Episodic memories enable powerful algorithms for goal-directed decision making that are context-aware and explainable

(October 25, 2025)

Yukun Yang and Wolfgang Maass

Contributing authors: yukun.yang@tugraz.at; wolfgang.maass@tugraz.at;

Abstract

Current AI tools are often criticized for their lack of content-awareness and explainability. This make their planning and action selection sometimes undesirable for a given context, or even unsafe. In contrast, our brains can adjust planning and action selection on the fly to the current context, and often make their action selection transparent by recalling related experiences. This capability results from a fundamental difference between the way how brains and current AI tools store learnt knowledge. Whereas the latter compress all experiences into parameter values, the brain also stores representations of the most salient experiences in the form of episodic memories, which support more sophisticated action selection and planning. We show that experimental data on neural codes for episodic memories in area CA1 of the brain support a method for reproducing these capabilities in artificial devices. The resulting Episodic Neighbor(EN)-algorithm complements reinforcement learning approaches by adding inherent flexibility, context-awareness, and explainability. In addition, the EN-algorithm only needs local rules for synaptic plasticity with binary weights in shallow neural networks, and can therefore be implemented through on-chip learning in energy-efficient neuromorphic hardware.

1 Introduction

Powerful current AI methods for planning or problem solving are based on Deep Neural Networks (DNNs) or Large Language Models (LLMs). A common criticism of these tools is that they are inflexible, make it hard to quickly adjust a plan when the goal or contingencies change, or to optimize the plan for a specific context, such as planning a route for driving during rush hour, or planning a walk at night. Also their lack of explainability or interpretability has been criticized (Bekkemoen, 2024; Burkart and Huber, 2021; Perez-Cerrolaza et al., 2024; Rawas, 2024). Explainability is especially important for applications in healthcare and law, but also with regard to safety issues in autonomous driving and human-robot interaction. Other obvious problems of DNN- and LLM-based AI solutions are the very large amount of training data which they require, and the staggering amounts of energy that are needed for training and inference.

Confidence that these problems can eventually be solved is provided by the brain, for which evolution has found ways to overcome these deficits. But according to (Mattar and Lengyel, 2022), it remains largely unknown which algorithmic approaches the brain employs for achieving that. Planning was defined in this review as *the process of selecting an action or sequence of actions in terms of the desirability of their outcomes*. Importantly, this desirability may depend on distant outcomes in the future, hence some form of look-ahead is required for efficient planning, and one needs data structures which enable such foresight.

Concrete hints regarding algorithmic approaches which the brain employs arise from experimental data that elucidate the way how it stores learnt knowledge. These data structures markedly differ

from those in DNNs and LLMs. In particular, the brains maintains direct access to episodic memories (EMs) of salient experiences. One interesting facet is that the brain selects neurons which serve as indices for EMs. For example, one found neurons that fired throughout a specific episode, both while encountering this episodic experience and during subsequent recall (Gelbard-Sagiv et al., 2008; John et al., 2025; Kolibius et al., 2023; Tacikowski et al., 2024). These neural codes, which one might see as analogues to keys in LLMs, are complementary to other neural codes that encode the content (values) of EMs and their sequential structure. But their functional role in the brain has remained opaque. We show here that index neurons for EMs are very useful for planning. In fact, they give rise to cognitive maps that support navigation to distant goals both in physical and abstract problem spaces. Numerous experimental data from the rodent and human brain had previously shown that cognitive maps are likely to support navigation not only in spatial environments, but in the human brain also in abstract concept spaces (Behrens et al., 2018; Bottini and Doeller, 2020). One interesting facet of cognitive maps in the brain was recently provided by (Elston and Wallis, 2025), where it was shown that the primate brain stores and switches different cognitive maps for the same task environment, and switches between them in dependence on the current context. We show that this can also be reproduced in our model.

