. Alexander, Neuron Activity Related to Short-Term Memory. *Science* **173** (3997), 652–654 (1971), doi:10.1126/science.173.3997.652.

[115] K. Kubota, H. Niki, Prefrontal cortical unit activity and delayed alternation performance in monkeys. *Journal of neurophysiology* **34** (3), 337–347 (1971).

[116] P. Goldman-Rakic, Cellular basis of working memory. *Neuron* **14** (3), 477–485 (1995), doi: 10.1016/0896-6273(95)90304-6.

[117] H. K. Inagaki, L. Fontolan, S. Romani, K. Svoboda, Discrete attractor dynamics underlies persistent activity in the frontal cortex. *Nature* **566** (7743), 212–217 (2019), doi:10.1038/s41586-019-0919-7.

[118] S. S. Kim, H. Rouault, S. Druckmann, V. Jayaraman, Ring attractor dynamics in the Drosophila central brain. *Science* **356** (6340), 849–853 (2017), doi:10.1126/science.aal4835.

[119] N. Brunel, Is cortical connectivity optimized for storing information? *Nature Neuroscience* **19** (5), 749–755 (2016), doi:10.1038/nn.4286.

[120] J. Chapeton, T. Fares, D. LaSota, A. Stepanyants, Efficient associative memory storage in cortical circuits of inhibitory and excitatory neurons. *Proceedings of the National Academy of Sciences* **109** (51), E3614–E3622 (2012), doi:10.1073/pnas.1211467109.

[121] M. M. Halassa, S. M. Sherman, Thalamocortical Circuit Motifs: A General Framework. *Neuron* **103** (5), 762–770 (2019), doi:10.1016/j.neuron.2019.06.005.

[122] S.-I. Terada, K. Kobayashi, M. Matsuzaki, Transition of distinct context-dependent ensembles from secondary to primary motor cortex in skilled motor performance. *Cell Reports* **41** (3), 111494 (2022), doi:10.1016/j.celrep.2022.111494.

[123] K. D. Harris, G. M. G. Shepherd, The neocortical circuit: themes and variations. *Nature Neuroscience* **18** (2), 170–181 (2015), doi:10.1038/nn.3917.

[124] S. Moberg, N. Takahashi, Neocortical layer 5 subclasses: From cellular properties to roles in behavior. *Frontiers in Synaptic Neuroscience* **14**, 1006773 (2022), doi:10.3389/fnsyn.2022.1006773.

[125] M. E. Larkum, Top-down Dendritic Input Increases the Gain of Layer 5 Pyramidal Neurons. *Cerebral Cortex* **14** (10), 1059–1070 (2004), doi:10.1093/cercor/bhh065.

[126] T. J. Buschman, E. K. Miller, Serial, covert shifts of attention during visual search are reflected by the frontal eye fields and correlated with population oscillations. *Neuron* **63** (3), 386–396 (2009).

**Acknowledgments** We are grateful to Gianluigi Mongillo for discussions and a careful reading of the manuscript.

**Funding** This work was supported by grants from the Fyssen foundation and the French National Research Agency (ANR) under the project Cano-T (ANR-23-CE37-0007) awarded to A.D.

**Author contributions** A.D. designed and performed research. A.L. discussed analysis of trained ANN, R.M. discussed analytical calculations. A.D. wrote the manuscript with inputs from all authors.

**Competing interests** There are no competing interests to declare.

**Data and materials availability:** All codes, for training ANN, solving analytical equations and generating figures, will be made available on a public repository at the time of publication.

# 5 Methods

## 5.1 Artificial neural networks

We trained networks of rate units that are discretized version of the continuous time equations

$$\tau\dot{\vec{x}} = -\vec{x} + W_{rec}\phi(\vec{x}) + W_{in}\vec{I}(t) + \sigma_\eta\vec{\eta}(t) \quad (1)$$

$$\vec{z} = W_{out}^T\phi(\vec{x}) \quad (2)$$

where $\vec{x} \in \mathbb{R}^N$ is the vector of currents to the $N$ neurons composing the network. Currents are injected in the network by $N_{in}$ input neurons $\vec{I}(t) \in \mathbb{R}^{N_{in}}$ through $N_{in}$ input weight vectors collected in the matrix $W_{in} \in \mathcal{M}(N_{in}, N)$. We divide inputs into two categories $W_{in}\vec{I} = W_{in}^{init}\vec{I}^{init} + W_{in}^{Go}\vec{I}^{Go}$, where $W_{in}^{init}\vec{I}^{init}$ stands for input used to initialize networks into specific stable states and $W_{in}^{Go}\vec{I}^{Go}$ for inputs triggering transitions between stable states. Currents flow in the network through the recurrent matrix $W_{rec} \in \mathcal{M}(N, N)$. These currents are transformed into firing rates through the activation function $\phi(.)$ which we take to be the rectified linear unit except for rank-restricted networks for which we used the hyperbolic tangent that leads to more interpretable mean-field equations. $N_{out}$ output neurons $\vec{z} \in \mathbb{R}^{N_{out}}$ read linear projections of the firing rate vector $\phi(\vec{x})$ through $N_{out}$ output weight vectors collected in the matrix $W_{out} \in \mathcal{M}(N, N_{out})$. $\vec{\eta}(t)$ is a vector of independent white noises, with standard deviations $\sigma_\eta = 0.05$.

Training is performed using the Pytorch library by tuning specified connectivity parameters so as to minimize a loss $\mathcal{L} = \sum_{t,k} ||\vec{z}_t^k - \hat{\vec{z}}_t^k||^2$, where $k$ indexes trials in the input-output dataset and $t$ indexes discretized time (we used a discretization time step of 20ms, for a time constant $\tau = 100$ms). $\hat{\vec{z}}_t^k$ represents the desired output activations, while $\vec{z}_t^k$ represents the actual output activations. Minimization is performed with stochastic gradient descent using the Adam optimizer with standard parameters. The choice of connectivity parameters used to minimize the loss depends on the specific training set-ups we used.

## 5.2 Training set-ups

### 5.2.1 Training NLSC

Networks are segregated into $N = N_{dic} + N_{gate}$ neurons through constraints on the connectivity (SI Fig. 2a). The $N_{dic}$ neurons receive inputs from the initializing inputs $\vec{I}^{init}$ and are connected to the readouts, while the $N_{gate}$ neurons receive inputs from the transition inputs $\vec{I}^{Go}$ and do not connect to the readouts. The $N_{dic}$ dictionary neurons are recurrently connected with each others, while the $N_{gate}$ gate neurons are not, though gate neurons are bidirectionally coupled with dictionary neurons. Dictionary connectivity parameters $W_{in}^{init}, W_{rec}^{dic}$ are trained on the dictionary task to implement $P$ stable states (Method 5.3.1) while output connectivity $W_{out}$ are fixed with readout neurons receiving from $P$ non-overlapping subsets of dictionary neurons to ease interpretation of trained networks. Input connectivity to gate $W_{in}^{Go}$ and gate/dictionary connectivities $W^{dg}, W^{gd}$ are trained to implement task-specific transitions between dictionary states. All initial connectivity parameters are drawn from Gaussian distributions. The time constant of gate neurons is

decreased from $100$ to $20$ms to obtain sharper state transitions. Except when notified, $N_{dic}$ is equal to $41$ multiplied by the number of dictionary states and the number of gate neurons equals the number of dictionary neurons.

#### 5.2.2 Training unconstrained networks

Neurons are not segregated into dictionary and gate neurons, and we train an unconstrained recurrent matrix $W_{rec}$ to implement stable states and task-specific transitions (SI Fig. 2b). Input weights $W_{in}^{init}$ and output weights $W_{out}$ are fixed to values obtained by independently training a network on the dictionary task, while transition inputs $W_{in}^{Go}$ target all neurons with a fixed connectivity weight drawn from a centered normalized Gaussian distribution.

#### 5.2.3 Training rank-restricted networks

Recurrent matrices of rank-K are parametrized as $W_{rec} = \sum_{k=1}^{K} \vec{m}_k \vec{n}_k^T$ with $\vec{m}_k \in \mathbb{R}^N$. The rank K corresponds to the number of dictionary states, and connectivity vectors $\{\vec{m}_k\}, \{\vec{n}_k\}$ are initialized as Gaussian vectors structured into populations ((40), SI 6.2) to instantiate the relevant neural architectures. Each population is composed of $512$ neurons. Values of the connectivity vectors are then fine-tuned to perform the task of interest by performing gradient descent of the loss parametrized with the $\{m_{k,i}, n_{k,i}\}_{k=1,\dots,K}^{i=1,\dots,N}$ instead of the $\{W_{ij}\}_{i,j=1,\dots,N}$.

#### 5.2.4 Assembling NLSC

Here we describe how we obtained the networks in Fig. 4,5. The overall process consists in splitting the behaviors into individual transitions and then in training individual NLSC with appropriate representations and transitions, wiring different NLSC with each others with our understanding of population structure of gate neurons (e.g. Fig. 2c,f), and fine tuning the connectivity by performing gradient descent on longer and longer behavioral sequences.

In order to train individual NLSC we use the procedure of 5.2.1 with the addition that gate populations are segregated into sub-populations, one per dictionary states permutation for instance. Two NLSC are then connected with each others by setting the connectivity from cell-assemblies of the dictionaries to gate sub-populations. We do so by specifying the initial wiring and fine-tuning it by training the circuit on individual sub-tasks that contains coordinated transitions in the two NLSC. For networks assembled to perform the tasks of Fig. 5, we augmented the stability of dictionary states by adding a saturating non-linearity to the rectified linear activation function using $\phi(.) = Hardtan[0, 1](.)$, which allowed ANN to reliably perform the large number of transitions required for running the associated programs. Programs are then run thanks to an external go cue that triggers a programmed transition each time it is activated. We also reduced the number of neurons per dictionary states from 41 to 20 to keep relatively small network sizes.

For each task for which we used this procedure, we provide, in the tasks descriptions below, more details about how NLSC are build and wired to each others.

## 5.3 Descriptions of individual tasks

In all tasks, networks are first trained on trials corresponding to individual transitions making up the behavior of interest. These trials have a fixation epochs of duration $T_{fix} = 100$ms, followed by a *cueing epoch* of duration $T_{cue} = 100$ms where the dictionary is set into its initial state through activation of input neurons $\vec{I}^{init}$. It is followed by a *maintenance epoch* of durations randomly drawn at each trial $T_{main.} \in [500, 2500]$ms and a *go epoch* of duration $T_{go} = 200$ms during which the go signal is applied to the network $\vec{I}^{Go}$, followed by a *response epoch* of duration $T_{resp} = 1000$ms. For tasks involving sequences longer than two dictionary states (e.g. Fig. 4a), in order to avoid dying out of stable states, networks are retrained to perform the longer sequences in single trials.

### 5.3.1 Dictionary task

In order to build a dictionary of $P$ states, we define an initializing input vector $\vec{I}^{init}$, and a readout vector $\vec{z}$ of size $P$. Individual input neurons $j \in [1, ..., P]$ are activated at a value 1 during the *cueing epoch* if the trial starts with maintaining dictionary state $X_j$ with a target activation of 1 for output neuron $j$. Here, and only for this task, we do not introduce additional transition inputs $\vec{I}^{Go}$, rather initializing input neurons can also be activated during the *go epoch*. If input neuron $k \in [1, ..., P]$ is activated at 1 during the go epoch, we trained networks to have output neuron $k$ activated during the response epoch, while if it is activated at 0.5, output neuron $j$ has to remain active during the response epoch (SI Fig. 2c).

Networks trained on this task develop positive feedback loops among neurons belonging to the same cellular assembly, defined as neurons projecting to the same readout neuron, as well as inhibition among neurons belonging to different cellular assemblies. This connectivity structure leads to attractor dynamics making dictionary states robust stable states (e.g. SI Fig. 2e).

### 5.3.2 Sensory-motor transformation task

Two initializing inputs $\vec{I}^{init} \in \mathbb{R}^2$ initialize the network in either state $A$ or $C$. Upon activation of the single go input neuron $I^{Go}$, the network is asked to transit to state $B$ or $D$ (SI Fig. 2d).

We trained NLSC on this task. Their dictionary network is characterized by positive feedback loops among cellular assemblies associated to each dictionary state, with symmetric interactions among all states (SI Fig. 2e). With such a connectivity, recurrent currents generated by the maintenance of state $A$ (or $C$) are similar onto neurons of state $B$ and state $D$ (SI Fig. 2f right-top). By contrast, specific patterns of gate neuron activation during the *go epoch* (Fig. 2f left) produce asymmetric currents onto cell-assemblies $B$ and $D$ (SI Fig. 2f, right bottom), triggering a transition towards the correct next cell-assembly.

We also trained 50 unconstrained networks on this task. They share similar characteristics to those of the NLSC, with positive feedback loops among cell-assemblies associated to each dictionary state (SI Fig. 2g). Neurons belonging to cell-assemblies $C$ or $D$ playing the role of gate neurons during transitions from $A$ to $B$ (SI Fig. 2h, left) and sending transient asymmetric currents to cell-assemblies $B$ and $D$ (SI Fig. 2h, right bottom). In addition to this source of asym-

metry, inhibitory connectivity between neurons belonging to different cell-assemblies (SI Fig. 2g) are asymmetric, contributing to the difference in currents to cell-assemblies $B$ and $D$ at time of transition (SI Fig. 2h, right top).

In fact, for this specific task, simply shaping the direct connectivity among cell-assemblies is enough, without the need to resort to gate neurons. This is shown by the network of SI Fig. 2i whose connectivity is initialized by adding strong inhibition from cell-assembly $A$ to $D$ and from $C$ to $B$, and which performs the task without gate neurons (Fig. 2j left). In this case, the symmetric go cue provides the same level of excitation onto cell-assemblies $B$ and $D$ (SI Fig. 2j right bottom), but in trial $A$, $B$ is less inhibited than $D$ throughout the maintenance epoch (SI Fig. 2j right top) and thus $B$ activates.

Training a rank $4$ network on the task allows to get the compact descriptions of the recurrent dynamics of Fig. 1f. It is characterized by 4 cognitive variables that take non-zero values at specific times (SI Fig. 2k), with functional connectivities between them that are controlled by the go cue (SI Fig. 2l) enabling neural activity to flow from active cell-assembly $A$ to $B$ or from $C$ to $D$.

#### 5.3.3 Single permutation over dictionary states

$P = 20$ initializing input neurons $\vec{I}^{init} \in \mathbb{R}^P$ send input to the networks, with a single of them $k_0 \in [1, ..., P]$ activated to 1 during the *cueing epoch*, instructing output neuron $z_{k_0}$ to be active during the *maintenance epoch*. Upon activation of the single go input neuron $I^{Go}$, the network is asked to transit from state $X$ to state $X+1$ as reported by activation of the output neurons $\vec{z}_{k_0+1}$ (Fig. 2m).

We trained NLSC on this task (Fig. 1f,g) and report their stereotyped behavior through an example network. For each transition a gate activation pattern is obtained by averaging the firing rate of each gate neuron over the *go epoch* (red vertical lines in SI Fig. 2k bottom). Examining pairs of transitions reveals they are associated with orthogonal gate activation patterns as shown in SI Fig. 2m. The distribution of firing rate of gate neurons at transitions is reported in SI Fig. 2l. To report statistics on the patterns of activations of gate neurons, we defined a gate neuron to be active in a transition if its firing rate is larger than $0.05$. In the NLSC implementations, training the dictionary onto the dictionary task leads to positive feedback loops among neurons belonging to the same cell-assembly, with inhibition between neurons belonging to different cell-assemblies. As expected from this training set up the inhibition is symmetric and independent on the transition path, i.e. state $X$ does not inhibit less state $X + 1$ compared to other states (SI Fig. 2o). As a result, currents coming from a cell-assembly activated during the *maintenance epoch* are self-exciting and equally inhibit all other states (SI Fig. 2p right top). On the other hand, currents produced by activation of the gate patterns send excitatory inputs to dictionary neurons belonging to the next cell-assembly to be activated (SI Fig. 2p right bottom).

We also trained 50 unconstrained networks on this task. They develop positive-feedback loops among neurons activated in a same cell-assembly (SI Fig. 2q). An asymmetry in inhibitory connectivity is observed, with less inhibitory connectivity from neurons of cell-assembly $X$ to neurons of cell-assembly $X + 1$ compared to the other (SI Fig. 2q). This leads currents produced by the maintenance state to provide *slightly* less inhibition to neurons belonging to the next cell-

assembly to be activated (SI Fig. 2r right top). At transition time, neurons associated neither with the maintenance state nor the response state are activated (Fig. 1r left), leading to provide *largely* more excitatory currents onto neurons of the response state (78% of the asymmetry in currents, SI Fig. 2r right bottom). In all the trained networks transient activations during the *go epoch* are necessary to trigger the transitions from one cell-assembly activated to another. This is shown in SI Fig. 2s where we report that selectively inactivating patterns of the 50 most transiently active cells at transition $k_0$ disrupts this transition but not the others.

#### 5.3.4 Head-direction system and multiple permutations over dictionary states

Here we consider a task where networks are trained to switch between $P$ states according to $P_c > 1$ permutations on the $P$ states. To model the head-direction system, we considered the case $P_c = 2$, with the two permutations being the $+1$ shift and the $-1$ shift once the $P = 20$ states are ordered on a ring (SI Fig. 3a). $P$ initializing inputs $\vec{I}^{init} \in \mathbb{R}^P$ send input to the networks, with a single input neuron $k_0 \in [1, ..., P]$ activating to 1 during the *cueing epoch*, instructing output neuron $z_{k_0}$ to be active during the *maintenance epoch*. Two inputs can trigger transitions $\vec{I}^{Go} \in \mathbb{R}^{P_c=2}$. Upon activation of one of this input neurons during the *go epoch*, the network is asked to transit from state $X$ to either state $X + 1$ as reported by activation of the output neurons $z_{k_0+1}$ either to state $X - 1$ as reported by activation of the output neurons $z_{k_0-1}$. In this case, the number of permutations is much lower than the number of states $P_c \ll P$.
We also trained networks on a task where $P_c \gg P$, implementing all the possible $P_c = 4! = 24$ permutations on a set of $P = 4$ dictionary states (SI Fig. 3h).

We trained NLSC on the first task and report their stereotyped behavior through an example network. For all the $2 \times 20$ transitions we analyzed gate activation patterns that can be visualized in the bottom part of SI Fig. 3b. To understand their organization we note that the trained connectivity $W_{in}^{Go}$ composed of two vectors gets structured upon training, with some gate neurons being preferentially targeted by one of the two transition inputs in $\vec{I}^{Go}$, SI Fig. 3c top. Reordering gate neurons by splitting them into two groups, according to whether they receive more inputs from the clockwise or counter-clockwise input, reveals the structure of the neural activity patterns of gate neurons (SI Fig. 3c bottom). We confirmed that this structure plays a functional role by performing inactivation experiments: inactivating the first population of gate neurons during the *go epoch* prevents clockwise transitions (SI Fig. 3d top) and leaves unaffected counter-clockwise transitions (SI Fig. 3d bottom), as summarized in SI Fig. 3e.

We also trained 50 unconstrained networks on this task. They develop positive-feedback loops among neurons activated in a same cell-assembly. As expected no asymmetry in inhibitory connectivity is observed from state $X$ to states $X+1$ and $X-1$, although these states are less inhibited than the others (SI Fig. 3f). This leads currents produced by the maintenance state (SI Fig. 3g left) to provide the same amount of inhibition to neurons belonging to the cell-assemblies associated with $X + 1$ and $X - 1$ (SI Fig. 3g right top). At transition time, a pattern of neurons associated with other states (Fig. 3g left), specific to the activated go cue, decides the following cell-assembly to be activated (SI Fig. 3g right bottom).

We trained a NLSC on the second task with $P_c \gg P$ (SI Fig. 3h) and report their stereotyped behavior through an example network. We take $N_{dic} = 41 * 4$ and $N_{gate} = 1640$. To obtain successful training, we scaled-up initial inputs $\vec{I}^{Go}$ and scaled-down input vectors $W_{in}^{Go}$ by a factor 100, allowing to balance learning speeds in $W^{dg}$ and $W_{in}^{Go}$. In order to understand how training shapes neural activity in the gate neurons (SI Fig. 3i bottom), we examined the average connection strength from dictionary neurons associated to one of the $P = 4$ states to individual gate neurons. SI Fig. 3j top shows that this connectivity gets structured during training, such that each gate neuron could be meaningfully assigned to one of the dictionary state by grouping the $N_{gate}$ gate neurons into $P = 4$ clusters using a Gaussian mixture clustering algorithm (SI Fig. 3j bottom). Reordering the X-axis of the raster of gate patterns according to these four groups, as well as the Y-axis, grouping transitions originating from the same dictionary states, reveals the organization of gate patterns (SI Fig. 3k top). In comparison, reordering the raster after performing clustering on averaged connection strengths from individual gate neurons to dictionary neurons associated to one of the $P = 4$ states, did not show any sign of meaningful structure (SI Fig. 3k bottom). We confirmed that the meaningful structure obtained by clustering plays a functional role by performing inactivation experiments: inactivating, during the *go epoch*, the population of gate neurons associated to state $X$ disrupts transitions where $X$ is a maintenance state but not the other transitions (SI Fig. 3l).

#### 5.3.5 Long-distance dependencies

In a first task we consider five symbols $A, A', B, C, C'$ that have to be produced in two sequences $S_1 \to ABC$ and $S_2 \to A'BC'$. A first initializing input $\vec{I}^{init} \in \mathbb{R}^2$ initialize the dictionary in either state $A$ or $A'$. Upon a first activation of the single go input neuron $I^{Go}$, the network is asked to transit to state $B$, and then to $C$ or $C'$ upon a second activation. In order to train on a task with long-distance-dependencies relying on more transitions, we introduced a second task on a set of $P = 20$ states with initializing inputs $\vec{I}^{init} \in \mathbb{R}^{22}$ setting the network in one of the state and indicating at each trial one of two contexts by activating either input neuron 21 or 22 during the *cueing epoch*. Again the 20 states are thought of as being arranged on a ring, and depending on context the network is asked, upon activation of a single go cue input, to transit in the clockwise or counter-clockwise direction. We first trained networks on individual transitions, and then on longer sequences up to 20 states.

A 5-states NLSC augmented with an external-memory trained on the first task has been presented in Fig. 3b,c. In order to obtain this network, we initialized the network with the NLSC, with random dictionary-gate connectivity, and added two populations of 41 neurons each preferentially activated by inputs activating either $A$ or $A'$. The network is then trained to perform the task using Method 5.2.2. We showed how gate neurons activated in a context-dependent manner upon transiting out of state $B$. Manipulating these gate neurons allow to control transitions: in a trial $S_1$, inhibiting the 50 most active gate neurons in transitions $B \to C$ and exciting the 50 most active neurons in $B \to C'$ triggers a transition from $B$ to $C'$ instead of $C$ (not shown).

For the second task, in figures SI Fig. 4e,f, we show the neural activity of a NLSC augmented with an external-memory while producing sequences of 20 states in either of the two contexts.

SI Fig. 4b,c shows an analysis of gate patterns at transitions. Similarly to networks trained on the head-direction task, upon training gate neurons segregate into two populations, with each population being preferentially inhibited by neural activation encoding memory maintenance of one of the two contexts. Inactivations during the *go epoch* confirmed that the observed population structure of gate patterns supports context-dependent transitions SI Fig. 4d.

We also trained unconstrained networks on this task. In SI Fig. 4i,j we show the neural activity of such a network in the two contexts. From the rasters of neural activity, it is difficult to grasp how the memory of context is maintained throughout production of the sequence. SI Fig. 4k shows that the two neural states corresponding to the production of symbol $X$ in the two different contexts are different, as opposed to NLSC networks SI Fig. 4g. The contextual memory is encoded in modulations of firing rates as observed in (47).

#### 5.3.6 Chunking

Here we explain how we built the neural architecture of Fig. 4b by assembling NLSC (Method 5.2.4). The full task is to produce two sequences $S_1$ and $S_2$. Each sequence is composed of $P_2 = 8$ chunks $A, B, C, D, E, F, G, H$ played in reverse order in the two sequences. Each chunk corresponds to the production of a sequence of $P_1 = 20$ states, starting with the same start and stop states. For instance chunk $A$ is associated with the sequence of states $Start, a, b, ..., r, Stop$, and other chunks are obtained by permuting symbols $a, b, ..., r$ without repeat (SI Fig. 5a,b top). A network with 20 states is trained on the dictionary task. $P_2 = 8$ populations of gate neurons are associated to this dictionary, each being trained to implement one of the $8$ sequences on the dictionary, giving us the low-level NLSC (Fig. 4b, blue). A second high-level NLSC (Fig. 4b, green) with 8 states in its dictionary and 2 gate populations is trained to implement the forward and backward transitions between chunks. A third dictionary with two states, or two sustained external inputs, encode sequence identity. These networks are then coupled with each others in the following way: each cell-assembly associated with a chunk inhibits all gate populations of the low-level NLSC except one ; low-level states, except the stop state, inhibit the two gate populations of the high-level NLSC ; each of the two gate populations of the high-level NLSC is inhibited by one of the two states encoding sequence identity. The full network is then trained to perform each individual transition simultaneously in the two NLSC in response to activation of the single go cue input neuron that targets all gate populations. It is then retrained to produce longer sequences of length 20 in response to 19 successive activations of the go cue.

#### 5.3.7 Algebraic patterns

Here we explain how we assembled the neural architecture of Fig. 4g. The task is to produce all possible sequences of readout activations of the form XYX, where the roles X and Y can be filled by one out of $5$ symbols, $A, B, C, D, E$. We first build a dictionary with 5 states that we duplicated to obtain the two memory slots. Each dictionary is associated with a set of *output* gate neurons unidirectionally excited by one of the 5 cell-assemblies. We trained another NLSC with a 3-states dictionary to produce the sequence $Q_1 \rightarrow Q_2 \rightarrow Q_3$. Each of this state is inhibiting the gates associated to the memory slot that should not be played. The full network is then retrained to perform all the possible 3 symbols sequences. In order to have this network perform another

sequence (e.g. XXY), we simply add a new gate population to the NLSC in order for it to produce a new sequence (e.g. $Q_1 \rightarrow Q_3 \rightarrow Q_2$).

#### 5.3.8 Foraging task

An agent is foraging for food in an environment that consists of $P = 10$ locations linked through $nb_{paths} = 3$ different paths (Fig.5a). The working memory of the agent is initially loaded with food values associated to each location. It then mentally travels along the 3 paths and counts, for each path, the distance $K_{path}$ from its current location to the nearest position with food available. It selects the path with shortest distance from food, travels along this path and consumes food to decrease its hunger level. It repeats these steps until the hunger level reaches 0. The task is summarized in the pseudo-code of SI Section 6.3.

In order to build the neural architecture implementing this program, we proceeded incrementally as described in Method 5.2.4, such that a go cue triggers relevant transitions inside the head, external memory and machine circuits. The final network is cartooned in SI Fig. 8e. We detail the assemblage of these circuits in SI Section 6.4. Generically, variables of the pseudo-code are each implemented by a NLSC, with a number of dictionary states corresponding to the number of states the variable can be in. Interactions between variables throughout the behavioral program are set by connectivity weights from and to gate neurons. NLSC are also used to temporally scaffold interactions between variables (e.g. implementing **for** or **while** loops).

#### 5.3.9 Sensory-motor hierarchy

In this task an agent is also decreasing its hunger level, leveraging 4 subroutines: dress the table, serve and consume food, bring new food, undress the table (see SI Section 6.5 for pseudo-codes). The network we built to solve the task and the environment on which it operates is cartooned in SI Fig. 8f. We first built individual neural Turing machines for each of the 4 subroutines, with the external memory here modeling the environment (a table for the dress and undress the table subroutines, a plate and a dish for the serve and consume food subroutine, a fridge for the bring new food subroutine). The machine circuits of the these low-level neural Turing machines serve as external memory cells of a higher-level neural Turing machine (circuits in both blue and green squares, Fig. 8f) whose machine states implement the routine coordinating the 4 subroutines. We detail the building of each of these neural Turing machines in SI Section 6.6.

#### 5.3.10 Motor sequence without go cue

Here we trained NLSC on the same permutation task as described in Method 5.3.3, except that there is not external go cue instructing the network to transit. Instead, the loss function is such that the network is asked to maintain each state for a predefined duration $T = 250, 500, 750, 1000, 1500$ms. The power spectrum of the average activity of gate neurons (Fig. 6c) is obtained using the *welch* algorithm of the *SciPy* python package with the *hann* window. In order to obtain Fig. 6d,e,f we perform a PCA analysis aligning individual dictionary neurons on a single temporal window of duration $T' = 1.2T$. Instead of looking for population modes as traditionally done, we defined

temporal modes by performing PCA on a correlation matrix of size $\frac{T'}{dt} \times \frac{T'}{dt}$ obtain from the aligned raster of dictionary neurons activation.

## 5.4 Mean-field theory for rank-restricted networks

In order to visualize neural activity as trajectories in state space for the network implementing the sensory-motor task (Fig. 1f) and the production of two sequences with long-distance dependencies (Fig. 3d,e), we built rank-restricted networks with Gaussian vectors segregated into populations (40). Their dynamical behavior **(1)** can be reduced to interactions between cognitive variables, which characterize the location of neural activity along directions $\vec{m}_k$:

$$\kappa_k(t) = \vec{m}_k^T \vec{x}(t) \tag{3}$$

For the sensory-motor transformation task, the dynamics is reduced to a dynamics over four variables, with a specific form obtained by shaping the connectivity matrix (SI Section 6.2):

$$\begin{aligned}
\tau \dot{\kappa_A} &= -\kappa_A + \tilde{\sigma}_{AA}\kappa_A \\
\tau \dot{\kappa_B} &= -\kappa_B + \tilde{\sigma}_{BB}\kappa_B + \tilde{\sigma}_{BA}\kappa_A \\
\tau \dot{\kappa_C} &= -\kappa_C + \tilde{\sigma}_{CC}\kappa_C \\
\tau \dot{\kappa_D} &= -\kappa_D + \tilde{\sigma}_{DD}\kappa_D + \tilde{\sigma}_{DC}\kappa_C
\end{aligned} \tag{4}$$

where the functional connectivities $\tilde{\sigma}_{XY}$ are non-linear functions of the four cognitive variables and the go cue input, these functions are parametrized by connectivity's summary statistics. In particular the go cue input, by setting the gain of neurons in the gate population from 0 to 1 (SI Fig. 1d), is able to control the values of $\tilde{\sigma}_{BA}$ and $\tilde{\sigma}_{DC}$ (SI Fig. 1b) and to trigger the transitions from the state $(\kappa_A = c, \kappa_B = 0, \kappa_C = 0, \kappa_D = 0)$ to the state $(\kappa_A = 0, \kappa_B = c, \kappa_C = 0, \kappa_C = 0)$ and from $(\kappa_A = 0, \kappa_B = 0, \kappa_C = c, \kappa_D = 0)$ to $(\kappa_A = 0, \kappa_B = 0, \kappa_C = 0, \kappa_D = c)$ (SI Fig. 1a), with $c = O(1)$ a constant.

A similar construction can be proposed to solve the task with long-distance dependencies. Here we have five cognitive variables $\kappa_A, \kappa_{A'}, \kappa_B, \kappa_C, \kappa_{C'}$ associated with the symbols to be produced and two cognitive variables associated with the two hidden states maintaining the identity of the sequence in working memory $\kappa_{hA}, \kappa_{hA'}$. In particular the dynamics of $\kappa_C$ writes

$$\tau \dot{\kappa_C} = -\kappa_C + \tilde{\sigma}_{CC}\kappa_C + \tilde{\sigma}_{CB}\kappa_B \tag{5}$$

where $\tilde{\sigma}_{CB}$ is modulated from 0 to 1 by the go cue only if $(\kappa_{hA} = 1, \kappa_{hA'} = 0)$. There is a symmetric equation for $\kappa_{C'}$, and other cognitive variables evolve as in **(4)**. Flow fields plotted in SI Fig. 1c and SI Fig. 1e,f represent the time evolution of cognitive variables $\dot{\vec{\kappa}}$ obtained from e.g. equation **(4)**, each panel is obtained using the *streamplot()* function of *matplotlib* on $(\dot{\kappa}_X, \dot{\kappa}_Y)$.

## 5.5 Capacity of neural architectures

Our results on the minimal number, $N_{min}$, of neurons required to implement NLSC-based architectures build on previous results obtained for perceptrons with binary neurons and binary

synaptic weights (91; 92; 93; 94). We consider $N_{out}$ perceptrons with $N_{in}$ input neurons. The $P_a$ input patterns $\{\xi_i^\mu\}_{i=1,...,N_{in}}^{\mu=1,...,P_a} \in \{0,1\}$ are drawn randomly with the constraint $\sum_{i=1}^{N_{in}} \xi_i^\mu = M_{in}$ and output activations $\{\zeta_j^\mu\}_{j=1,...,N_{out}}^{\mu=1,...,P_a} \in \{0,1\}$ are drawn randomly.

We present calculations for computing the probability that the synaptic connectivity matrix $W \in \mathbb{R}^{N_{in} \times N_{out}}$ is not overloaded by the stored associations, i.e. that the output neurons correctly classify the $P_a$ input patterns: $\Theta(\sum_{i=1}^{N_{in}} W_{ij}\xi_i^\mu - \theta) = \zeta_j^\mu$, for all $\mu = 1, ..., P_a$ and $j = 1, ..., N_{out}$, with $\Theta(.)$ the Heaviside function and $\theta = M_{in}^-$ the activation threshold. The matrix of binary synaptic weights is assumed to undergo Hebbian learning while the $P_a$ correct input and output activations are imposed onto the network: $W_{ij} = \Theta\left(\sum_{\mu=1}^{P_a} \xi_i^\mu \zeta_j^\mu\right)$. We note $g$ the fraction of synapses activated after presenting $P_a$ associations. It is a crucial quantity for our analysis as it controls the amount of interferences between associations and indicates for which value of $P_a$ the learning rule is not able to correctly store the associations anymore. It can be estimated as

$$g = 1 - (1 - \frac{M_{in}}{N_{in}} f_{out})^{P_a} \tag{6}$$

The probability that $P_a$ associations in $N_{out}$ perceptrons are learnt without errors can be expressed as (SI Section 6.1.1):

$$P_{ne}^{perc}(N_{out}, P_a, N_{in}, M_{in}, f_{out}) = \exp\left(P_a(1 - f_{out})N_{out} \ln(1 - g^{M_{in}})\right) \tag{7}$$

The neural architectures we built can be decomposed into multiple independent such perceptrons (Fig. 1l, SI Fig. 7a-d, SI Fig. 8). The probability that all the associations of the architecture are learnt without errors, $P_{ne}^{all}$, can thus be estimated by a product of the functions **(7)** estimated with the appropriate input variables. In order to estimate $N_{min}$ for a given architecture and given numbers of associations, we performed grid-search over the number of neurons and coding properties $N_{in}, M_{in}, f_{out}$ and look for sets of parameters that minimize the total number of neurons while keeping the probability of no error above a threshold $P_{ne}^{all} > 0.9$. For instance, an attractor neural network of $N$ neurons can be seen as $N$ perceptrons. For $M$ active neurons per pattern and $P$ patterns, we have $P_{ne}^{all} = P_{ne}^{perc}(N, P, N, M, \frac{M}{N})$, and we perform grid-search on parameters $N$ and $M$ to find the values that minimize $N$ keeping $P_{ne}^{attractor} > 0.9$. In SI Section 6.1, we detail this process for each of the neural architectures considered in the main text.

# 6 Supplementary

## 6.1 Capacity calculations

### 6.1.1 Capacity for perceptrons

We detail how to obtain the probability **(7)** that the $P_a$ associations are perfectly retrieved for $N_{out}$ perceptrons with input size $N_{in}$, input coding level $f_{in} = \frac{M_{in}}{N_{in}}$ and output coding level $f_{out}$ (SI Fig.7a). We do so following (94). To simplify mathematical expressions we have considered the case where the number of active neurons in the input patterns is fixed and does not fluctuate from pattern to pattern: $\sum_{i=1}^{N_{in}} \xi_i^\mu = M_{in}$ for all $\mu$'s. With this constraint, the activation threshold of the output neurons can be taken as $\theta = M_{in}^-$, and, given the learning rule leading to

$$W_{ij} = \Theta\left(\sum_{\mu=1}^{P_a} \xi_i^\mu \zeta_j^\mu\right) \tag{8}$$

we are guaranteed that no errors will be produced when retrieving the $P_a f_{out}$ associations with $\zeta_j^\mu = 1$. Errors can then only occur for associations with $\zeta_j^\mu = 0$. They occur when the output neuron receives an input $h = M_{in}$, i.e. when all the synapses between the output neuron and active input neurons in $\vec{\xi}^\mu$ have been potentiated by one of the other $P_a - 1$ associations. The probability of such an event can be simply estimated by

$$P_e := P(h \geq M_{in}^- | \zeta^\mu = 0) \simeq g^{M_{in}} \tag{9}$$

with $g$ taken from **(6)**. The probability that no errors are produced in all the $P_a$ associations and $N_{out}$ perceptrons is then given by

$$P_{ne} = (1 - P_e)^{(1-f_{out})N_{out}P_a} \tag{10}$$

for sparse enough coding levels $\frac{M_{out}}{N_{out}} \ll \frac{1}{\sqrt{N_{in}}}$ correlations between two synapses $W_{ij_1}$ and $W_{ij_2}$ can be ignored (95) and the probability of error is well estimated by **(9)**. In order to obtain more precise estimates for finite-size networks and compare with simulations (Fig. 1m, crosses) we took these correlations into account using exact calculations recapitulated in (96) equation (42):

$$P_e = \sum_{s=0}^{M_{in}} (-1)^s \binom{M_{in}}{s} \left(1 - \frac{M_{out}}{N_{out}}\left(1 - \frac{\binom{N_{in}-M_{in}}{s}}{\binom{N_{in}}{s}}\right)\right)^P \tag{11}$$

This expression is cumbersome to estimate, such that when optimizing architectures made of multiple NLSC we used **(9)** instead. SI Fig. 7h shows that the two estimates of $N_{min}$ are very close to each other and follow the same dependency on parameters. Only the parameters $M_{in}$ and $M_{out}$ at which $N_{min}$ is found are slightly different, which can be taken into account to improve the match with simulations.

### 6.1.2 Storage capacity of attractor neural networks

Here we apply the above results to compute the minimal number of neurons $N$ required to store $P$ fixed-points in an attractor neural network. This problem is equivalent to storing $P$ associations

in $N$ perceptrons with input layer of size $N$ and equal input and output coding levels $f = \frac{M}{N}$ (SI Fig.7b). Applying equation **(7)** to our problem, we can estimate the probability of no error

$$P_{ne}^{ANN}(P, N, M) = P_{ne}^{perc}(N, P, N, M, \frac{M}{N}) = \exp\left(PN(1 - \frac{M}{N})\ln(1 - g^M))\right) \tag{12}$$

In figure Fig.1m dashed line, we show how the minimal number of neurons N increases when more fixed-points are to be stored. It is obtained by computing $P_{ne}^{ANN}$ for a region of parameters $(N, M)$ and report the minimal value of $N$ such that $P_{ne} > 0.9$. Crosses represent the minimal value of N obtained with simulations, they are obtained as follows: we built $100$ attractor neural networks with parameters $N, M, P$, we tested the stability of all the $P$ patterns for each, and reported the probability that no error occurs for all the P patterns $P_{ne}$ ; for each cross we fixed $P$ as well as the value of $M$ that is found to optimize the capacity according to **(12)**, then simulated networks with various values of $N$ and reported the smallest value for which $P_{ne} > 0.9$. As expected from the infinite-size estimate (94), the minimal number of neurons required to store $P$ patterns of activity scales as $N = O(\sqrt{P})$.