More precisely, we show that indices for EMs give rise to an implicit form of a cognitive maps that enables efficient context-dependent selection of those EMs that are salient for the current context and goal. They also enable an explanation of each action selection and plan in terms of specific episodic experiences which support them. One further ingredient of the resulting Episodic Neighbor (EN)-algorithm for context-aware and explainable planning is an interlinking of states and observations with own actions. This has been proposed to be a fundamental strategy of the brain (Buzsáki, 2019). For the EN-algorithm it suffices that likely consequences of each action are learned through Hebbian learning. These learned predictions enable neurons that can initiate an action to also respond to sensory inputs that inform about the outcome of this action, i.e., they become mirror neurons (Bonini et al., 2022). The name Episodic Neighbor algorithm was chosen because it shares an important aspect with the famous k-NN (k Nearest Neighbors) algorithm: It does not compress all learned knowledge into parameter values, but maintains explicit representations for learning experiences, and uses them according to their relation to the challenge at hand. But this relation has to be defined for planning and action selection differently from that in the case of the k-NN algorithm, where learning experiences are previously classified feature vectors of items, and the relation is a metric in the corresponding space of feature vectors. Instead, the EN-algorithm estimates for an action selection the correlation between previously experienced outcomes of specific actions and experiences that involved the currently given goal. This strategy gives rise to a cognitive map based on correlations between experiences, and supports context-aware decision making by activating only those EMs that are salient for the current context. It also provides automatically an explanation for each action selection in terms of specific EMs that support it.

Learning of EMs requires, in contrast to statistical learning, rules for synaptic plasticity that are suitable for one-shot learning. A plasticity rule which achieves that had been discovered in (Bittner et al., 2017), hence in the same brain CA1 of the hippocampus where indices for EMs were found in the human brain according to the previously cited literature. BTSP has so far only been documented for the rodent brain. But the seconds-long learning window and fast expression of weight changes that is demonstrated by the data of (Gelbard-Sagiv et al., 2008; John et al., 2025) from the human brain point to the presence of a similar rule for synaptic plasticity in area CA1 of the human brain. We show here that the simple model for BTSP from (Wu and Maass, 2025) does in fact suffice for creating a cognitive map from individual EMs that supports context-aware and explainable planning.

The EN-algorithm shares with Transformers (Vaswani et al., 2017) and other LLMs the property that it does not require a teacher for learning, and that observations are projected into a space of high-dimensional vectors. We have already mentioned the analogy between neurons in area CA1 that provide indices for EMs and keys in transformers. In addition, a given planning goal can be seen as analogue to a query, and the content (observations) of an EM as values. Furthermore, both LLMs and the EN-algorithm apply some form of WTA (Winner-Take-All) for producing an output. But in contrast to LLMs, the EN-algorithm neither requires deep learning nor large amounts of data, and can be implemented in an energy-efficient sparse activity regime.

We will first explain the basic principles of the EN-algorithm in a generic setting for planning and problem solving: finding a shortest path from some start to some goal node in an abstract graph. The EN-algorithm does not require any prior knowledge about the graph, and explores it instead in a self-supervised manner. We then show that close to optimal performance of the EN-algorithm holds also

for large random graphs. Subsequently we show that the EN algorithm can also be applied to online path finding in large directed and stochastic graphs, i.e. to Markov Decision Processes (MDPs), where action outcomes are not deterministic, and actions are in general not reversible. Also in these more challenging environments the EN-algorithm produces approximately optimal plans for reaching any given goal in an online manner, using just local synaptic plasticity in shallow networks. Hence the EN-algorithm can be readily implemented through on-chip learning in energy-efficient neuromorphic hardware that enables action selection with lower latency than DNN- or LLM-based approaches. Altogether, the EN-algorithm offers in comparison with RL approaches both significant functional advantages and more energy-efficient implementation options. In fact, it appears to be especially suited for energy-efficient computations with sparse activity in various types of large neuromorphic systems that are currently built (Kudithipudi et al., 2025; Zhang et al., 2020).

2 Results

2.1 Goal-directed decision making with episodic memories

A fundamental concept for goal-directed decision making and problem solving is that of a state, which summarizes those aspects of current and recent external sensory inputs and internal signals that are salient for action selection and planning in a chosen task domain. In Reinforcement Learning (RL) and classical symbolic AI approaches the states were defined for simpler task domains explicitly by the algorithm designer (Russell and Norvig, 2020; Sutton and Barto, 2018). For challenging task domains the states are defined nowadays by a DNN or LLM. This works quite well, although planning is not considered to be a major strength of current LLMs. But it makes the planning and decision making process opaque for the human user.