### 6.1.3 Capacity for the implementation of transitions between dictionary states

We consider a dictionary network of $N_{dic}$ neurons storing $P$ fixed-points $\{\vec{\xi}^{\mu}\}_{\mu=1,\dots,P}$ with coding level $f_{dic} = \frac{M_{dic}}{N_{dic}}$, together with $N_{gate}$ gate neurons (SI Fig.7c). We want this circuit to implement a total of $T = P$ transitions between the fixed-points of the dictionary with each transition associated to a pattern of activation of gate neurons $\{\vec{\zeta}^{\mu}\}_{\mu=1,\dots,P}$ with coding level $f_{gate} = \frac{M_{gate}}{N_{gate}}$. To trigger a transition, the appearance of a go cue at time $t$ has the following two effects: a strong inhibition on gate neurons is released, so that at time $t+1$ gate neurons represent pattern $\vec{\zeta}^{\mu(t)}$ if the dictionary was in pattern $\vec{\xi}^{\mu(t)}$, as well as an increase of the activation threshold in the dictionary, so that the dictionary forgets its current pattern. With this, at time $t+2$ the dictionary is put into $\vec{\xi}^{\mu+1}$ by the pattern printed in gate neurons [1]. As explained in the main text, constraints on such an architecture comes from i) storing the $P$ fixed-points in the dictionary, ii) storing $P$ associations from dictionary to gate neurons, and iii) from gates to dictionary neurons. Learning these associations under the Hebbian learning rule for binary synapses **(8)** leads to a fraction of potentiated synapses from dictionary to gates and gates to dictionary:

$$g' = 1 - (1 - \frac{M_{in}}{N_{in}}\frac{M_{out}}{N_{out}})^P \simeq 1 - \exp(-P\frac{M_{in}}{N_{in}}\frac{M_{out}}{N_{out}}) \tag{13}$$

as in the previous section successful and efficient storage of the $P$ associations is obtained for $0 < g' < 1$. Applying **(7)** and **(12)** to i) the $N_{dic}$ perceptrons of input-size $N_{dic}$ of the dictionary, ii) the $N_{gate}$ perceptrons of input-size $N_{dic}$ from dictionary to gate, iii) the $N_{dic}$ perceptrons of input-size $N_{gate}$ from gate to dictionary we get

[1] An alternative to this lowering of the activation threshold would be to have inhibitory connections from gate to dictionary neurons so that $\vec{\zeta}^{\mu(t)}$ both activates $\vec{\xi}^{\mu+1}$ and silences $\vec{\xi}^{\mu}$.

$$
\begin{aligned}
P_{ne}^{i)} &= P_{ne}^{ANN}(P, N_{dic}, M_{dic}) \\
P_{ne}^{ii)} &= P_{ne}^{perc}(N_{gate}, P, N_{dic}, M_{dic}, \frac{M_{gate}}{N_{gate}}) \\
P_{ne}^{iii)} &= P_{ne}^{perc}(N_{dic}, P, N_{gate}, M_{gate}, \frac{M_{dic}}{N_{dic}})
\end{aligned} \tag{14}
$$

For a given value of $P$ one can then scan parameters $N_{dic}, M_{dic}, N_{gate}, M_{gate}$ to find values that minimize the total number of neurons $N_{dic} + N_{gate}$ under the constraint that $P_{ne}^{all} = P_{ne}^{i)} P_{ne}^{ii)} P_{ne}^{iii)} < 0.9$ (estimates do not dramatically depends on the chosen threshold, not shown).

The full curve in Fig.1m shows the minimal total number of neurons $N_{min}$ for storing $P$ patterns as fixed-points of the dynamics and implementing a permutation, i.e. $P$ transitions, among these fixed-points. SI Fig.7e shows variations with $P$ of the number of dictionary and gate neurons, the coding levels in the dictionary $f_{dic} = \frac{M_{dic}}{N_{dic}}$ and gate $f_{gate} = \frac{M_{gate}}{N_{gate}}$ layers, and the fraction of activated synapses among dictionary neurons and in between gate and dictionary neurons.

In the set-up considered above specific patterns of activation of gate neurons $\{\vec{\zeta}^{\nu}\}_{\nu=1,\dots,P}$ are learnt via Hebbian learning of the dictionary to gate connectivity. Instead random gate patterns could be implemented with a random binary connectivity, with inhibitory interactions among gate neurons tuning their coding level at a desired value $f_{gate} = \frac{M_{gate}}{N_{gate}}$. With such an assumption, constraint ii) on the dictionary-to-gate connectivity does not have to be taken into account anymore. Considering only constraints i) and iii), i.e. ignoring the second line in **(14)** leads to the orange curve in SI Fig.7g.

If only a subset of dictionary states are associated with a transition $T = \epsilon P < P$, the number of necessary gate neurons is reduced linearly as shown in SI Fig.7f obtained by replacing $P$ by $T$ to estimate the second and third lines of **(14)**.

#### 6.1.4 Capacity for permutations with direct hetero-associative connectivity

Most sequence models use direct hetero-associative connectivity (e.g. (97)) whereby neurons active in pattern $\mu$ directly excite neurons active in pattern $\mu + 1$. In order to compare the performance of the NLSC with such a neural architecture, we consider a model of $N$ neurons, with $M$ active neurons per patterns, connected through

$$
W_{ij} = W_{ij}^{a} + \lambda W_{ij}^{h} \tag{15}
$$

with $W_{ij}^{a} = \Theta\left(\sum_{\mu=1}^{P_a} \xi_i^{\mu} \xi_j^{\mu}\right)$ the auto-associative connectivity stabilizing neural patterns $\{\vec{\xi}^{\mu}\}$ and $W_{ij}^{h} = \Theta\left(\sum_{\mu=1}^{P_a} \xi_i^{\mu+1} \xi_j^{\mu}\right)$ the hetero-associative connectivity directing transitions from neural pattern $\vec{\xi}^{\mu}$ to $\vec{\xi}^{\mu+1}$. Compared to the NLSC, we do not have the extra gate neurons, but we expect more interferences due to the fact that both auto- and hetero-associative connectivity can activate

neurons that are supposed to remain silent. As for the NLSC model, we take an activation threshold $\theta = M^-$ when probing the stability of a neural pattern. Transitions are triggered by an external input $I_0$, uniform on all neurons, and we take an activation threshold $\theta = (I_0 + \lambda M)^-$ when probing errors in transitions. With this, active neurons in $\vec{\xi}^{\mu+1}$ are sure to be activated. Activation of $\vec{\xi}^{\mu+1}$ silences $\vec{\xi}^{\mu}$ by a mechanism that we do not model. The probability of no error for this model is computed as $P_{ne}^{all} = P_{ne}^{a} P_{ne}^{h}$ with

$$\begin{aligned} P_{ne}^{a} &= \exp\left(PN(1-\frac{M}{N})\ln(1-p_a))\right) \\ P_{ne}^{h} &= \exp\left(PN(1-\frac{M}{N})\ln(1-p_h))\right) \end{aligned} \tag{16}$$

with

$$\begin{aligned} p_a &= \sum_{k,l} \binom{M}{k} g_a^k (1-g_a)^{M-k} \binom{M}{l} g_h^l (1-g_h)^{M-l} \mathbb{1}_{k+\lambda l \geq M} \\ p_h &= \sum_{k,l} \binom{M}{k} g_a^k (1-g_a)^{M-k} \binom{M}{l} g_h^l (1-g_h)^{M-l} \mathbb{1}_{k+\lambda l \geq \lambda M} \end{aligned} \tag{17}$$

with $g_a = g_h$ the fraction of activated auto-associative and hetero-associative synapses potentiated at 1. Figure SI Fig.7i shows that $N_{min}$ is higher for all values of $\lambda$ compared to the NLSC implementation.

#### 6.1.5 Capacity for multiple permutations among dictionary states

Instead of a single go cue associated with a single permutation, here we consider $P_c$ go cues, or contextual signals, associated with $P_c$ permutations among $P$ dictionary states, $P_c = 2$ for the head-direction system. The role of gate neurons is to demix its $P_c \times P$ possible inputs ($P$ configurations of the dictionary, $P_c$ configurations of the go cue inputs) into linearly separable representations. This allows to learn the gate to dictionary connectivity so that a gate pattern elicit the correct next dictionary pattern. Training ANN in the different regimes $P_c \ll P$ and $P_c \gg P$ revealed that patterns of gate neurons organize differently in these two extreme cases. In the first regime, a go cue input activates the gain (releasing inhibition brings neuron from below up to firing threshold) of gate neurons in one of $P_c$ populations. Each dictionary pattern instead drives a pattern of activation of gate neurons in this sub-population (Fig. 2c). In the second regime, the situation is reversed, with dictionary patterns defining $P$ populations of gate neurons by gain modulation (Fig. 2f), and the $P_c$ go cue inputs driving patterns of activity in the activated sub-population. These arrangements can be understood as optimal from the capacity point of view. Indeed, the total number of gate neurons scales linearly with the number of populations, while it only scales with the square root of the number of driving patterns, thanks to sparse coding. The split into $P_c$ populations thus gives $N_g = O(P_c\sqrt{P})$ for $P_c \ll P$, while the split into $P$ populations gives $N_g = O(P\sqrt{P_c})$ for $P_c \gg P$. It remains to be seen whether there is a better way that can be proposed to mix dictionary and go cue inputs in the regime $P \simeq P_c$.

In the regime $P_c \ll P$ with $P_c$ populations of gate neurons, following the reasoning of the section above, the probability that there is no error for all the $P$ stable states and all the $P_c \times \epsilon P$ transitions can be written as

$$P_{ne}^{archi1}(P_c, \{NLSC\}) = P_{ne}^{archi1}(P_c, P, N_{dic}, M_{dic}, N_{gate}, M_{gate}, \epsilon) = P_{ne}^{i)} P_{ne}^{ii)} P_{ne}^{iii)} \tag{18}$$

with

$$\begin{aligned} P_{ne}^{i)} &= P_{ne}^{ANN}(P, N_{dic}, M_{dic}) \\ P_{ne}^{ii)} &= P_{ne}^{perc}(P_c N_{gate}, \epsilon P, N_{dic}, M_{dic}, \frac{M_{gate}}{N_{gate}}) \\ P_{ne}^{iii)} &= P_{ne}^{perc}(P_c N_{dic}, \epsilon P, N_{gate}, M_{gate}, \frac{M_{dic}}{N_{dic}}) \end{aligned} \tag{19}$$

$\{NLSC\} = \{P, N_{dic}, M_{dic}, N_{gate}, M_{gate}, \epsilon\}$ contains all the parameters defining anatomical and physiological properties of the NLSC. Compared to the previous case **(14)** we simply took into account the fact that the number of perceptrons from dictionary to gate and gate to dictionary are proportional to $P_c$. $N_{gate}$ and $M_{gate}$ here refer to the parameters of a single gate population, as is the case afterwards. To obtain $N_{min}$ we looked for parameters that minimize the total number of neurons $N_{min} = N_{dic} + P_c N_{gate}$ while keeping $P_{ne} > 0.9$.

For the regime $P_c \gg P$ all the $P^2$ possible transitions are wired into the network. To do so, all the go cues associated with the same transition drive the same pattern of neural activity into the activated population of gate neurons. This is done without any error since here neural patterns associated to each permutation are non-distributed, i.e. a single go cue neuron is activated in only one permutation, thus there are no interferences between go cue to gate neuron synaptic wires associated with different permutations. This explains the independence of $N_{min}$ on $P_c$ for this architecture (Fig. 2k, dashed line). Again the probability that there is no error for all the $P$ stable states and all the $P_c \times P$ transitions can be written as

$$P_{ne}^{archi2}(\{NLSC\}) = P_{ne}^{archi2}(P, N_{dic}, M_{dic}, N_{gate}, M_{gate}) = P_{ne}^{i)} P_{ne}^{ii)} P_{ne}^{iii)} \tag{20}$$

with

$$\begin{aligned} P_{ne}^{i)} &= P_{ne}^{ANN}(P, N_{dic}, M_{dic}) \\ P_{ne}^{ii)} &= P_{ne}^{perc}(P N_{gate}, P, N_{dic}, M_{dic}, \frac{1}{P}) \\ P_{ne}^{iii)} &= P_{ne}^{perc}(P N_{dic}, P, N_{gate}, M_{gate}, \frac{M_{dic}}{N_{dic}}) \end{aligned} \tag{21}$$

The second line is obtained considering $P$ associations to be learnt on $N_{out} = P * N_{gate}$ perceptrons, with $f_{in} N_{in} = M_{dic}$ and an output coding level $f_{out} = \frac{1}{P}$. To obtain $N_{min}$ we looked for parameters that minimize the total number of neurons $N_{min} = N_{dic} + P N_{gate}$ while keeping $P_{ne}^{archi2}(\{NLSC\}) > 0.9$.

### 6.1.6 Capacity of architectures for sequences with long-distance-dependencies

We consider $P_2$ contextual signals that, instead of being carried by non-distributed representations, are distributed over the $N_2$ neurons of an external memory with a number of activated neurons per patterns $M_2$ (SI Fig.8a). We ask the patterns of neural activity associated to each permutation in the external memory to be stable, i.e. the external memory is an attractor network. We now labeled $N_1$ and $N_{1g}$ the number of neurons in the dictionary and in each of the gate population of the NLSC, $M_1$ and $M_{1g}$ are the number of active neurons in dictionary and gate patterns, $P_1$ is the number of dictionary states: $\{NLSC_1\} = \{P_1, N_1, M_1, N_{1g}, M_{1g}, \epsilon_1\}$. Compared to the section above, we also have to take into account interferences between connectivity patterns from the external memory to the gates of the NLSC, for the regime $P_2 \ll P_1$, the overall probability of having no-errors can be written as

$$P_{ne}^{all} = P_{ne}^{archi1}(P_2, \{NLSC_1\}) P_{ne}^{ANN}(P_2, N_2, M_2) P_{ne}^{perc}(P_2 N_{gate}, P_2, N_2, M_2, \frac{1}{P_2}) \tag{22}$$

where the last two terms correspond to errors in imprinting $P_2$ attractors in the external memory and errors in modulating the $P_2$ populations of gate neurons (gate neurons have an activation threshold $\theta = M_1 + M_2$). We have used $\epsilon$ introduced above, the fraction of transitions $T = \epsilon P_1$ associated with each contextual signal.
For the case $P_2 \gg P_1$, we have

$$P_{ne}^{all} = P_{ne}^{archi2}(\{NLSC_1\}) P_{ne}^{ANN}(P_2, N_2, M_2) P_{ne}^{perc}(P_1 N_{gate}, P_2, N_2, M_2, \frac{M_{1g}}{N_{1g}}) \tag{23}$$

In the main text Fig.3h, we show $N_{min}$ obtained from equations **(22)** (full line) and **(23)** (dashed line). We compared these values of $N_{min}$ to those obtained by considering a neural architecture composed of $P_2$ NLSC with $P_1$ states in their dictionary (dotted-dashed line) for which

$$P_{ne}^{all} = \left(P_{ne}^{archi1}(1, \{NLSC_1\})\right)^{P_2} \tag{24}$$

### 6.1.7 Capacity of architectures for sequences organized into chunks

In the previous section, distinct states of the external memory provide distinct contextual inputs onto gate neurons to decide on transitions taking place in the lower-level NLSC. Instead of having only this uni-directional interaction, we would like the dictionary state of the lower-level NLSC to also trigger transitions in the external memory. To do so, we augment the external memory with gate neurons, giving us two interacting NLSC (SI Fig.8b). Each of the $NLSC_i$, $i \in \{1, 2\}$ has $P_i$ dictionary states and is composed of $N_i$ and $N_{gi}$ dictionary and gate neurons. Each state in $NLSC_i$ is associated with $T_j = \epsilon_j P_j$ transitions in $NLSC_j$. Depending on whether $P_2 \ll P_1$ or $P_2 \gg P_1$ it will be more efficient to have dictionary states of $NLSC_2$ to modulate populations of gate neurons and $NLSC_1$ to drive them or the converse, here we arbitrarily take $P_2 \ll P_1$. We have

$$P_{ne}^{all} = P_{ne}^{archi1}(P_2, \{NLSC_1\}) P_{ne}^{archi2}(\{NLSC_2\}) P_{ne}^{perc}(P_2 N_{1g}, P_2, N_2, M_2, \frac{1}{P_2}) P_{ne}^{perc}(\epsilon_2 P_2 N_{2g}, P_1, N_1, M_1, \frac{M_{2g}}{N_{2g}}) \tag{25}$$

where the last two terms correspond to the connectivity from the dictionary of $NLSC_1$ to the gates of $NLSC_2$ and from the dictionary of $NLSC_2$ to the gates of $NLSC_1$.

In order to obtain the full curve in Fig. 4d, we consider a specific case where i) a single of $NLSC_1$'s states, the stop state, triggers transitions in $NLSC_2$ and ii) the ensemble of $\epsilon_1 P_1$ transitions in $NLSC_1$ associated with a state in $NLSC_2$ are closed (e.g. $A \to O$, $O \to Z$, $Z \to E$, not $A \to O$, $B \to P$, $C \to Q$). In order to point at the fact that this construction can be iterated hierarchically, we also require that the stop-state triggers closed transitions in $NLSC_2$, and we take a total of $P_3 N_{2g}$ gate neurons associated with $P_3$ different sequences in $NLSC_2$. For the figure we take $P_1 = 31$, $P_2 = 10,000$, $\epsilon_1 P_1 = 3 + 1$ and for each of the $P_3$ sequences, the stop state is associated with $T_2 = 100$ transitions in $NLSC_2$: the system we built stores $P_3$ sentences each composed of $100$ words (out of $10,000$) each composed of $3$ phonemes (out of $30$). We found $N_{min}$ for this architecture by minimizing $N_1 + P_1 N_{1g} + N_2 + P_3 N_{2g}$ under the constraint $P_{ne}^{all} > 0.9$, with

$$\begin{aligned} P_{ne}^{all} &= P_{ne}^{archi2}(\{NLSC_1\}) P_{ne}^{ANN}(P_2, N_2, M_2) P_{ne}^{perc}(P_1 N_{gate}, P_2, N_2, M_2, \frac{M_{1g}}{N_{1g}}) \times \\ & P_{ne}^{perc}(P_3 N_{2g}, T_2, N_2, M_2, \frac{M_{2g}}{N_{2g}}) P_{ne}^{perc}(P_3 N_2, T_2, N_{2g}, M_{2g}, \frac{M_2}{N_2}) \times \\ & P_{ne}^{perc}(P_3 N_{2g}, P_1, N_1, M_1, \frac{1}{P_1}) \end{aligned} \tag{26}$$

where the first three terms correspond to equation **(23)** for the articulatory apparatus, the next two terms correspond to connections in-between the dictionary and gates of $NLSC_2$, and the last one to the connections from the stop state of $NLSC_1$ to the gates of $NLSC_2$.

We compared this implementation with one that does not make use of a stop state (Fig. 4d, dashed curve). $NLSC_1$ has one dictionary state less. In order to program transitions in $NLSC_2$ for relevant conjunctions of $NLSC_1$ and $NLSC_2$ states, gate neurons are segregated into $P_1 \ll P_2$ populations and each of the $P_1$ subpopulations will be involved with probability $\frac{1}{P_1}$ in triggering a transition in $NLSC_2$ at the end of each word. The probability of no error associated with this architecture is obtained by introducing $P_1 P_3$ populations of gate neurons in $NLSC_2$:

$$\begin{aligned} P_{ne}^{all} &= P_{ne}^{archi2}(\{NLSC_1\}) P_{ne}^{ANN}(P_2, N_2, M_2) P_{ne}^{perc}(P_1 N_{gate}, P_2, N_2, M_2, \frac{M_{1g}}{N_{1g}}) \times \\ & P_{ne}^{perc}(P_3 P_1 N_{2g}, \frac{T_2}{P_1}, N_2, M_2, \frac{M_{2g}}{N_{2g}}) P_{ne}^{perc}(P_3 N_2, \frac{T_2}{P_1}, N_{2g}, M_{2g}, \frac{M_2}{N_2}) \times \\ & P_{ne}^{perc}(P_3 P_1 N_{2g}, P_1, N_1, M_1, \frac{1}{P_1}) \end{aligned} \tag{27}$$

The scheme we presented for two hierarchical levels can be straightforwardly generalized to multiple levels by piling-up more than two NLSC and writing corresponding equations as in **(25)**.

#### 6.1.8 Capacity of architectures for algebraic patterns

We explain how we obtained the theoretical curves of Fig. 4i,j,k that quantify the number of neurons used to produce sequences with a temporal structure characterized by algebraic patterns. We

consider an external memory composed of $K$ cells, each composed of a dictionary with $P$ states, associated with $N_{gate}$ *output* gates neurons driven by the $N_{dic}$ dictionary neurons (SI Fig.8c). These $K$ populations of gates then converge onto an output network of size $N_o$ carrying neural representations used for production. In order for these representations to be manipulated by another downstream circuit we ask them to be stabilized by recurrent connectivity, i.e. the output neurons form an attractor network. The probabilities of no errors for this neural architecture is

$$\begin{aligned} P_{ne}^{all} &= \left( P_{ne}^{ANN}(P, N_{dic}, M_{dic}) P_{ne}^{perc}(N_{gate}, P, N_{dic}, M_{dic}, \frac{M_{gate}}{N_{gate}}) P_{ne}^{perc}(N_o, P, N_{gate}, M_{gate}, \frac{M_o}{N_o}) \right)^K \\ &\times P_{ne}^{ANN}(P, N_o, M_o) \end{aligned} \tag{28}$$

We compared this architecture with an alternative one that uses a single attractor neural network of size $N$ with K-items working memory, instead of using K distinct single-item working memory attractor networks. In Fig. 4j, we report a lower bound on the minimal number of neurons that have to be used to maintain these representations. The network stores a total of $KP$ patterns with $M$ active neurons per pattern. We want to test the stability of a superposition of $K$ patterns. As for the single item case, given the learning rule **(8)**, we are guarantee there is no error on foreground neurons. The probability of no error on background neurons is estimated as follows:

$$P_{ne} = \exp\left(PK(N - KM)\ln(1 - p_e)\right)) \tag{29}$$

with $N - KM$ the estimate of the number of background neurons, $PK$ is the total number of pattern on which we impose there is no error. Note this number is smaller than the total number of possible patterns $P^K$ on which the network can be in, i.e. we only evaluate the effect of increased probability of error on background neurons $p_e$ due to increased activity from $M$ to $KM$

$$p_e = \sum_{k=M}^{KM} g^k (1 - g)^{KM-k} \tag{30}$$

with

$$g = 1 - \left(1 - \left(\frac{M}{N}\right)^2\right)^{KP} \tag{31}$$

The number of gate neurons associated to the multi-item working memory is expected to be even higher than for the other architecture: the number of gate populations also scales as $K$ given the need for gate representations to be modulated by the $K$ states of the head circuit, and for each of these population, the associations are more difficult to implement given the increased amount of interferences in multi-item working memory representations.

#### 6.1.9 Capacity of neural Turing machines

The neural Turing machines we built are composed of three circuits (Fig. 5b):

An **external memory** circuit composed of $K$ cells embodied in $K$ $NLSC_{e.m.}$. We note $P_2$ the number of symbols that can appear on each of the cells, $N_2$ the number of dictionary neurons in each

NLSC ($M_2$ the associated number of active neurons composing a cell-assembly), $N_{g2}$ the number of *transition* gate neurons per sub-population used to implement transitions in each NLSC (coding level $M_{2g}$) and $N_{2g'}$ the number of *output* gate neurons used to transmit the currently supported state to the machine circuit (coding level $M_{2g'}$).

A **machine** circuit (implementing a finite-state automaton) embodied by a single $NLSC_m$. We note $P_1$ the number of machine states, $N_1$ the number of dictionary neurons in the NLSC (coding level $M_1$), $N_{1g}$ the number of transition gate neurons per subpopulation used to implement machine transitions (coding level $M_{1g}$).

A **head** circuit embodied by an $NLSC_h$ with a K-states dictionary. Gate neurons used to implement transitions between dictionary states. These gate neurons can be split into subpopulations of size $N_{gh}$ to implement $P_h$ distinct permutations of dictionary states, with a dictionary of size $N_h$. For instance, a head circuit with two subpopulations, one associated with the $+1$ shift the other with the $-1$ shift, would implement left/right head movements as for the canonical Turing machine. In order to limit the number of variables on which the optimization is performed to compute $N_{min}$, we do not take the head-circuit into account. This is justified because we consider a parameter regime where the number of cells of the external memory $K$ is small compared to the numbers of machine states $K \ll P_1$ and symbols of the external memory $K \ll P_2$.

Connections among these three circuits are as follows. The machine's state conditions symbol transitions in a cell $k$ of the external memory by connections from its dictionary to the transition gates of the $k$th NLSC. The state of the currently considered cell on the external memory is transmitted to the machine by connections from its output gates neurons to the transition gates of the machine. The head circuit routes communication between the machine and the $k$th NLSC of the external memory, this is done through connections from cell-assembly $k$ of its dictionary to transition and output gate neurons of $NLSC_k$. Transitions in the head circuit are conditioned by machine states thanks to connections from the machine cell-assemblies to the transition gates of the head circuit.

To estimate $N_{min}$ for a Turing machine with $P_2 \ll P_1$ (cf full line Fig. 5h), we implemented NLSC of the external memory with architecture 2 and the machine NLSC with architecture 1. Reusing previous calculations, the probability that there is no error in all the programmed transitions and stable states can be decomposed as

$$\begin{aligned} P_{ne}^{TM} \quad = \quad & P_{ne}^{e.m.} P_{ne}^{m.} P_{ne}^{h.} \times \\ & P_{ne}^{e.m.\to m.} P_{ne}^{m.\to e.m.} \times \\ & P_{ne}^{h.\to e.m.} \times \\ & P_{ne}^{m.\to h.} \end{aligned} \tag{32}$$

where the first line describes individual circuits, the second line describes connections between the external memory and machine circuits, the third line connections from the head to the external memory circuits, the last line the connections from the machine to the head circuit. Individual terms can be written as

$$P_{ne}^{e.m.} = \left( P_{ne}^{archi2}(\{NLSC_{e.m.}\}) P_{ne}^{perc}(N_{2g'}, P_2, N_2, M_2, \frac{M_{2g'}}{N_{2g'}}) \right)^K \tag{33}$$

$$P_{ne}^{m.} = P_{ne}^{archi1}(P_2, \{NLSC_{m.}\}) \tag{34}$$

$$P_{ne}^{h.} = P_{ne}^{archi2}(\{NLSC_{h.}\}) \tag{35}$$

$$P_{ne}^{e.m.\to m.} = \left( P_{ne}^{perc}(P_2 N_{1g}, P_2, N_{2g'}, M_{2g'}, \frac{1}{P_2}) \right)^K \tag{36}$$

$$P_{ne}^{m.\to e.m.} = \left( P_{ne}^{perc}(P_2 N_{2g}, P_1, N_1, M_1, \frac{M_{2g}}{N_{2g}}) \right)^K \tag{37}$$

$$P_{ne}^{h.\to e.m.} = \left( P_{ne}^{perc}(P_2 N_{2g}, K, N_h, M_h, \frac{1}{K}) P_{ne}^{perc}(N_{2g'}, K, N_h, M_h, \frac{1}{K}) \right)^K \tag{38}$$

$$P_{ne}^{m.\to h.} = P_{ne}^{perc}(P_h N_{hg}, P_1, N_1, M_1, \frac{M_{hg}}{N_{hg}}) \tag{39}$$

We used these equations to estimate how the number of neurons varies when varying various parameters describing the computational ability of the machine. The top-left panel in SI Fig.8d shows how the numbers of dictionary and gate neurons vary with the number of machine states, it is obtained for $K = 5$ external memory cells $P_2 = 100$ symbols per cell. We optimized only over $N_1, M_1, N_{1g}, M_{1g}$, with fixed parameters $\{NLSC_{e.m.}\}$. The top-right panel shows variations of the number of neurons in the external memory with the number of symbols $P_2$ for $K = 5$ and $P_1 = 1,000$. The bottom-left panel shows variation in numbers of neurons in the head circuit when the number of external memory cells $K$ is varied, for a machine circuit with $P_1 = 1,000$. The bottom-right panel shows variation in the number of neurons in both the machine and external memory as a function of the fraction of transitions $\epsilon_{m.} = \epsilon_{e.m.} = \epsilon$. As previously introduced, $\epsilon_{m.}$ is the fraction of machine transitions associated to each symbol (at most there are $P_1$ transitions per symbol) and $\epsilon_{e.m.}$ is the fraction of external memory transitions associated to each machine state (at most there are $P_2$ transitions per machine state).

In the main text we highlight that splitting the machine into multiple NLSC can be beneficial when the transitions to be programmed exhibit a form of structure. In order to obtain the dashed curves in Fig. 5h, we considered a machine circuit split into two NLSC with $\sqrt{P_1}$ states each, for a total of $P_1$ machine states. To obtain the red curve, corresponding to random transitions, gate neurons implementing transitions in the two machine NLSC and the external memory NLSC are split into $P_2\sqrt{P_1}$ subpopulations, each positively gain modulated upon conjunction of a specific cell symbol and a specific state of the other machine NLSC. $N_{min}$ is obtained by minimizing $2N_1 + KN_2 + \sqrt{P_1}P_2N_{1g} + K\sqrt{P_1}P_2N_{2g}$ under the constraint $P_{ne}^{TM} > 0.9$ replacing equations **(33)**, **(34)** of **(32)** with

$$P_{ne}^{e.m.} = \left( P_{ne}^{ANN}(P_2, N_2, M_2) \left( P_{ne}^{perc}(P_2 N_{2g}, P_2, N_2, M_2, \frac{1}{P_2}) P_{ne}^{perc}(P_2 N_2, P_2, N_{2g}, M_{2g}, \frac{M_2}{N_2} \right)^{\sqrt{P_1}} \times P_{ne}^{perc}(N_{2g'}, P_2, N_2, M_2, \frac{M_{2g'}}{N_{2g'}}) \right)^K \tag{40}$$

$$P_{ne}^{m.} = \left( P_{ne}^{ANN}(\sqrt{P_1}, N_1, M_1) \right)^2 \times \left( P_{ne}^{perc}(P_2 N_{1g}, \sqrt{P_1}, N_1, M_1, \frac{M_{1g}}{N_{1g}}) P_{ne}^{perc}(P_2 N_1, \sqrt{P_1}, N_{1g}, M_{1g}, \frac{M_1}{N_1}) \right)^{2\sqrt{P_1}} \tag{41}$$

$$P_{ne}^{e.m.\to m.} = \left( P_{ne}^{perc}(P_2 N_{1g}, P_2, N_{2g'}, M_{2g'}, \frac{1}{P_2}) \right)^{2\sqrt{P_1}K} \tag{42}$$

$$P_{ne}^{m.\to e.m.} = \left( P_{ne}^{perc}(P_2 N_{2g}, \sqrt{P_1}, N_1, M_1, \frac{M_{2g}}{N_{2g}}) \right)^{2\sqrt{P_1}K} \tag{43}$$

In order to obtain the dotted-dashed curve in Fig. 5h, we considered a special structure for transitions in machine and external memory transitions such that it makes sense to use a compositional representation for them. In our example, half of the cell symbols are associated with transitions of a subset of $P_{1/2}$ machine states, and the other half of the cell symbols are associated with transitions of the complementary subset of $P_{1/2}$ machine states. Symbol transitions also follow the same structure, with transitions from the first half of cell symbols depending only on the state of one of the machine NLSC and transitions from the second half of cell symbols depending only on the state of the other machine NLSC. Formally, we have $P_1$ machine states given by couples $Q(t) = (Q^1(t), Q^2(t))$ with $Q^1(t) \in \mathcal{T}^1$,$Q^2(t) \in \mathcal{T}^2$ with $|\mathcal{T}^1| = |\mathcal{T}^2| = \sqrt{P_1}$. $P_2$ symbols taken from the alphabet $\mathcal{NT} = \mathcal{NT}_1 \cup \mathcal{NT}_2$ with $|\mathcal{NT}_1| = |\mathcal{NT}_2| = P_2/2$. For a symbol $S(t) \in \mathcal{NT}_k$, we have $S(t+1) = F_k(Q^k(t), S(t))$ and $Q^k(t+1) = G_k(Q^k(t), S(t))$ and $Q^{j\neq k}(t+1) = Q^{j\neq k}(t)$.

$$\{S_1, ..., S_{P_2/2}\} \cup \{S_{P_2/2+1} ..., S_{P_2}\} \tag{44}$$

With this constraint on transitions, the number of transition gate neurons in the two machine NLSC and the external memory NLSC can be drastically reduced because we do not need to subdivide each gates into the $\sqrt{P_1}$ subpopulations. $N_{min}$ is obtained by minimizing $2N_1 + KN_2 + 2\frac{P_2}{2}N_{1g} + KP_2N_{2g}$ under the constraint $P_{ne}^{TM} > 0.9$ replacing equations **(33)**, **(34)** of **(32)** with

$$P_{ne}^{e.m.} = \left( P_{ne}^{ANN}(P_2, N_2, M_2) P_{ne}^{perc}(P_2 N_{2g}, P_2, N_2, M_2, \frac{1}{P_2}) P_{ne}^{perc}(P_2 N_2, P_2, N_{2g}, M_{2g}, \frac{M_2}{N_2}) \times P_{ne}^{perc}(N_{2g'}, P_2, N_2, M_2, \frac{M_{2g'}}{N_{2g'}}) \right)^K \tag{45}$$

$$P_{ne}^{m.} = \left( P_{ne}^{archi1}(\{\frac{P_2}{2}, \sqrt{P_1}, N_1, M_1, N_{1g}, M_{1g}, 1\}) \right)^2 \tag{46}$$

$$P_{ne}^{e.m.\to m.} = \left( P_{ne}^{perc}(\frac{P_2}{2} N_{1g}, P_2, N_{2g'}, M_{2g'}, \frac{1}{P_2}) \right)^{2K} \tag{47}$$

$$P_{ne}^{m.\to e.m.} = \left( P_{ne}^{perc}(\frac{P_2}{2} N_{2g}, \sqrt{P_1}, N_1, M_1, \frac{M_{2g}}{N_{2g}}) \right)^{2K} \tag{48}$$

#### 6.1.10 Sequence representations and tree-structures

We consider an environment that produces sequences of length $L$ over an alphabet of $D$ symbols, with $D \gg 1$ ($10,000$ words) and $L = O(1)$ (sentence length). We look for a machine that will transform the $D^L$ possible sequential inputs into $D^L$ stable states (meaning of sequences). This can be done straightforwardly by the FSA pictured in SI Fig.8g, which relies on memory states structured into a tree. The NLSC implementing this FSA has $P = \sum_{k=0}^{L} D^k = O(D^L)$ dictionary patterns (SI Fig.8h), and the minimal number of neurons associated to this type of FSA are expected to scale as $N_{dic} = O(\sqrt{P}) = O(\sqrt{D^L})$ and $N_{gate} = O(D\sqrt{D^L})$ (SI section 6.1.5).

Another possible circuit would buffer the symbols of each sentence into $L$ temporary slots implemented by $L$ dictionaries with $D$ states (SI Fig.8i). This reduces the complexity of the FSA (only $P = D^L$ dictionary states), at the price of increasing the complexity of inputs to the FSA. Although the number of neurons required to implement the FSA dictionary is smaller, in the limit $D \gg 1$ the total number of neurons scales the same way $N_{dic} = O(\sqrt{D^L})$ compared to the previous implementation. To ensure proper categorization gate neurons have to mix inputs from the slots and from the FSA dictionary to point towards the correct category. This job is made more difficult by the fact that the $D^L$ possible inputs to the FSA are represented in the $L$ slots by non-linearly separable neural patterns. Linearly separable representations for the inputs can be performed by $N_{gate} = O(D^L)$. Although we do not prove that this number can not be lowered we still expect it to be larger than $O(D\sqrt{D^L})$, the number of gate neurons used in the neural architecture without memory buffer. This argument could be improved by extending our theory for neural FSA to the case $P_c = O(P)$ (SI section 6.1.5) to compute the minimal number of gate neurons required for this last implementation.

We now consider the case where sequences are structured into chunks: high-level sequences of length $L$ are composed on an alphabet of $D$ chunks, or non-terminal symbols (SI Section 6.8) ; each chunk is associated with a sequence of length $l$ onto $d$ terminal symbols. To categorize these sequences, a naive neural implementation ignoring the chunk structure as the one discussed above will lead to number of neurons scaling as $N_{dic} = O(\sqrt{d^{l\times L}})$ and $N_{gate} = O(d\sqrt{d^{l\times L}})$. Instead of this naive implementation, we can break down the NLSC implementing the naive FSA into two NLSC (SI Fig.8i) and get rid off the $l \times L$ product: $N_{dic} = O(\sqrt{d^l}) + O(\sqrt{D^L})$ and $N_{gate} = O(d\sqrt{d^l})) + O(D\sqrt{D^L}))$.

## 6.2 Rank-restricted networks

We trained rank-restricted networks to perform the tasks of Fig. 1a and Fig. 3a. To do so we initialized the connectivity matrix as $W = \sum_{k=1}^{K} \vec{m}^k \vec{n}^k$ such that recurrently generated activity lies in a subspace of dimension $K$ (98) and minimized the loss by updating the parameters $\{\vec{m}_i^k\}, \{\vec{n}_i^k\}_{i=1,\dots,N}^{k=1,\dots,K}$, where $N = N_{dic} + N_{gate}$ is the total number of neurons. Connectivity vectors are structured into populations (40) using the support vectors $\vec{S}_{Dic}$ and $\vec{S}_{Gates}$ with non-zero entries at 1 on dictionary and gate neurons. For the task with the simple temporal structure of Fig. 1a, we used the following rank $K = 4$ connectivity matrix associated with the four cognitive variables:

$$\begin{aligned}
\vec{n}_A &= \sqrt{\rho}\vec{A} \odot \vec{S}_{DicA} + \vec{I}_A \odot \vec{S}_{Dic} \\
\vec{m}_A &= \sqrt{\rho}\vec{A} \odot \vec{S}_{DicA} + \vec{AB} \odot \vec{S}_{Gate} \\
\vec{n}_B &= \sqrt{\rho}\vec{B} \odot \vec{S}_{DicB} + \vec{AB} \odot \vec{S}_{Gate} \\
\vec{m}_B &= \sqrt{\rho}\vec{B} \odot \vec{S}_{DicB} + \vec{w}_{out} \odot \vec{S}_{Dic}
\end{aligned} \tag{49}$$

and symmetrically

$$\begin{aligned}
\vec{n}_C &= \sqrt{\rho}\vec{C} \odot \vec{S}_{DicC} + \vec{I}_C \odot \vec{S}_{Dic} \\
\vec{m}_C &= \sqrt{\rho}\vec{C} \odot \vec{S}_{DicC} + \vec{CD} \odot \vec{S}_{Gate} \\
\vec{n}_D &= \sqrt{\rho}\vec{D} \odot \vec{S}_{DicD} + \vec{CD} \odot \vec{S}_{Gate} \\
\vec{m}_D &= \sqrt{\rho}\vec{D} \odot \vec{S}_{DicD} + \vec{w}_{out} \odot \vec{S}_{Dic}
\end{aligned} \tag{50}$$

Vectors $\vec{X}, \vec{XY} \in \mathbb{R}^N$ are Gaussian vectors with entries drawn independently from the normalized centered Gaussian distribution. The dictionary population is split into four populations with support vectors $\vec{S}_{DicX}$ having non-zero entries at 1 on dictionary neurons of population X, so has to have four independent fixed points by taking $\rho > 1$ such that $\langle \vec{n}_X, \vec{m}_X \rangle > 1$. $\odot$ is the Hadamard product performing entry-wise multiplication. Vectors $\vec{n}_X$ decide which inputs are picked up by recurrent activity. In order for the two cognitive variables $\kappa_A$ and $\kappa_C$ to activate in response to the two stimuli, we take $\langle \vec{n}_A, \vec{I}_A \rangle, \langle \vec{n}_B, \vec{I}_C \rangle = O(1)$, with $\vec{I}_A$ and $\vec{I}_C$ Gaussian sensory input vectors. Vectors $\vec{m}_X$ decide in which direction recurrent currents are generated. We take $\langle \vec{m}_B, \vec{w}_{out} \rangle, \langle \vec{m}_D, \vec{w}_{out} \rangle = O(1)$ for generating the output, and $\langle \vec{m}_A, \vec{n}_B \rangle, \langle \vec{m}_C, \vec{n}_D \rangle = O(1)$ to transfer activations of $\kappa_A$ or $\kappa_C$ into activation of $\kappa_B$ or $\kappa_D$ upon presentation of the go cue, which changes the gain of gate neurons from 0 to 1 through the input $(1 - u_{go}(t))\sigma_{SAT}\vec{X} \odot \vec{S}_{gate}$ with $\sigma_{SAT} \gg 1$.