We introduce here a radically different approach for the definition of states. It emerges from autonomous exploration of a task environment. In contrast to DNN and LLM approaches this knowledge is not compressed into synaptic weights or other parameters, which would make it impossible to select specific knowledge that is most salient for the current context. Loosely speaking, the EN-algorithm defines the state $Q(o)$ that is triggered by the current observation o of external and/or internal signals as the set of indices for EMs in which the same or similar observations have occurred in the past. We illustrate this approach in Fig. 1 for a simple standard model for planning and problem solving domains (Russell and Norvig, 2020), a graph, see Fig. 1A. Three sample trajectories e from exploring this graph are marked by different colors in Fig. 1B. Inspired by area CA1 of the brain, we assume that there exists for each of these exploration episodes e a neuron $n(e)$ which fires both during experiencing and later during recall of this exploration episode. The brain actually dedicates several neurons $n(e)$ to each EM e, which makes the neural code more robust. But in our model it suffices to dedicate just a single neuron $n(e)$ to each episode e, see Fig. 1C. Provided that the input to these neurons is about as sparse as the input to pyramidal cells in area CA1 of the brain, see (Wu and Maass, 2025) for references to related biological data, one can use the simple version of BTSP from that article for inducing index neurons $n(e)$ in a model. As indicated in Fig. 1D, these indices for EMs make it possible to represent spatial locations, landmarks, concepts in terms of the EMs in which they occurred. In other words, each content item is represented by the web of relations between it and other items that are created through EMs in which it occurred. Note that also the human brain embeds each location or concept, say a particular restaurant, into a web of EMs in which it was involved. In Fig. 1 each item is represented by a node o of the given graph, and it becomes now represented by a high-D binary vector that is defined by the set of neurons $n(e)$ that represent indices for exploration episodes e in which this node was visited (Fig. 1D). This defines an embeddings $\mathbf{Q}$ of the graph nodes into a high-S state space that captures salient relations between the nodes, analogously to the embedding $\mathbf{Q}$ that was defined in (Stöckl et al., 2024) for simpler task environments. We refer to Section 4.1 for details of this embedding.

For an application of the EN-algorithm it only remains to relate these high-D representations of states to actions, more precisely to the activity of neurons n_a in another area that can trigger the execution of a particular action a. To do that, it suffices to define a matrix $\mathbf{W}$ that maps states to actions through one-shot learning with a local rule for Hebbian plasticity. This rule sets a weight from 0 to 1 if the presynaptic neuron represents an EM that occurred right after the execution of action a. It is illustrated in Fig. 1E for the case that its action triggers a move from node 9 to node 2, thereby activating the neurons that fire according to Fig. 1D for the embedding of node 2 into the