For the task with long-distance dependencies of Fig. 3a, we used a rank $K = 7$ matrix generating five dictionary states $A, A', B, C, C'$ and the two states of the external memory $hA, hA'$:

$$
\begin{aligned}
\vec{n}_A &= \sqrt{\rho}\vec{A} \odot \vec{S}_{DicA} + \vec{I}_A \odot \vec{S}_{Dic} \\
\vec{m}_A &= \sqrt{\rho}\vec{A} \odot \vec{S}_{DicA} + \vec{AB} \odot \vec{S}_{Gate} + \vec{w}_{out} \odot \vec{S}_{Dic} \\
\vec{n}'_A &= \sqrt{\rho}\vec{A'} \odot \vec{S}_{DicA'} + \vec{I}_{A'} \odot \vec{S}_{Dic} \\
\vec{m}'_A &= \sqrt{\rho}\vec{A'} \odot \vec{S}_{DicA'} + A \odot \vec{S}_{Gate} + \vec{w}_{out} \odot \vec{S}_{Dic} \\
\vec{n}_B &= \sqrt{\rho}\vec{B} \odot \vec{S}_{DicB} + \vec{AB} \odot \vec{S}_{Gate1} + \vec{A'B} \odot \vec{S}_{Gate1} \\
\vec{m}_B &= \sqrt{\rho}\vec{B} \odot \vec{S}_{DicB} + \vec{BC} \odot \vec{S}_{GateBC} + \vec{BC'} \odot \vec{S}_{GateBC'} + \vec{w}_{out} \odot \vec{S}_{Dic} \\
\vec{n}_C &= \sqrt{\rho}\vec{C} \odot \vec{S}_{DicC} + \vec{BC} \odot \vec{S}_{GateBC} \\
\vec{m}_C &= \sqrt{\rho}\vec{C} \odot \vec{S}_{DicC} + \vec{w}_{out} \odot \vec{S}_{Dic} \\
\vec{n}_{C'} &= \sqrt{\rho}\vec{C'} \odot \vec{S}_{DicC'} + \vec{BC'} \odot \vec{S}_{GateBC'} \\
\vec{m}_{C'} &= \sqrt{\rho}\vec{C'} \odot \vec{S}_{DicC'} + \vec{w}_{out} \odot \vec{S}_{Dic}
\end{aligned} \tag{51}
$$

and

$$
\begin{aligned}
\vec{n}_{hA} &= \sqrt{\rho}h\vec{A} \odot \vec{S}_{hA} + \vec{I}_A \odot \vec{S}_{Dic} \\
\vec{m}_{hA} &= \sqrt{\rho}h\vec{A} \odot \vec{S}_{hA} + \sigma_{SAT}\vec{X} \odot \vec{S}_{GateBC'} \\
\vec{n}_{hA'} &= \sqrt{\rho}h\vec{A'} \odot \vec{S}_{hA'} + \vec{I}_{A'} \odot \vec{S}_{Dic} \\
\vec{m}_{hA'} &= \sqrt{\rho}h\vec{A'} \odot \vec{S}_{hA'} + \sigma_{SAT}\vec{X} \odot \vec{S}_{GateBC}
\end{aligned} \tag{52}
$$

which uses the same principles than those used for the first task, with the addition of saturating currents ($\sigma_{SAT} \gg 1$) generated by $\vec{m}_{hA}$ and $\vec{m}_{hA'}$ on the two gate populations supported by $\vec{S}_{GateBC'}$ and $\vec{S}_{GateBC}$, to control transitions from $B$ to $C$ and from $B$ to $C'$.

Equations over cognitive variables $\kappa_k = \vec{m}_k^T \vec{x}$ are obtained by projecting the dynamical equation **(1)** onto directions $\vec{m}_k$ leading to equations **(4)**,**(5)** (40) with

$$
\tilde{\sigma}_{XY} = \sum_{p=1}^{P} \sigma_{XY}^{(p)} \langle \phi' \rangle_p
$$

$$
\sigma_{XY}^{(p)} = \langle \vec{m}_X, \vec{n}_Y \rangle_p
$$

$$
\langle \phi' \rangle_p = \int Dz \phi'(\Delta^{(p)} z)
$$

$$
\Delta^{(p)} = \sqrt{\sum_{k=1}^{K} (\sigma_k^{(p)})^2 \kappa_k^2 + (\sigma_{I_{go}}^{(p)})^2 (1 - u_{go})^2}
$$

$$
\sigma_k^{(p)} = \sqrt{\langle \vec{m}_k, \vec{m}_k \rangle_p}
$$

$$\sigma_{I_{go}}^{(p)} = \sqrt{\langle \vec{I}_{go}, \vec{I}_{go} \rangle_p} \tag{53}$$

where the scalar products $\langle ., . \rangle_p$ are taken over neurons belonging to population $p$.

## 6.3 Pseudo-code for foraging task

```
# initializations
nb_paths = 3
P = 10

# each of the 3 paths is defined as a vector of P subsequent non-
   repeating positions
paths = permutations[nb_paths,P]

# vector of size P with entries drawn as 0 (no food) or 1 (food) to
   initialize reward map
reward_map = random[0,1,P]

start_position = 1
hunger_level = 3

# variable holding the actual position of the animal
current_location = start_position

# variable holding the virtual position of the animal
virtual_location = start_position

while hunger_level > 0:

    # planing step:
    for i in range(nb_paths):
        # variable holding the distance between the actual position of
           the animal and the next available food location on path i.
        count_first_encounter[i] = 10
        reward_found = 0
        for j in range(P):
            virtual_location = paths[i](virtual_location)
            reward_at_virtual_location = reward_map[virtual_location]
            if (reward_at_virtual_location == 1) and (reward_found ==
               0):
                count_first_encounter[i] = j
                reward_found = 1

    # decision step
```

```
chosen_path = argmin(count_first_encounter[:])
nb_steps = min(count_first_encounter[:])

# traveling step
for j in range(nb_steps):
    current_location = paths[chosen_path](current_location)

# food consumption step
reward_map[current_location] = 0
virtual_location = current_location
hunger_level = hunger_level - 1
```

## 6.4 Assembling NLSC for the foraging task

The head circuit is a NLSC with a 10-states dictionary each encoding one of the 10 positions. This dictionary is associated with three gate populations, each encoding a permutation, or a path, across the 10 locations. Wiring of the head circuit reflects information encoding in the parameters contained in **paths** initialized at the beginning of the pseudo-code. Updates of the variables **current_location** and **vritual_location** are implemented by dictionary state transitions in the head NLSC.

The external memory implements the **reward_map** variable. It is composed of 10 NLSC, one per location (SI Fig. 8e, green square). Each is composed of a 2-states dictionary encoding the absence or presence of reward at the corresponding location, together with a single gate population activated to update reward value, e.g. when food is consumed at this location. As explained in Method 5.2.4 the connectivity from the dictionary's head circuit to the gates of the external memory are first initialized: a cell-assembly encoding location $X$ in the dictionary inhibits all 10 gate populations of the external memory, but the one associated to the NLSC encoding value at position X. This connectivity is then fine-tuned by training the assembled network to update, upon activation of an external go cue, the value of the external memory NLSC-X only when the head circuit is in state X. In addition to these *transition* gates bi-directionaly coupled with the dictionary and implementing transitions between dictionary states, these NLSC have *output* gates that only receive from the dictionary and that are modulated by the head circuit to decide from which of these 10 NLSC the machine is focusing its attention.

What we considered here as the machine operating on the external memory through the head circuit is composed of multiple NLSC with different functions (SI Fig. 8e, blue square). We first present NLSC that implement specific operations, and then NLSC that coordinate these operations. To each path is associated a counter memorizing the distance from food on this path (**count_first_encounter** variable). Each counter is a NLSC with a 10-states dictionary encoding distances from 1 to 10 and a single gate population implementing a single permutation allowing to increment the counter. As for the external memory, these NLSC have *output* gates that provide, after the planing steps, the number of locations to go through to reach reward.

These transitions in the counters only occur when a binary memory slot signals that no reward

has yet been encountered during planing a path. This memory, carried by the **reward_found** variable in the pseudo-code, is implemented by a NLSC with a 2-states dictionary, encoding whether or not a reward has been encountered (Fig. 5g). A single gate population activates to trigger transitions between these two states, with e.g. an update when a location with reward is encountered (Fig. 5g).

When the three paths have been planed, a decision circuit reads from the 3 counters to select which path will actually be taken. This decision circuit is a NLSC with a dictionary state composed of a neutral state and 3 states each encoding a potential chosen path. A gate neuron population receiving inputs from the 3 NLSC counters has its connectivity with the dictionary trained to perform the *argmax* operation, transiting from the neutral state to the state associated to the path $P^*$ with the smallest counter value, updating the variable **chosen_path**. Once such a state is reached, it releases inhibition on the gate population corresponding to the chosen path in the head NLSC such that the agent travels on path $P^*$ through $K_{P^*}$ locations.

Reaching reward, food consumption is modeled by switching to an eat-state in the 2-states dictionary of a NLSC. Upon this switch, inhibition is released on the gate population of another NLSC. This hunger level NLSC has a 5-states dictionary each associated with a possible hunger-level of the agent (**hunger_level** variable). The dictionary state associated with hunger level 0 inhibits all gate populations in the neural architecture, terminating the **while** loop in the pseudo-code. The gate population triggers transition towards a state encoding the next lower hunger levels, corresponding to the line of the pseudo-code where the **hunger_level** variable is updated.

The **inner for-loop** in the planing step of the program (SI Section 6.3) is implemented by a NLSC with a 10-states dictionary. This dictionary is associated with a gate population implementing a single permutation among these 10 states. Before planing each path, the dictionary of this NLSC is in state $1$, the gate population activates each time the head circuit is updated such that after going through a path, it ends up in state $10$. The cell-assembly associated to this final state informs, through its connectivity to a gate population, a coordinating NLSC to plan on another path or to move on to the decision step. This coordinating NLSC is composed of a 5-states dictionary. 3 states are associated with planing the 3 paths (**outer for-loop**), one state is associated with the decision and traveling steps (when the corresponding cell-assembly is activated it allows for transitions in the decision making NLSC and enables movements of the agent) and one state is associated with the food consumption step (when the corresponding cell-assembly is activated it allows for transitions in the eat, reward map and hunger-level NLSC).

## 6.5 Pseudo-code for sensory-motor hierarchy

### 6.5.1 Dress the table

```
# Dress subroutine
dress():

    number_of_location = 10
    pile_size = 6
```

```
    # representation of agent's current location
    current_location = 0

    # model of the environment, sensed by sensory system: all locations
        initially empty except location X0
    location_occupancy = [pile_size, 0, ..., 0]

    # internal representation of the state of the motor apparatus
    grapple_state = released # 0 for \textit{Released}, 1 for \textit{
       Grasp}

    while grapple_state is not Stop:
        if current_location == 0:
            if pile_size == 0:
                grapple_state = Stop
            else:
                grapple_state = grasp
                pile_size = pile_size - 1
                current_location = current_location + 1
        elif current_location > 0:
            if grapple_state == grasp:
                if location_occupancy[current_location] == 1:
                    current_location = current_location + 1
                elif location_occupancy[current_location] == 0:
                    grapple_state = released
                    location_occupancy[current_location] = 1
                    current_location = current_location - 1
            if grapple_state == released:
                current_location = current_location - 1
```

### 6.5.2 Serve and consume food

```
# Serve and Eat subroutine
se():

    # current_location in {Plate,Dish}
    current_location = plate
    plate_level = 0
    dish_level = 15
    hunger_level = 8

    # motor_state in {Released, Grasp, Pick, To Mouth, Drop}
    motor_state = Released
```

```
    # meta_state  in {Serve, Eat, Stop}
    meta_state = Serve

    while meta_state != Stop:
        while meta_state == Serve:
            do serve()
        while meta_state == Eat:
            do eat()

# Serve subroutine

serve(meta_state, motor_state, current_location, plate_level,
   dish_level):
    if current_location == Plate AND motor_state == Released:
        current_location = Dish
    elif current_location == Dish AND motor_state == Released AND
       dish_level > 0:
        motor_state = Grasp
    elif current_location == Dish AND motor_state == Grasp:
        motor_state = Pick
        dish_level = dish_level - 1
    elif current_location == Dish AND motor_state == Pick:
        motor_action = Grasp
        current_location = Plate
    elif current_location == Plate AND motor_state == Grasp:
        motor_action = Drop
        plate_level = plate_level + 3
    elif current_location == Plate AND motor_state == Drop:
        motor_action = Grasp
        meta_state = Eat
    elif current_location == Dish AND motor_state == Released AND
       dish_level == 0:
        meta_state = Stop

    return meta_state, motor_state, current_location, plate_level,
       dish_level

# Eat subroutine

eat(meta_state, motor_state, plate_level, hunger_level):
    if motor_state == Grasp AND plate_level > 0 AND hunger_level > 0:
        motor_state = Pick
        plate_level = plate_level - 1
    elif motor_state == Pick:
        motor_state = To Mouth
```

```
        hunger_level = hunger_level - 1
    elif motor_state == To Mouth:
        motor_state = Grasp
    elif motor_state == Grasp AND (plate_level == 0 OR hunger_level ==
       0):
        motor_action = Released
        current_location = Plate
        if hunger_level == 0:
            meta_state = Stop
        elif hunger_level > 0:
            meta_state = Serve

    return meta_state, motor_state, plate_level, hunger_level
```

### 6.5.3 Bring new food

```
# Bring Food subroutine:
bf(plate_level):
    if meta_state == Wait:
        plate_level = plate_level + 5
        meta_state = Stop
    return plate_level
```

### 6.5.4 Undress the table

```
# Undress subroutine
undress():

    number_of_location = 10
    pile_size = 0

    # representation of agent's current location
    current_location = 0

    # model of the environment, sensed by sensory system: some
       locations (associated with value 1) are initially occupied by a
       plate
    location_occupancy = [1, 1, ..., 0]

    # internal representation of the state of the motor apparatus
    grapple_state = released

    while grapple_state != Stop:

        if grapple_state == Released:
            if location_occupancy[current_location] == 0:
```

```
                current_location = current_location + 1

                # current_location = 9+1 = 0
                if current_location == 0:
                    grapple_state = Stop

            elif location_occupancy[current_location] == 1:
                grapple_state = grasp
                location_occupancy[current_location] = 0
                current_location = current_location - 1

        elif grapple_state == Grasp:
            if current_location == 0:
                grapple_state = Released
                location_occupancy[0] = location_occupancy[0] + 1
            elif current_location > 0:
                current_location = current_location - 1
```

### 6.5.5 Coordination of subroutines

```
# Dinner subroutine:
Dinner()

hunger_level = 8
plate_level = 0
dish_level = 3

# meta_state in {Dress, Undress, EatServe, BF}
meta_state = Dress

# cell_symbol in {Stop, Released, Grasp, Serve, Eat, Wait}
# Dress and Undress initial state
cell_symbols[1] = Released
# Serve and eat initial state
cell_symbols[2] = Serve
# Bring food initial state
cell_symbols[3] = Wait

current_cell_symbol = cell_symbols[1]

# go out of the loop when hunger level at 0 and undress() sets the
   current_cell_symbol to Stop
while hunger_level > 0 AND current_cell_symbol != Released:
    if current_cell_symbol != Stop:
        if meta_state == Dress:
```

```
            cell_symbols[1] = dress(current_cell_symbol)
            current_cell_symbol = cell_symbols[1]
        elif meta_state == EatServe:
            cell_symbols[2] = se(current_cell_symbol)
            current_cell_symbol = cell_symbols[2]
        elif meta_state == Undress:
            cell_symbols[1] = undress(current_cell_symbol)
            current_cell_symbol = cell_symbols[1]
        elif meta_state == BF:
            cell_symbols[3] = bf(current_cell_symbol)
            current_cell_symbol = cell_symbols[3]
    elif current_cell_symbol == Stop:
        if meta_state == Dress:
            meta_state = EatServe
            cell_symbols[1] = Released
            current_cell_symbol = cell_symbols[2]
        elif meta_state == EatServe:
            if hunger_level == 0:
                meta_state = Undress
            elif hunger_level > 0:
                meta_state = BF
                cell_symbols[2] = Serve
                current_cell_symbol = cell_symbols[3]
        elif meta_state == BF:
            cell_symbols[3] = Wait
            current_cell_symbol = cell_symbols[2]
```

## 6.6 Assembling NLSC for the sensory-motor hierarchy

**Dress the table**

With this subroutine (SI Section 6.5.1) the agent operates on a table of $P = 10$ locations with a grapple that can be in one of two states, *Grasp* or *Released*, to handle plates. The agent senses each location to report whether the location is empty or occupied by a plate, and it travels clock-wise or counter-clockwise across locations on the table. The grapple grasps plate from a pile at an initial location, travels until finding an empty location, drops the plate and returns to the pile.

Note that compare to the previous task, where the reward map models an internal model of the environment (i.e. the external memory is implemented as a neural network), the 'location occupancy' here models the environment. The agent senses it through a visual system, and acts upon it through a motor apparatus.

We first built a NLSC which can be thought of as a head circuit deciding the location of the environment on which the agent senses and acts upon. It has a 10-states dictionary, one state per location, and 2 populations of gate neurons, instructing clockwise and counter-clockwise moves on the environment.

The machine running the subroutine consists of a NLSC with a 3-states dictionary. The *Grasp* and *Released* states are internal models of the motor configuration of the grapple, and are directly linked to corresponding motor neurons. A third *Stop* state is activated when the subroutine is complete. This NLSC has a gate population composed of 2 sub-populations, one programming transitions while the agent is at location 0, the other for other locations. Gate neurons receive sensory inputs informing the agent whether there is a plate or not at the current location.

**Serve and consume food**
Here the agents can operate either on its plate to eat, or on a dish to serve food in the plate. It controls a motor apparatus able to handle a fork and a spoon, its sensory system monitors the amount of food in the plate or the dish. Moves between plate and dish, and actions performed by the agent are coordinated according to the subroutine described by its pseudo-code in SI Section 6.5.2, it calls two independent lower-level routines operating on the same motor apparatus but sensing either the state of the plate or the dish.

In order to implement this subroutine, we built a NLSC with a 5-states dictionary encoding the configurations of the motor apparatus, a NLSC with a 2-states dictionary encoding the location of the agent, i.e. whether it operates on the plate or the dish, a NLSC with a 10-states dictionary encoding hunger levels and a NLSC with a 3-states dictionary encoding meta states for temporally coordinate the sub-routine. These NLSC are wired with connections from cell-assemblies in dictionaries to gate neurons to implement the state transitions listed in the pseudo-codes. The neural architecture is sensing the environment through sensory neurons reporting plate and dish levels, and acting upon it through motor neurons updating plate and dish levels.

**Bring new food** When the dish level is at 0 while the hunger level is not, this subroutine is called to put new food in the dish. We do not modeled it in details and assume it corresponds to the agent resorting to another set of sensory and motor skills on a new part of the overall environment (e.g. operating a fridge).This is minimally modeled by a NLSC with a 2-states dictionary encoding the *wait* and *stop* states.

**Undress the table** With this subroutine the agent operates on the table with the grapple and has the same moving ability as in the **Dress the table** subroutine. The agent, with its grapple, collects one-by-one the plates around the table and brings them at location $X_0$.
The associated neural architecture is the one used for the subroutine **Dress the table**, we simply add a new set of gate neurons to the machine NLSC with the *Grasp*, *Released* and *Stop* states. This new set of gates allows to program the machine states transitions, conditioned by sensory inputs and agent's location, corresponding to this subroutine.

**Coordination of subroutines** Each of the subroutine described above is implemented by a neural Turing machine composed of a head circuit and machine circuit operating on an external memory constituted by the environment (either the table for **Dress** and **Undress**, either the plate and the dish for **Serve and eat**, either the fridge for **Bring food** ). The neural Turing machine that implements the following pseudo-code to coordinate the subroutines is described by the pseudo-code (SI Section 6.5.5).

The 3-cells external memory is composed of the three NLSC with stop states that we have described in the above subroutines. The head circuit is composed of a NLSC with a 4-states dictionary encoding the *Dress*, *Undress*, *EatServe* and *BF* meta states. When in one of this state, the machine attends to one of the cell of the external memory and drive transition in the cell corresponding to performing the associated subroutine. Transitions in this 4-states NLSC occur when the currently read symbol is the Stop symbol. Here we consider that the hunger level NLSC is constituting the machine such that when the 0 hunger level state is reached in this network it can instruct an higher-order program to switch to another activity.

## 6.7 Automata

In order to support the claims made throughout the main text, we recap on what is presented in (30) about automata and computation.

### 6.7.1 Finite-state automata

A finite-state automaton (FSA) or finite-state machine is composed of a finite number of states $Q \in \{Q_1, ...., Q_P\}$, at each time step it can receive a stimulus $S \in \{S_1, ..., S_K\}$, or symbol, from its environment. In response to that stimulus it updates its state $Q(t)$ to $Q(t+1)$ and might produce a response $R(t+1)$:

$$Q(t+1) = G(Q(t), S(t)) \tag{54}$$
$$R(t+1) = F(Q(t), S(t)) \tag{55}$$

Below we give specific examples of FSA to illustrate the work exposed here.

FSA with $P$ states can be used to build a head-direction system with each state $Q_k$ representing an angular position $\frac{k}{P}360°$ (Fig. 2b). At any time, the environment of this system (e.g. the vestibular part of the nervous system of the animal) provides one of two stimuli $S \in \{-1, +1\}$, indicating if the animal's body has rotated counter-clock wise or rotated clock wise. Such automaton can be referred to as a categorizer in the sense that it categorizes all the possible rotation sequences (e.g. $+1, -1$ or $+1, +1, +1, -1, +1, -1$) into one of $P$ classes each representing a current head-orientation with respect to some arbitrary landmark. Note here that computation is paced not by a clock but by the arrival of environmental inputs onto the system.

A peculiar class of FSA is the class of autonomous FSA, or transducers (Fig.2a). They take only a single symbol as input, instructing the automaton to move on to its next state. A common example is the one of a traffic light that switches states each time a timer signal is ON, with each state associated to the production of a color. Another example that we encountered in Fig. 1g is a NLSC with $P$ dictionary states that can autonomously produce a sequence of $P$ motor actions.

Throughout figures of the main text, we have used two different graphical representations for FSA. In e.g. Fig. 2g, we represented each state by an hexagon together with labeled arrows telling

which input would trigger which transition between states. In e.g. Fig. 3g we used a more compact representation in which the entire FSA is represented by a box, the front-side of which displays the current state it is in.

### 6.7.2 Turing machines

Modeling the operations of machines with FSA can be considered incomplete in the sense that reactions of the environment $S(t+1)$ in response to the actions of the machine $R(t)$ are left unspecified. In order to model and define the notion of computation, Turing proposed a minimal model in which FSA are augmented with a head that can read, write and travel on an external memory modeling the environment (Fig. 5g). The tape is split into $K$ cells each displaying at time $t$ a symbol $S^k(t) \in \{S_1, ..., S_K\}$. If at time $t$ the head reads cell $k_0(t)$ of the external memory, it leads the machine to i) update its state according to

$$Q(t+1) = G(Q(t), S^{k_0}(t)) \tag{56}$$

to ii) act upon the external memory in a predictable manner by updating the current cell's state's according to a well defined function $F(.,.)$

$$S_0^k(t+1) = F(Q(t), S^{k_0}(t)) \tag{57}$$

where F and G are independent on $k_0$, as well as to iii) move the head to another location depending on what has been read

$$k_0(t+1) = D(Q(t), S^{k_0(t)}(t)) \tag{58}$$

where $D(Q(t), S^{k_0}(t)) = k_0(t) \pm 1$ such that the head moves immediately either left or right from the current cell being read. Turing machines are automata composed of such FSA, head and tape.

In our neural instantiation of Turing machines, machine states and tape symbols are carried by cell-assemblies in dictionaries while transition tables represented by the functions $G(.,.), F(.,.)$ and $D(.,.)$ are instantiated by linearly separable representations in gate neurons and specific wiring between dictionary and gate neurons.

### 6.7.3 Other automata

While FSA alone are limited in the procedures they can implement, it is commonly postulated (Church-Turing thesis) that any procedure that can be well-defined can be implemented by a Turing machine, provided that the tape can grow an infinite number of cells. This is done by specifying the alphabets of the machine ($Q's$) and tape-cells ($S's$), as well as the transition functions $G(.,.), F(.,.)$ and $D(.,.)$. Turing machines can be extended with e.g. multiple tapes or in other way, but the obtained automata do not gain in computational power. Otherwise, other automata can be defined as restrictions from Turing machines. Different classes of automata maps onto levels of the Chomsky's hierarchy. For instance, linear-bounded automata are Turing machines with a non-infinite tape. This class of automata has been shown to contain acceptors for languages generated by context-sensitive grammars. Pushed-down automata can be seen as Turing machine

with a single-cell tape with restrictions on the way symbols are updated on this cell. This class of automata has been shown to contain acceptors for languages produced from context-free grammars. Finite-state automata are Turing machines without tape, and this class contains acceptors for languages produced from regular grammars.

## 6.8 Formal languages and generative grammar

Formal languages, such as those defined by generative grammar, are sets of words obtained by forming strings of symbols taken from an alphabet of size $P$. Grammatical or production rules used to compose strings are written using the *terminal* symbols of the alphabet $\mathcal{T}$ and another alphabet of *non-terminal* symbols $\mathcal{NT}$. An example is the following set of rules $\{R_k\}$ over $\mathcal{T} = \{a, b, ..., z\}$ and $\mathcal{NT} = \{Start, Stop, A, B, ..., Z\}$

$$R_{1,...,26} : Start \rightarrow \begin{cases} A \\ B \\ ... \\ Z \end{cases} \tag{59}$$

$$R_{27,28} : A \rightarrow \begin{cases} aB \\ aStop \end{cases} \tag{60}$$

$$R_{29,30} : B \rightarrow \begin{cases} bC \\ bStop \end{cases} \tag{61}$$

$$...$$

$$R_{77,78} : Z \rightarrow \begin{cases} zA \\ zStop \end{cases} \tag{62}$$

these alphabets and production rules in finite number define the language L as an infinite subset of the set of all the possible strings that could be composed from the alphabet $\mathcal{T}$, namely the subset of strings that are arranged in alphabetical order (Fig. 1g). Every string in L can be printed by application of the rules, e.g. string $abc$ is obtained by application of $R_1$, $R_{27}$, $R_{29}$ and $R_{32}$.

The above language is particularly simple owing to the following restrictions on the form of its rules: on both the left and right hand-side of the arrows, only a single non-terminal symbol is allowed. This restriction defines regular languages, the lowest level of complexity of Chomsky's hierarchy. Allowing multiple non-terminal symbols on the right hand-side of production rules defines context-free languages, the second complexity level of Chomsky's hierarchy. Strings produced from applications of such grammatical rules can exhibit long-distance dependencies, as in the following example generating the finite set of sequences of Fig. 3a with an alphabet of terminal symbols $\mathcal{T} = \{a, a', b, c, c'\}$ and non-terminal symbols $\mathcal{NT} = \{Start, Stop, S1, S2, B, C\}$

$$R_{1,2} : Start \rightarrow \begin{cases} aBS1 \\ a'BS2 \end{cases} \tag{63}$$

$$R_3 : B \rightarrow b \tag{64}$$

$$R_{4/5} : S1/S2 \rightarrow c/c' \tag{65}$$

for a finite set of sequences it is always possible to propose rules with regular-language restrictions (e.g. $S \rightarrow abc/a'bc'$). By contrast, in order to generate the infinite set of sequences with long-distance dependencies that we considered in Fig. 3g, a compact description of all the possible sequences is given by context-sensitive production rules. Context-sensitive production rules can have multiple symbols on the left-hand side such as

$$R : A\beta \rightarrow \gamma\beta \tag{66}$$

where $\beta$ and $\gamma$ are strings of terminal and non-terminal symbols, with $\gamma$ non-empty. The symbol A is thus replaced depending on its surrounding context $\beta$. For our example, a language with all alphabetical ordered sequences starting from odd letters (A, C, ...) and counter-alphabetical ordered sequences starting from even letters (B,D,...), the production rules are the following

$$R_{1,...,13} : Start \rightarrow \begin{cases} AS_1 \\ CS_1 \\ ... \end{cases} \tag{67}$$

$$R_{13,...,26} : Start \rightarrow \begin{cases} BS_2 \\ DS_2 \\ ... \end{cases} \tag{68}$$

$$R_{27,...,78} : XS_1 \rightarrow \begin{cases} x(X+1)S_1 \\ xStop \end{cases} \tag{69}$$

$$R_{79,...,130} : XS_2 \rightarrow \begin{cases} x(X-1)S_2 \\ xStop \end{cases} \tag{70}$$

with alphabets $\mathcal{T} = \{a, b, ..., z\}$ and $\mathcal{NT} = \{Start, Stop, A, B, ..., Z, S_1, S_2\}$. These rules can easily be generalized to generate sequences organized hierarchically (Fig. 4a), by introducing non-terminal symbols for chunks at multiple levels. By contrast, writing generative grammar rules that produce sequences with an algebraic pattern structure (24) such as XYX (Fig. 4f) is cumbersome, owing to the requirement that the two X's should lead to printing the same symbol. Possible context-sensitive rules are the following:

$$R_1 : Start \rightarrow XYX \tag{71}$$

$$R_{2,...,27} : X \rightarrow \begin{cases} a \\ b \\ ... \end{cases} \tag{72}$$

when the context on the left of X is empty, while if this context is non-empty X yields

$$R_{28,\ldots,53} : \alpha Y X \rightarrow \alpha Y \alpha \tag{73}$$

for $\alpha \in \mathcal{T}$. And,

$$R_{54,\ldots,79} : Y \rightarrow \begin{cases} a \\ b \\ \ldots \end{cases} \tag{74}$$

We needed to introduce many particular rules for the expression of the last X ($R_{28,\ldots,53}$), making the generative grammar formalism not well suited to compactly capture this type of temporal structures.

Context-sensitive rules have the restriction that the symbol $\gamma$ in **(66)** has to be non-empty. If this restriction is lifted, one gets unrestricted rules that can generate recursively enumerable languages. The different types of grammatical rules that we have successively introduced (regular, context-free, context-sensitive and unrestricted) form the so-called Chomsky's hierarchy, with a type of rule at one level being a particular case of the more general type at the next level. The different levels of the Chomsky's hierarchy can be mapped onto automata in the following sense: for every language L produced by type-k grammatical rules, one can build a type-k automaton that recognizes whether or not a given string of terminal symbol is part of L.

## 6.9 Relationship with natural language and tree structures

We find that the primitive temporal structures outlined in (24) maps more naturally onto properties of neural architectures compared to the levels of complexity defined by Chomsky's hierarchy. For instance when going from regular to context-free languages, the associated automata are FSA and push-down automata. Pushed-down automata are composed of a FSA reading on a single-cell external memory, as is the case in the neural architecture proposed for sequences organized into chunks. However the functioning of this external memory, which works as a stack of symbols where the top symbol can be popped or replaced by a new one, is more difficult to map onto neural network operations compared to the single cell-external memory composed of a NLSC. This adds to previous works questioning the relevance of this definition of temporal structures for the study of cognition (11).

A feature that has been outlined regarding natural language and other forms of high-level cognitive processes is its combinatorial nature: from a finite number of words one can built a combinatorially large number of sentences, e.g. to describe situations that have never been encountered. The sensory-motor programs point of view developed here proposes a solution: a program is implemented with a finite number of external memory and machine symbols, a finite number of programmed transitions, but its external memory can be loaded with a combinatorially large number of different inputs which triggers a combinatorially large number of meaningful neural trajectories. This view contrasts with previous attempts at trying to build attractor neural networks with a number of fixed points scaling exponentially with network size (99; 100; 101).

Beyond sequences organized into chunks or algebraic patterns, cognitive science studies outlined that high-level cognitive processes such as natural language relies on tree structures (24). Following physical constraints on neural networks, we found it more natural to jump from neural architectures for sequences organized into chunks or algebraic patterns, to neural Turing machines associated with the running of sensory-motor programs. In our framework, tree-like mental constructions appear when trying to build a circuit that receives strings of words and maps those into stable meaning representations (SI Section 6.1.10). Building such meaning representations requires to use intermediary neural representations that evolve on a tree structure (SI Fig.8g). Tree-structures can be repeated at multiple levels of abstractions if sequences to be mapped are organized into chunks (SI Fig.8i). Within this setup, we compared a scenario where each new incoming symbol is merged with the current neural representation (SI Fig.8g right), with a different scenario where incoming symbols are buffered before being merged (SI Fig.8h). We found the first scenario to be more economical in terms of number of neurons (SI Section 6.1.10). We wonder whether this physical constraint on neural architectures could be related to the MERGE operation that has been postulated to be the fundamental mental operation underlying natural language (24).

## 6.10 Supplementary discussion

### 6.10.1 Relationship with previous theoretical works

**Statistical physics of neural networks**. The elementary circuit is composed of a dictionary in which cell-assemblies can be activated thanks to positive feedback loops in its connectivity. The dictionary is augmented with gates allowing to trigger transitions between cell-assemblies. We have used tools from statistical physics to characterize the NLSC. From a dynamic-mean-field point of view (102; 40), cell-assemblies are associated with cognitive variables, the gain of populations of gate neurons regulate functional interactions between cognitive variables (SI Fig.1) (40). We have leveraged storage capacity calculation techniques originally applied to perceptrons (91; 92), Hopfield-like models (44; 43; 102; 94) and attractor neural networks in general (103; 104; 105), to provide a quantitative characterization of NLSC with minimal number of neurons. This minimal number of neurons is obtained when minimizing interferences between auto-associative loops of the dictionary and the hetero-associative connectivities with gate neurons (Fig.1h,i, SI Section 6.1). This is done by tuning synaptic connectivity probabilities, which are controlled by the coding levels of neural patterns under the Hebbian learning scenario considered here. As for previous works (43; 95), we found that the maximal number of stable states and permutations among states are reached for small coding levels, for which Hebbian-type plasticity leads to an optimal use of neural ressources (106).

**Sequences in neural networks**. Focusing on temporal structures, our work departs from a previous work interested in simple transitions between attractor states (34) or from storage capacity studies of networks implementing unstructured sequences either with direct hetero-associative connectivities (97; 33; 107; 108; 109) or gated-hetero-associative connectivity as used here (41). Previous works proposed networks producing or recognizing sequences with hierarchical temporal structures (110; 111). Gate neurons similar to the one we introduced have also been proposed to

model sub-cortical circuits and to confer flexibility in generating sequential behaviors (21; 35).

**Neural networks population structure and temporal structure**. Here we set out to systematically study temporal structures, and described how NLSC can be combined to produce sequences with various forms of temporal structures. Previous works studied context-dependent computations (48) and found that external contextual inputs could reconfigure functional interactions between cognitive variables and inputs, provided that networks are structured into populations (40). Some population implementing attractor dynamics similar to dictionaries and other populations, gain-modulated by the contextual inputs, similar to gate neurons. Here we treated input from dictionary states onto gate neurons as contextual inputs, such that, from the dynamic-mean-field point of view, cognitive variables in one dictionary controls the functional interactions between cognitive variables of another dictionary (SI Fig.1e,f, Method 5.4), with bi-directional communications between two levels allowing structured sequences to unfold (Fig.4b,c). Storage capacity calculations coupled with a minimization principle have allowed us to relate the population structure of the architectures with the behavior they implement. We thus provide a theoretical background that supports our previous observations regarding the emergence of population structure in ANN (40): gate neurons of each NLSC segregate into populations to mix current dictionary states and external inputs into linearly separable representations (Fig.2g,j), while neural architectures are segregated into NLSC depending on computational demand (Fig.5h).

**Limitation to point attractors**. As for the Hopfield model, stable neural patterns of the dictionaries are scattered randomly in state space. Modeling work and experimental data have shown that for specific computations, neural networks support stable neural patterns which are arranged on manifolds with specific geometries (112; 103; 104; 105). For instance for the head-direction system, these neural patterns can be arranged on a ring. Two nearby head-orientations are encoded by bumps of neural activity that overlap to an extent determined by the width of the bumps. It controls the dimensionality of the manifold, and the head-direction model we presented here (Fig.2b) corresponds to a limiting case of very small width and high-dimensional activity. It would be interesting to compute how the number of gate neurons depends on this width, and more generally to describe how properties of gate neurons depends on the geometry of the manifold of stable neural patterns.

**Computations through neural dynamics perspective**. The computations performed by Hopfield or attractor networks are obtained by convergence towards fixed points. Dynamical features such as fixed points, stable and unstable, or limit cycles have been proposed to underlie other forms of autonomous neural computations. This point of view is often referred to as computation through neural dynamics (31). Computations on time varying external inputs have also been described within this framework, both for inputs aligned with the network recurrent structure, allowing temporal integration (48) or recognition of sequences of inputs (113), as well as for inputs orthogonal to the network recurrent structure, allowing contextual inputs to reconfigure network's dynamics (48),(31),(40). We have shown in (40) how such external inputs, by modulating the gains of specific populations of neurons, allow to reconfigure the interactions between aligned inputs and cognitive variables sustained by recurrent activity. Here we have shown how external go cues can also reconfigure interactions among cognitive variables supported by the recurrent connectivity of NLSC's dictionaries, by controlling the gains of specific populations of gate neurons (SI

Fig.1a-d, Method 5.4). We have also shown how the activation of high-level cognitive variables can play the same role as these external inputs and reconfigure interactions between lower-level cognitive variables (SI Fig.1e-g). These modulatory interactions among cognitive variables are key to organizing communication among NLSC so as to endow neural architectures with elaborated computational abilities.

#### 6.10.2 Neural networks as automata

**Functioning of neural Turing machines**. In order to provide a well defined notion of temporal structure, we interpreted cognitive variables and their associated cell-assembly activations as automata states. This has allowed us to interpret various forms of temporal structures as restrictions on the computational abilities of automata. By building Turing machines, we have shown that combining NLSC allows in principle to implement any program, procedure or algorithm. A program is implemented by wiring connectivity between dictionaries and gates in order to have the correct context-dependent transitions in the dictionaries of the external memory, the machine and the head circuits. As input of the program, the external memory is loaded by setting the dictionary of each NLSC into persistent states representing cells' symbols. Transitions between machine, head and cells states correspond to computation unfolding in time. The output of the program is the final state of the tape or of the machine.

**Relationship with previous studies**. Older works treated neural networks as automata. For instance McCulloch and Pitts put forward a recipe to implement any finite-state-automaton with a neural network (30). A main difference between this and the implementation with NLSC regards the timing of computation, with the computational clock equated with single neuron's update, to be contrasted with the external go cues or oscillations used here to pace computation. Another difference is that machine states have attractor properties, rendering them robust to individual neural or synaptic failures, and that they are encoded with sparse distributed neural patterns, leading to neural architectures with a total number of neurons scaling sub-linearly with the number of machine states (Fig.1i,(43; 106)) smaller than the architecture of (30) which has a total number of neurons scaling linearly with both number of machine states and number of inputs. Recent works proposed a similar correspondance between attractor states and machine states: (36) could estimate numerically a minimal number of neurons required to implement finite-state-automata, they also found that sparse coding was beneficial to minimize number of neurons. In (37) the authors proposed to combine non-distributed cell-assemblies to implement Turing machine, implementation of the external memory differs from ours in that, for instance, it uses plasticity to update the external memory.

**Finite neural networks**. In practice, the neural Turing machines we introduced have a finite tape as any practical machine. In order to state that any program can be implemented by combining NLSC, we follow classic arguments (30) and assume that new cells can be added to the external memory if the computation requires it. The quantitative theory we developed allows to compute how many neurons are required to add a new cell to a given architecture. We note that adding a new cell requires adding a new NLSC with the correct wiring with the FSA to implement the set of transitions between cell symbols and machine states associated with the program to implement. We note that human working memory is rather limited, more or less seven items, such that

looking to build machines with infinite memory might not be particularly relevant to model the functioning of brain circuits.

### 6.10.3 Relationship with brain circuits

**Recurrent circuitry and cell-assemblies**. In dictionaries of NLSC, neural dynamics exhibits attractor states which have been shown to be well suited to encode cognitive variables involved in specific brain computations (13). Early physiological characterizations of brain circuits align well with the notion of attractor neural networks, with persistent activity recorded in various cortical areas during memorization phases of delay-match-to-sample tasks (114; 115; 15; 116). Modern experimental techniques, allowing to perturb neural states in behaving animals, have allowed to directly probe the attracting nature of neural states linked to behavior, for instance in motor-cortex (117) or in fly's ellipsoid body responsible for the encoding of head-orientation (118). On the anatomy side, the recurrent circuitry of the brain appears well suited to implement attractor states, for instance it has be shown that statistical features of local cortical connectivity matches those of attractor neural network's connectivity optimizing memory storage (119; 120).