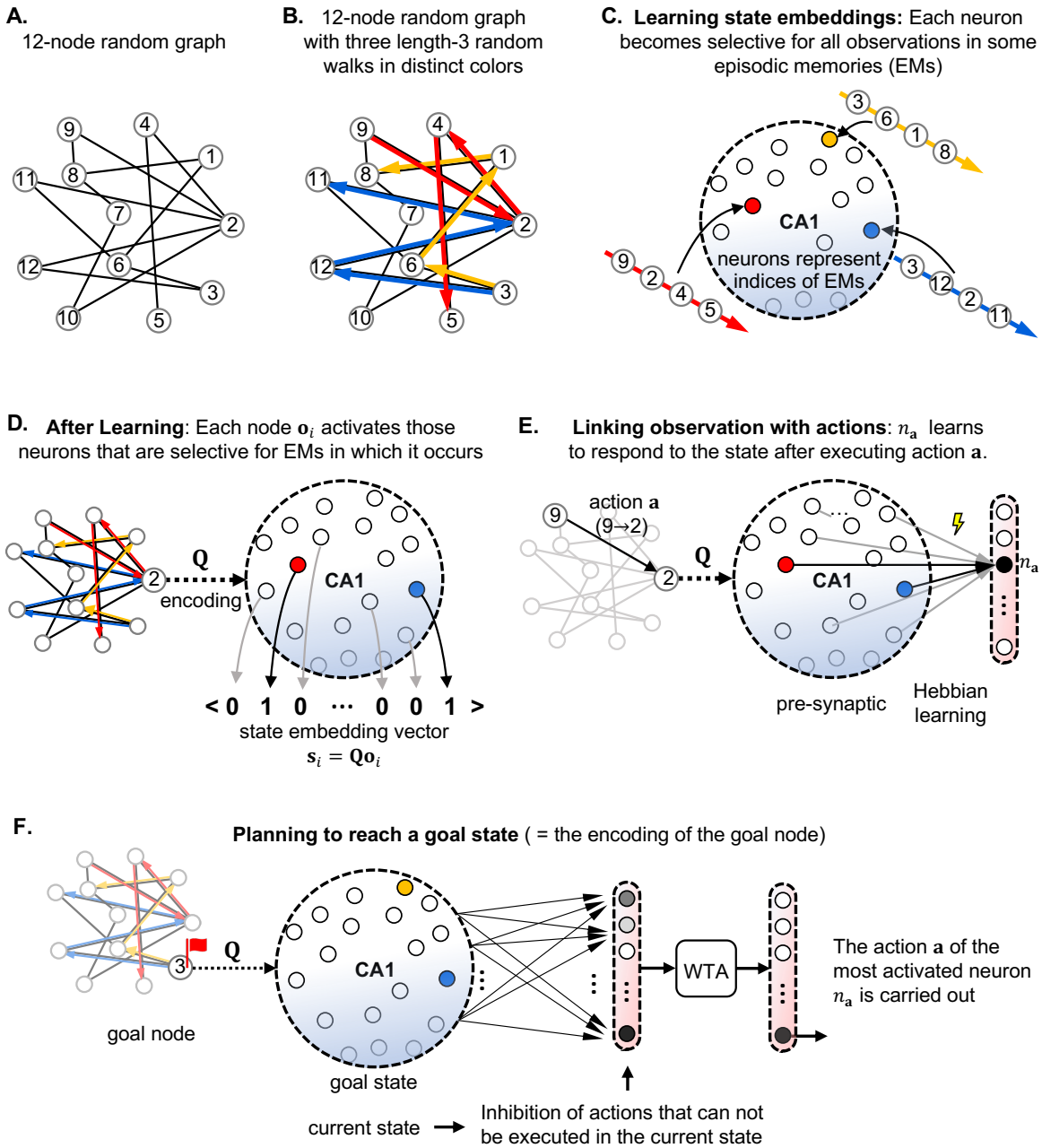


Fig. 1: **Network- and learning-architecture for the EN-algorithm.** **(A)** A small random graph, which serves as running example for this figure. **(B)** Three sample exploration trajectories indicated by three colors. **(C)** Population of neurons, corresponding to area CA1 in the brain, where each neuron becomes the selective for the codes of the graph nodes visited during a specific EM. This selectivity can be created for example through BTSP. **(D)** Definition of the state Q(o) for a sample node $o = 2$ in the graph. It is a binary vector which assumes value 1 for each episode in which this node 2 was visited. **(E)** Relating states to actions through Hebbian synaptic plasticity: A neuron n_a that usually triggers a specific action a learns to respond also to the observation (state) that results from executing this action. **(F)** The EN-algorithm for online planning of a short path from the current node to an adhoc given goal node o^* (marked by a red flag). It chooses as first action that one where the resulting next state co-occurs with the goal in the largest number of EMs. This action selection can be implemented through a WTA (Winner-Take-All) network motif, applied to all actions $\mathbf{a}$ that can be executed in the current state. The next step towards the goal is chosen by iterating this process at the next state.

high-D state space. As a result, the neuron n_a becomes activated by an observation of the outcome of its action a, like mirror neurons in the brain (Bonini et al., 2022; Mukamel et al., 2010).