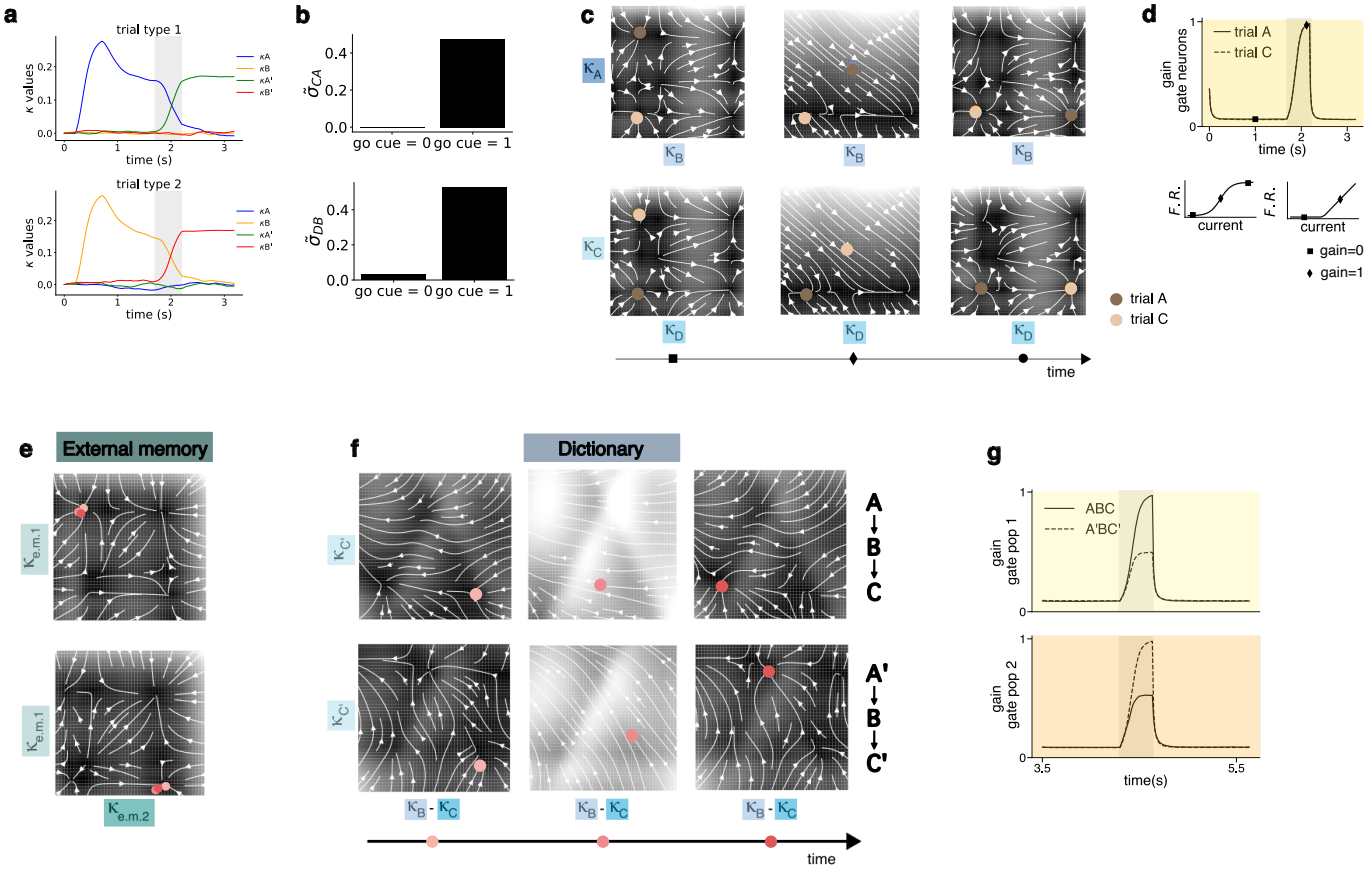
**Thalamic circuits as gate neurons**. The other ingredients of the NLSC are gate neurons, in the main discussion we pointed at works in invertebrates exhibiting gate neurons and wondered how NLSC could be implemented in mammalian brains. As mentioned, cortical networks with their rich recurrent connectivity are well suited to implement dictionaries. A possible neural substrate for gates is thalamic circuits (Fig.1). The cognitive variable interpretation (Fig.1f, Fig.3d,e), in which gates allow for the reconfiguration of interactions between cognitive variables (Method 5.4), assigns a well defined cognitive role for gate circuits. This would be in line with the ideas that cortical's functional connectivity is under the control of thalamic activity (50) and that specific cortico-thalamic and thalamo-cortical connectivity motifs (121) implement specific programs. We note however that thalamic circuits might not be well suited for long-term storage of programs given the relatively few number of thalamic neurons compared to neocortical ones (roughly one for ten).

**NLSC and cortical columns**. Another possibility we mentioned is that cortical columns (29) implement NLSC: L2/3 neurons are well suited to implement dictionaries given the observation of persistent activity and its connectivity statistics (119). L2/3 neurons of primary motor cortex of rodents have been shown to mix representations of one of two task contexts with current motor kinematics (122), and could serve as transition gate neurons and be well suited to trigger transitions between neural representations driving the sequential behavior associated to each context. In architectures implementing Turing machines or producing sequences with an algebraic pattern structure, gate neurons, in addition to trigger transitions between dictionary states (*transition* gate neurons), are used to control whether the dictionary of a given NLSC sends its current state to another NLSC (*output* gate neurons, Fig.4g). It is tempting to identify these gate neurons with L5/6 neurons that have been proposed to serve as outputs of the cortical column (123; 124). Gain modulation of L2/3 or L5 neurons via long-range cortical inputs, implemented either through local micro-circuitry (71) or dendritic (125) non-linearities, could mediate interactions between NLSC. In our neural architectures, transition gate neurons are exclusively driving their target dictionary neurons. By contrast, output gate neurons can both drive or modulate their target, depending on

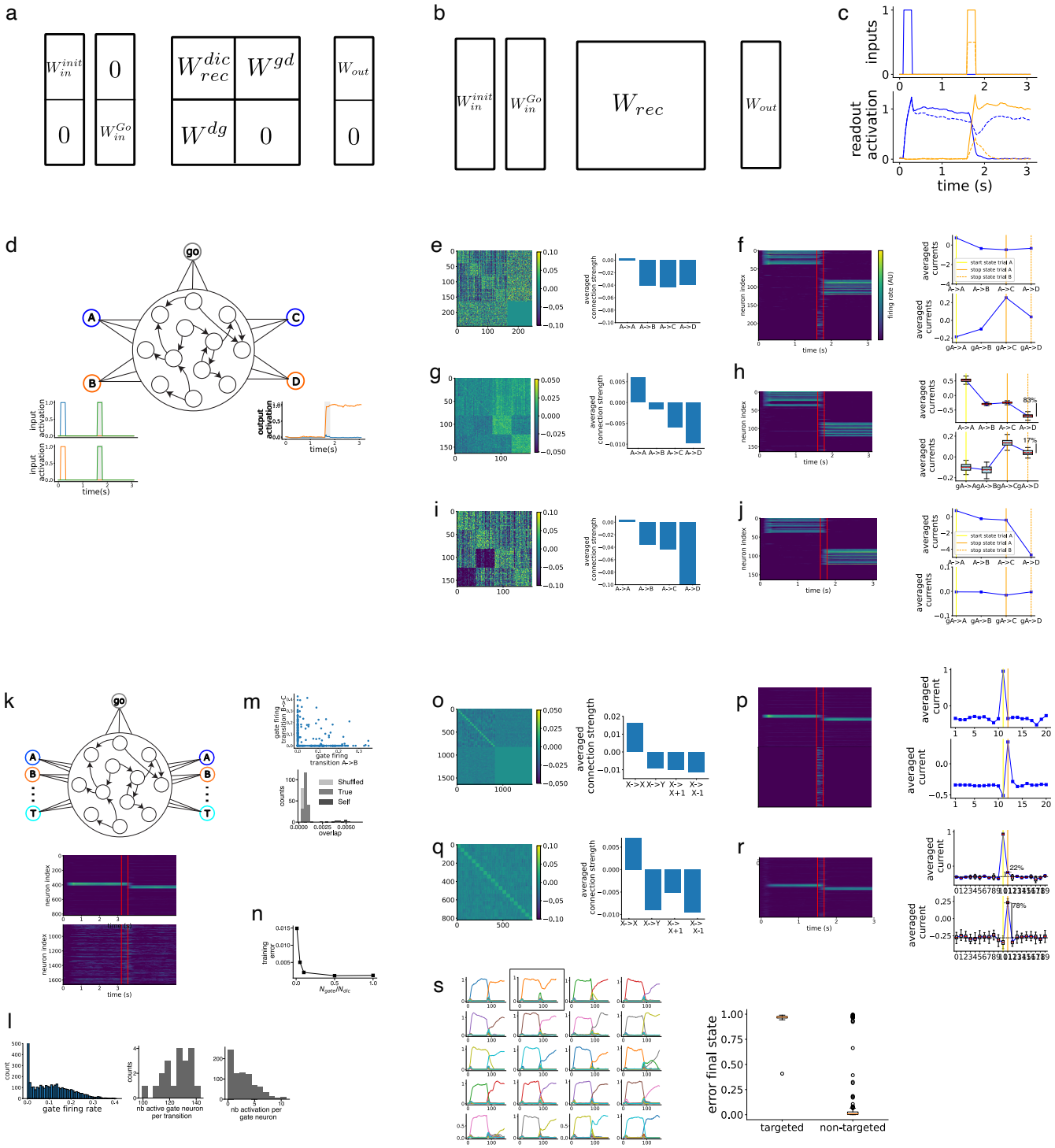
whether it targets a NLSC with higher or lower number of dictionary states, this could be checked in experimental data.

**Oscillatory activity**. There are experimental evidence that local microcircuitry could support autonomous state transitions through oscillations, for instance it has been observed that populations of hippocampal PV cells oscillating at $\gamma$-frequency update their state at each oscillation cycle (81). Hippocampal replays at $\theta$-frequency could also be interpreted as updating the state of a NLSC. It would also be nice to understand how oscillations can coordinate various NLSC involved in running a hierarchy of sensory-motor programs. Along such a line it has been observed that in an attentional system such as the frontal-eye-field, LFP recordings exhibit $\beta$-oscillations while animals switch their attention at $\beta$-frequency in a spatial visual attention task (126).

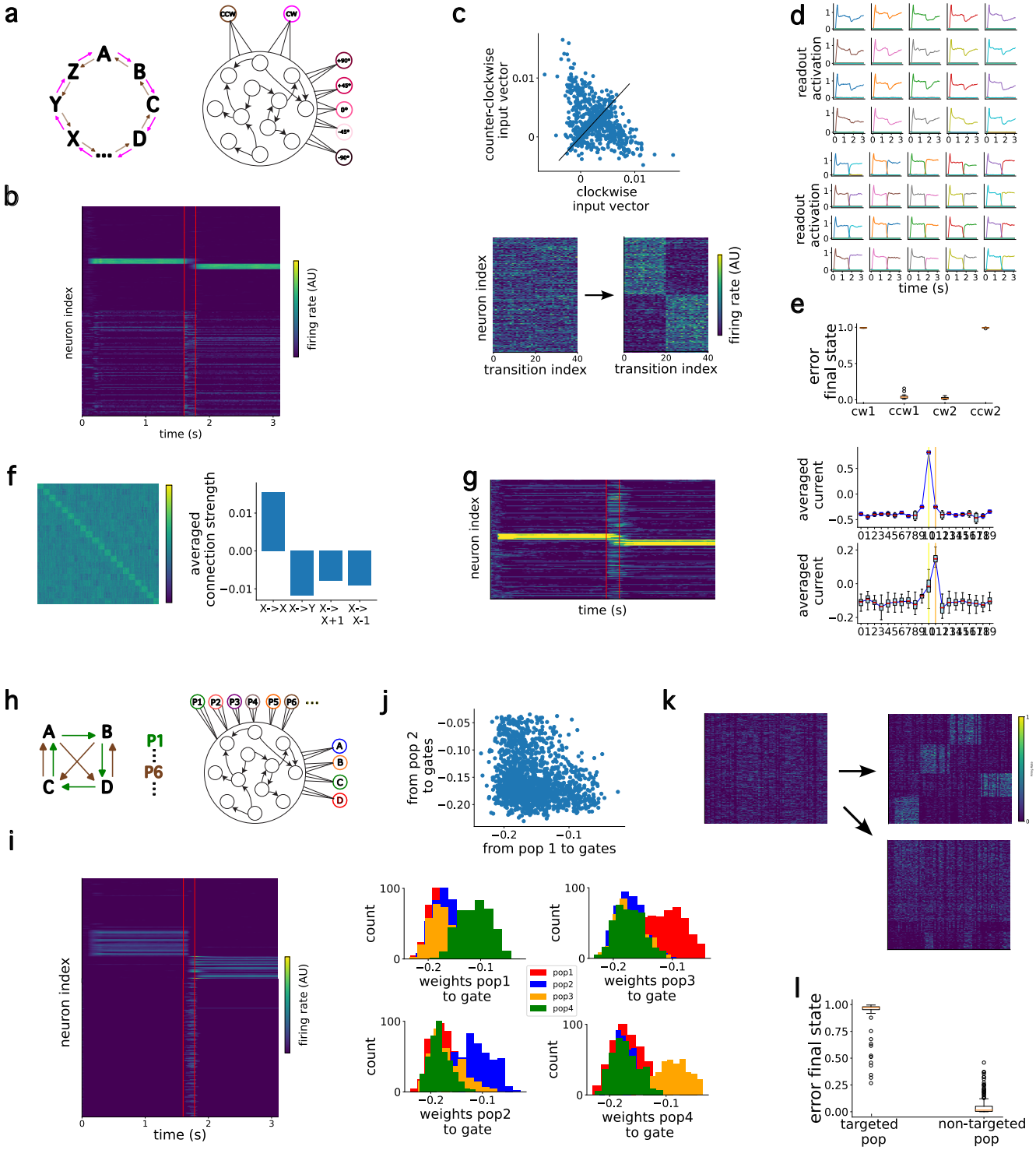
# Supplementary figures



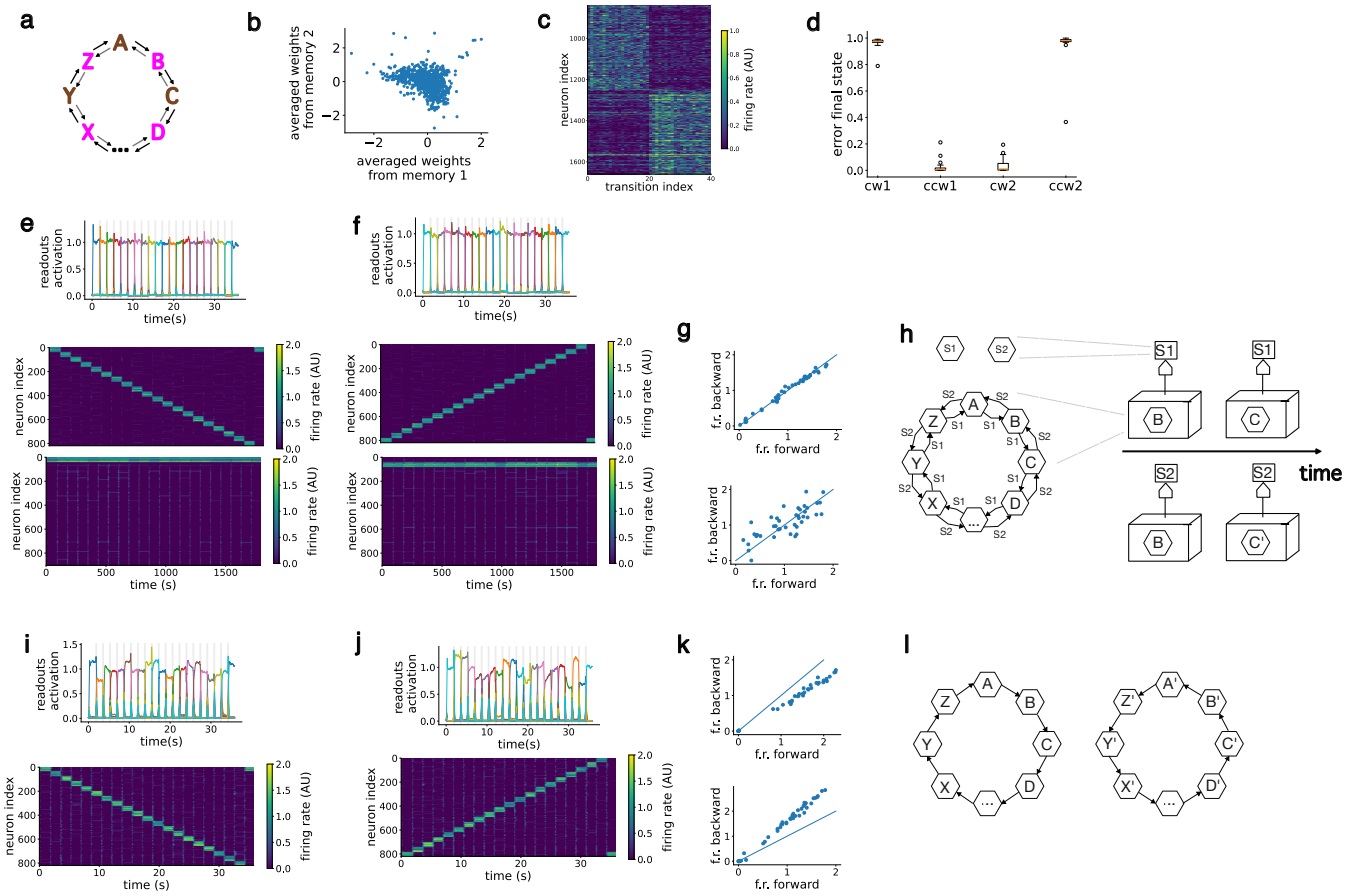

**SI Fig. 1:** ***Reduced dynamical systems for NLSC.*** ***a****, Sensory-motor transformation task: time evolution of the macroscopic variables describing the activation of cell-assemblies during task performance.* ***b****, Interactions between macroscopic variables are modified by the go cue: activation transfers neural activity from state A to C or B to D.* ***c****, Flow fields for the 4-D reduced dynamical system of interacting cognitive variables. Light and dark brown dots represent current network state in trials A to B and C to D respectively.* ***d****, Flow-field is reshaped by setting the gain of a gate neurons from 0 to 1 in both types of trials.* ***e****, Flow-fields depicting the dynamics in the subspace representing the neural activity in the external memory, points of different colors represent neural activity location at different time points throughout sequence production.* ***f****, Flow-fields in a subspace representing activity in the dictionary, we chose a cut allowing to visualize the three fixed points associated with the production of B, C and C′.* ***g****, Gain of the two populations of gate neurons routing activity from B to C or B to C′.*


a
b
c
inputs
readout activation
time (s)
d
go
A
B
C
D
input activation
output activation
time(s)
e
f
g
h
i
j
averaged connection strength
A->A A->B A->C A->D
neuron index
time (s)
firing rate (AU)
averaged currents
start state trial A
stop state trial A
stop state trial B
gA->A gA->B gA->C gA->D
83%
17%
gA->A gA->B gA->C gA->D
k
T
m
gate firing transition A->B
counts
Shuffled
True
Self
overlap
n
training error
l
count
gate firing rate
nb active gate neuron per transition
nb activation per gate neuron
o
p
q
r
averaged connection strength
X->X X->Y X->X+1 X->X-1
averaged current
22%
78%
s
error final state
targeted
non-targeted

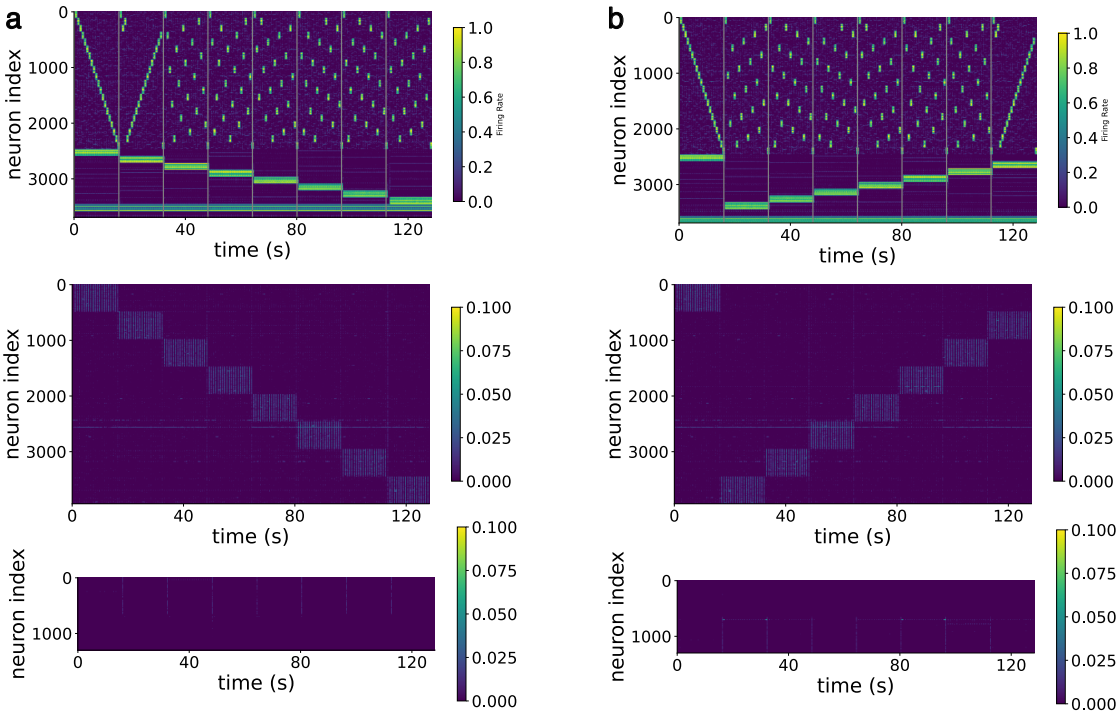
**SI Fig. 2:** ***Neural Latching Switch Circuits in ANN trained on tasks with simple temporal structure.*** ***a***, *Schematic of constraints on the connectivity for training NLSC, or* ***b*** *unconstrained networks.* ***c***, *Top: inputs for two trials (plain and dashed lines) of the dictionary task. Bottom: activations of the blue and orange readout neurons for a trained networks in response to these inputs.* ***d***, *Schematic of an ANN trained on the sensory-motor transformation task, below are inputs and outputs for the first (first row) and second (second row) trial type.* ***e***, *Left: connectivity matrix of a NLSC trained on the sensory-motor transformation task. Right: average connection strengths over blocks of the matrix. First bar: connection strength among neurons belonging to the same cell-assembly ; second bar: connection strength in between cell-assemblies A and B, third bar: connection strength from cell-assemblies A to C and B to D, fourth bar: connection strength from A to D and B to C.* ***f***, *Left: raster of neural activity for a singe trial showing the activity of all neurons in the trained NLSC. Red lines materialize activation of the go cue. Right, top: averaged currents produced during the maintenance epoch by the maintenance state to: the maintenance state itself (e.g. A), the other potential maintenance state (e.g. B), the response state (e.g. C) and the other potential response state (e.g. D). Right, bottom: averaged currents produced during the go epoch by gate neurons.* ***g***, *Same as* ***e*** *for unconstrained networks trained on the task. Left: example network, right: averaged over 50 networks.* ***h***, *Same as* ***f***, *with currents on the right averaged over the 50 networks. For the bottom plot, what we consider as gate neurons are the neurons that are not supposed to be active in that trial (e.g. neurons belonging to B and D for the first type of trials), but some of which are activated during the go epoch, see left.* ***i,j*** *Same as* ***e,f*** *for the network without gate neurons (see text).* ***k***, *Schematic of an ANN trained on the permutation task, with raster of the NLSC with 820 dictionary and 820 gate neurons described in the main text.* ***l***, *Left: Distribution of activations for the 820 gate neurons for the 20 transitions. Middle: number of gate neurons active in one transition. Right: number of transitions in which a gate neuron is activated.* ***m***, *Top: comparing activations of all gate neurons for two particular transitions. Bottom: across all transitions as measured by the overlaps between pairs of gate patterns.* ***n***, *Training error as a function of the ratio between numbers of gate and dictionary neurons.* ***o***, *Left: connectivity matrix of a NLSC trained on the permutation task. Right: average connection strengths over blocks of the matrix. First bar: connection strength among neurons belonging to the same cell-assembly ; second bar: connection strength in between two cell-assemblies X and Y$\neq$ X, X-1, X+1 ; third bar: connection strength from X to X+1 ; fourth bar: connection strength from X to X-1.* ***p***, *Left: raster of neural activity for a single trial showing the activity of all neurons in the trained NLSC. Red lines materialize activation of the go cue. Right, top: averaged currents produced during the maintenance epoch by the maintenance state to each of the states. The average is performed over neurons and over transitions. X-axis represents the index of the states shifted for each transition such that the maintenance state has index 11 (yellow line) and the response state has index 12 (orange line). Right, bottom: averaged currents produced during the go epoch by gate neurons to each of the states.* ***q***, *Same as* ***p*** *for the averaged connectivity of 50 unconstrained networks trained on the permutation task.* ***r***, *Same as (Q) for the unconstrained networks. For the bottom plot, what we consider as gate neurons, for the transition from state X, are the neurons that belong to cell-assembly Y $\neq$ X, X-1, X+1.* ***s***, *Left: Readout activations for all the 20 transitions while inactivating 50 gate neurons during the go epoch. It prevents only the second transition (black square). The inactivated neurons are taken as the 50 with highest firing rate during the unperturbed transitions, with the constraint that they belong neither to the former, current, following activated cell assembly. Right: statistics of inactivations over all transitions. For each perturbed transition X, error in final state is measured as* $(z_{X+1}(t = end) - 1)^2$.


a
A
B
C
D
X
Y
Z
CCW
CW
+90°
+45°
0°
-45°
-90°
b
neuron index
time (s)
firing rate (AU)
c
counter-clockwise input vector
clockwise input vector
neuron index
transition index
firing rate (AU)
d
readout activation
time (s)
e
error final state
cw1
ccw1
cw2
ccw2
f
averaged connection strength
X->X
X->Y
X->X+1
X->X-1
g
neuron index
time (s)
averaged current
h
A
B
C
D
P1
P6
i
neuron index
time (s)
firing rate (AU)
j
from pop 2 to gates
from pop 1 to gates
count
weights pop1 to gate
weights pop3 to gate
weights pop2 to gate
weights pop4 to gate
pop1
pop2
pop3
pop4
k
l
error final state
targeted pop
non-targeted pop

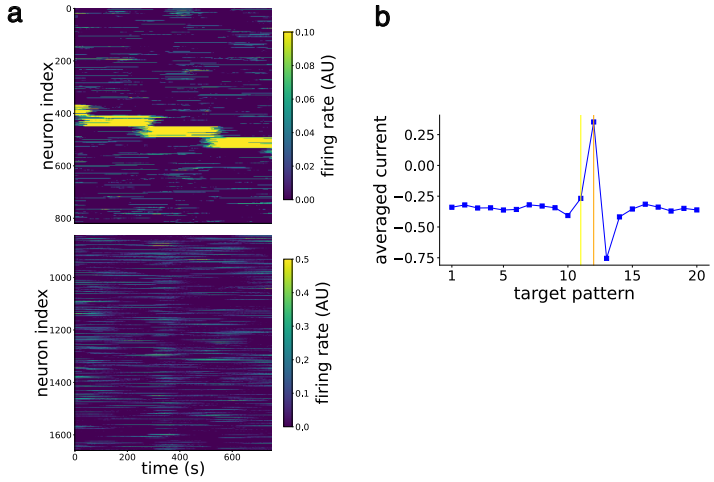
**SI Fig. 3:** ***Head-direction system and multiple permutations over dictionary states***. ***a***, *Schematic of an ANN trained on the head-direction task, where readout neurons and associated cell-assemblies in the recurrent network encodes angular positions.* ***b***, *For a single transition, raster of the NLSC with 820 dictionary and 820 gate neurons described in the main text.* ***c***, *Top: individual entries of the two trained vestibular input connectivity vectors. Bottom: Reordering of the raster of activations of gate neurons for the clockwise and counter-clockwise transitions.* ***d***, *Readouts activations for all the transitions (top clockwise, bottom counter-clockwise transitions) while inactivating the first population of gate neurons during the go epoch.* ***e***, *Statistics of inactivation results for all transitions. The first two columns quantifies what is shown in (D), the next two represent inactivations of the second population of gate neurons.* ***f***, *Averaged connectivity for 50 unconstrained networks trained on the head-direction task.* ***g***, *Neural activity (left) and averaged currents generated by the maintenance state (right top) and transient activations during the go epoch (right bottom).* ***h***, *Schematic of an ANN trained to implement all possible permutations on 4 states.* ***i***, *For a single transition, raster of the NLSC with 164 dictionary and 1640 gate neurons described in the main text.* ***j***, *Top: For individual gate neurons, averaged connectivity from dictionary neurons belonging to cell-assembly encoding state 1 and state 2. Bottom: Each panel shows the histogram of averaged connectivity from dictionary neurons belonging to a cell-assembly. Gate neurons are split into four groups (colors, see text).* ***k***, *Reordering of the raster of gate neurons activations according to clustering results.* ***l***, *Statistics of inactivation results for all transitions and inactivation of the four populations.*



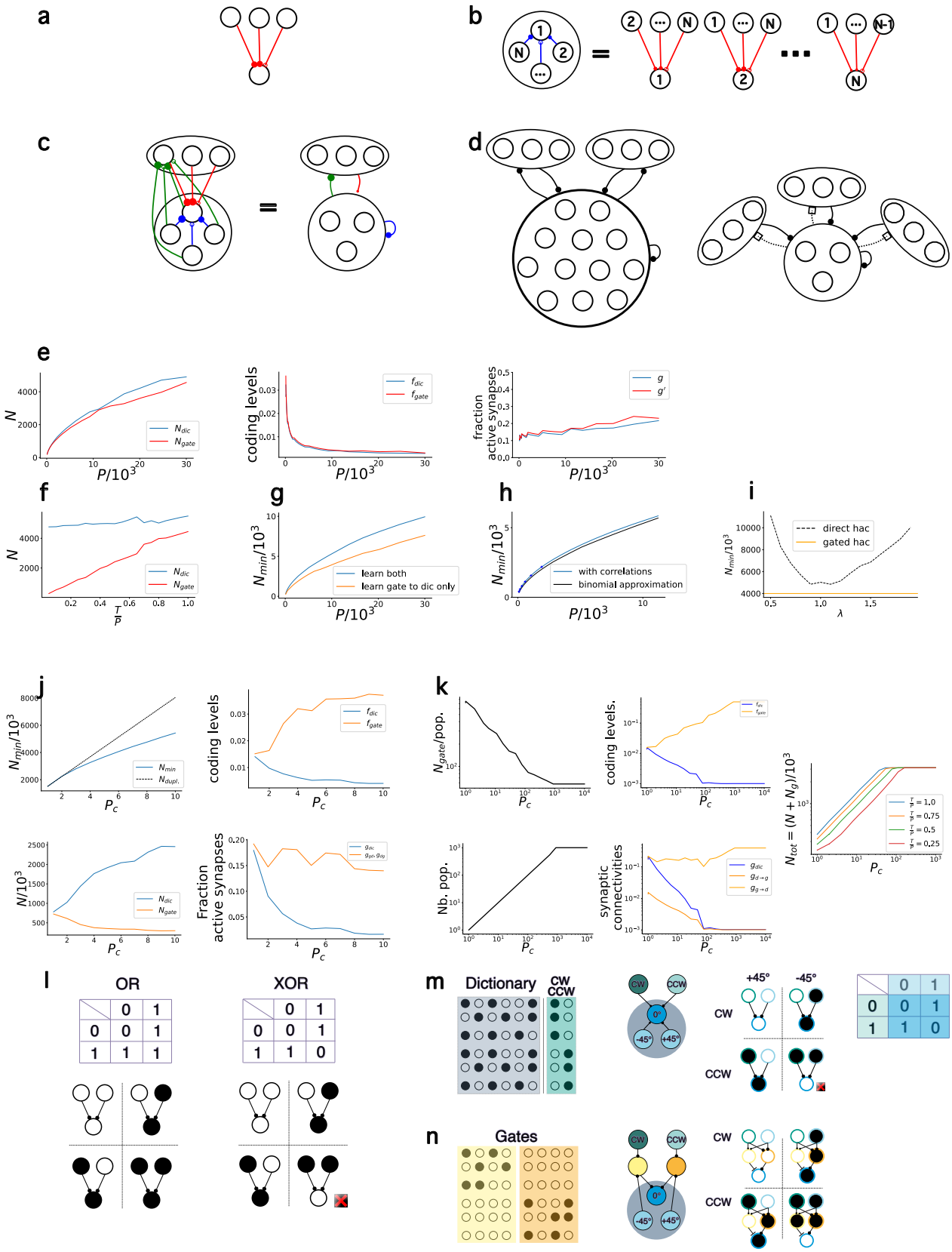

**SI Fig. 4:** ***Neural networks for long-distance-dependencies.*** ***a****, Illustration for the tasks with 20 states.* ***b****, Averaged weights from the two cell-assemblies of the external memory onto gate neurons of the NLSC, for the network discussed in Method 5.3.5.* ***c****, Raster of gate activation patterns for the* $2 * 20$ *transitions.* ***d****, Results of inactivating the two gate sub-populations in each of the two contexts.* ***e****, Activity of output (top), dictionary of the NLSC (middle) and external memory and gate neurons (bottom) in the first,* ***f*** *second context.* ***g****, Activation of a cell-assembly of dictionary neurons while the network produces a symbol in both contexts. Top and bottom show two examples.* ***h****, Automata cartoon summarizing the functioning of the NLSC augmented with an external memory.* ***i-l****, Same as* ***e-h*** *for an unconstrained networks trained on this task.*



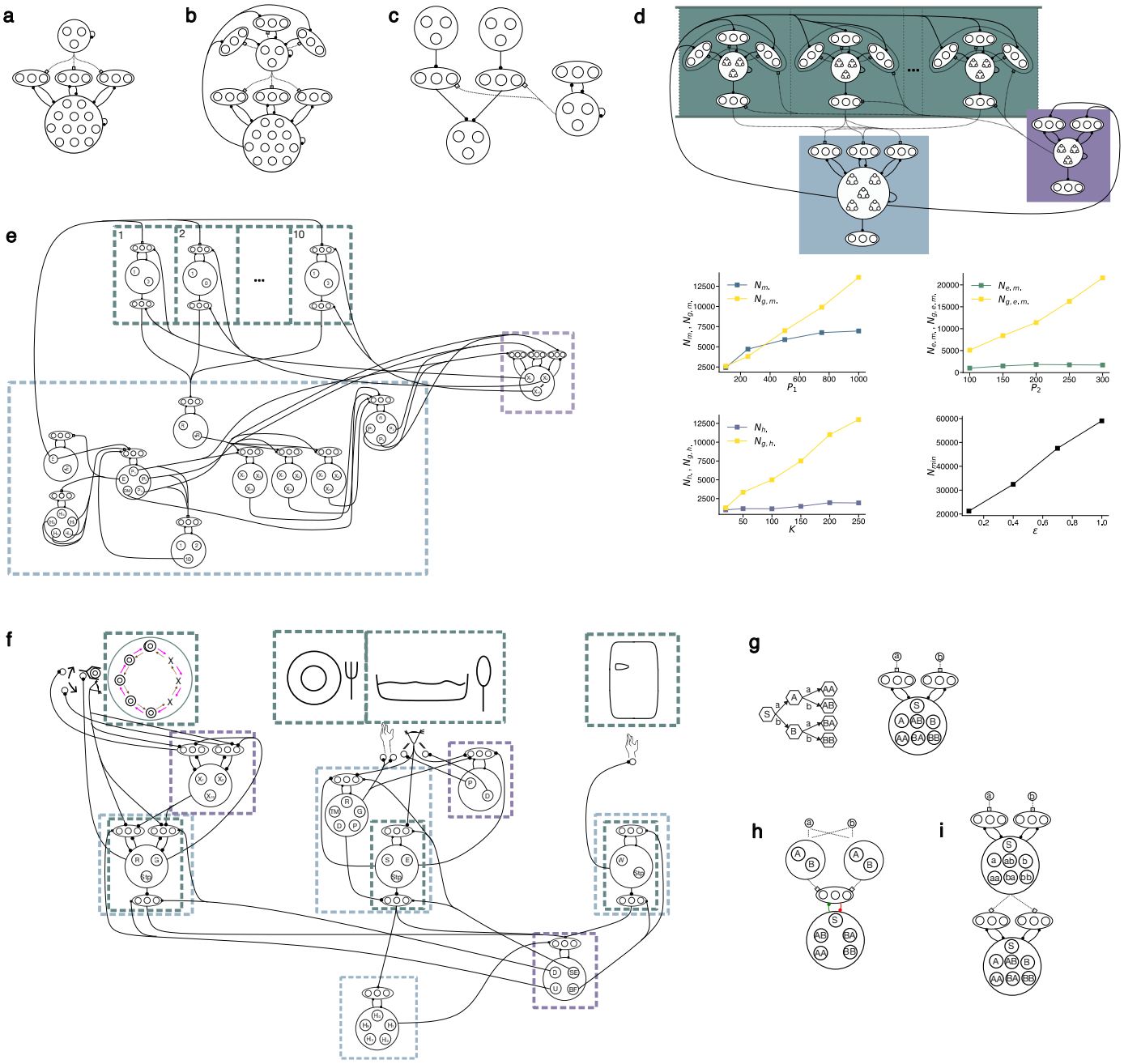

**SI Fig. 5:** ***Piling-up NLSC.***. *For the architecture presented in Fig. 4b, rasters of activations of dictionary neurons in the three NLSC (top), gate neurons associated with the lowest-level NLSC (middle), gate neurons associated with the high-level NLSC while playing sentence S1* ***a*** *or S2* ***b****.*




**SI Fig. 6:** ***Relationship with oscillatory activity***. **a,b**, NLSC trained to autonomously update its dictionary states. **a**, Raster of activity for three transitions. **b**, Average current send from gate neurons to cell-assemblies of the dictionary, averaged on all transitions, recentered on the transition from cell-assembly 11 to cell-assembly 12.


a
b
c
d
e
f
g
h
i
j
k
l
m
n
OR
XOR
Dictionary
Gates
CW
CCW
+45°
-45°
coding levels
fraction active synapses
learn both
learn gate to dic only
with correlations
binomial approximation
direct hac
gated hac
Nb. pop.
synaptic connectivities

**SI Fig. 7:** ***Capacity analysis of the NLSC.*** *Cartoons for* ***a*** *a perceptron,* ***b*** *an attractor neural network,* ***c*** *a NLSC with a single population of gate neurons implementing a single permutation among dictionary states,* ***d*** *NLSC with multiple populations of gate neurons implementing multiple permutations on a large (left) or small (rigth) dictionary.* ***e****, Description of an optimized NLSC implementing a single permutation with an increasing number of dictionary states: number of dictionary and gate neurons (left), coding levels (middle), average synaptic weights (right).* ***f*** *,Number of dictionary and gate neurons as a function of the number of programmed transitions* $T$*.* ***g****,* $N_{min}$ *for a network implementing a single permutation with learnt and random connectivity from dictionary to gate neurons.* ***h****,* $N_{min}$ *as computed from* ***(9)*** *(blue) or* ***(11)*** *(black).* ***i****,* $N_{min}$ *for the model with direct hetero-associative connectivity (black-dashed) compared to the NLSC model (orange).* ***j****, Same as* ***e*** *for an optimized NLSC implementing an increasing number of permutations that remain small compared to dictionary size.* ***k****, Properties of optimized NLSC in the regimes of both low and large number of permutations per dictionary states.*


a
b
c
d
e
1
2
10
...
$N_{m,}$
$N_{g,m,}$
$N_{m,}, N_{g,m,}$
$P_1$
$N_{e,m,}$
$N_{g,e,m,}$
$N_{e,m,}, N_{g,e,m,}$
$P_2$
$N_{h,}$
$N_{g,h,}$
$N_{h,}, N_{g,h,}$
$K$
$N_{min}$
$\varepsilon$
f
g
h
i

**SI Fig. 8:** ***Assembling neural architectures from multiple NLSC.*** ***a****, NLSC augmented with an external memory to produce behaviors with long-distance dependencies.* ***b****, Two NLSC piled-up, e.g. to produce sequences organized into chunks.* ***c****, Using NLSC with output gate neurons as memory slots (top), with a conditional routing to the machine NLSC (bottom) by the head NLSC (right), e.g. to produce sequences following an algebraic pattern.* ***d****, Assembling NLSC to build neural Turing machines. Bottom: Number of neurons in machine, external memory, and head circuits when the number of machine states, symbols of the external memory, external memory cells are varied.* ***e****, Assembling NLSC to implement the foraging task.* ***f****, Piling-up neural Turing machines, e.g. to implement a hierarchy of sensory-motor programs. The cartoon depicts the task and networks programmed in Method 5.3.9.* ***g****, FSA for mapping temporal sequences of symbols into meaning states (left) and its NLSC implementation (right).* ***h****, Neural architecture with an intermediary memory buffer composed of two slots implemented by dictionaries. The machinery to fill slots from sequentially presented inputs is not described, but see circuit Fig. 4g.* ***i****, Piling-up NLSC to map sequences organized into chunks.*