Surprisingly, this simple learning method suffices for planning a path from any start node to any given goal node in the task environment, as indicated in Fig. 1F. The one-hot representation $\mathbf{o}^*$ of the goal node o^* (node 3 in this example) is mapped by $\mathbf{Q}$ into a corresponding state that activates all neurons that represent EMs in which it had previously occurred (See supplementary Sec. A for cases observations are non-orthogonal). If all action neurons n_a are inhibited that cannot be executed in the current state, the EN-algorithm chooses one of the remaining actions a for which the neuron n_a is strongest activated by the goal state. This simple rule will select an action for which the state to which it leads occurred in the largest number of EMs in which also the goal node occurred. In case that no such EM exists, heuristic steps such as choosing a random action, or selecting a random state that co-occurred with the goal in an EM as subgoal are useful.

We are describing for simplicity here only the simplest case, where one has first an exploration (learning) phase, and subsequently only carries out inference. Faster convergence to approximately optimal performance can be achieved if learning and inference are interwined, i.e., paths from arbitrary starts to arbitrary goals that are after a short exploration phase can be added to the reservoir of exploration trajectories, and be represented be index neurons for EMs. This is likely to speed up convergence since these initial goal-directed episodes are likely to contain fewer unnecessary detours than random walks.

2.2 Close to optimal performance for path planning in general graphs

The quality of a planning algorithm is commonly evaluated by the average length of paths that it produces for any pair of start and goal nodes in a general graph. We show in Fig. 2C, G that the EN-algorithm produces paths whose length is close to optimal, provided that it could carry out sufficiently many exploration episodes. This holds in spite of the fact that the EN-algorithm is an online planning method that produces one step at a time with low latency. Optimal path length can be computed by the offline Dijkstra algorithm. The required length of exploration episodes grows with the size of the graph (Fig. 2D, H).

An intriguing question is why this online planning method works so well, since each step has to be chosen with "foresight", i.e., depending on ensuing possibilities for reaching the goal. Fig. 2B, F show that this is made possible by the way in which relations between nodes are encoded in the geometry of the embedding $\mathbf{Q}$ of graph nodes into indices for EMs (see Fig. 1D). As in other types of cognitive maps (Behrens et al., 2018; Stöckl et al., 2024) the geometry of this representation of the graph nodes supports a simple heuristic for planning a path to a distal goal: Choose an action, i.e., an edge in the graph, that moves most directly into the direction of the goal. We commonly apply this heuristic for navigation in 2D environments. The representation of nodes via indices for episodic experiences that have visited this node enable us to apply this simple heuristic also for navigation in general graphs, in spite of the fact that they are in general not planar. The reason why one can do that is that the correlation between the binary vectors that represent two nodes (see Fig. 1D), or equivalently the cosine between these two vectors measures the number of EMs in which they co-occur. Hence, choosing a greedy action that maximally increases correlation with the goal state is obviously meaningful. This geometry of the binary vectors that represent states can best be visualized via projection onto a 3D sphere, see Fig. 2B, F. One sees clearly that it supports the previously described simple heuristic for choosing actions with fore-sight.

Fig. 2C, G also show that the performance of the basic EN-algorithm can be somewhat improved by a refined action selection strategy that aims at reaching next states that are connected by short paths with the goal. The basic EN-algorithm has already an inherent preference for choosing path segments to the goal that are present in the most EMs, thereby implicitly favoring shorter routes. The graded EN-algorithm employs a somewhat more sophisticated representation of EMs that is inspired by brain data. A neuron in area CA1 that provides an index for an EM does not respond equally strongly to all frames of that EM. Instead it often has a bell-shaped response profile for the sequence of frames that make up the EM. This is suggested on one hand by details of the BTSP rule, which is likely to be instrumental for the creation of these index neurons in area CA1, see the black curve in Fig. 3E of (Milstein et al., 2021). Also in the human brain the measured response to different images in a remembered sequence exhibits a bell-shaped profile, see Fig. 2 of (John et al., 2025). A simple way to mimic this biological detail is to let the neuron that encodes a sequence according to Fig. 1C respond stronger to items near the middle of the sequence. This implies that the nodes of the graph

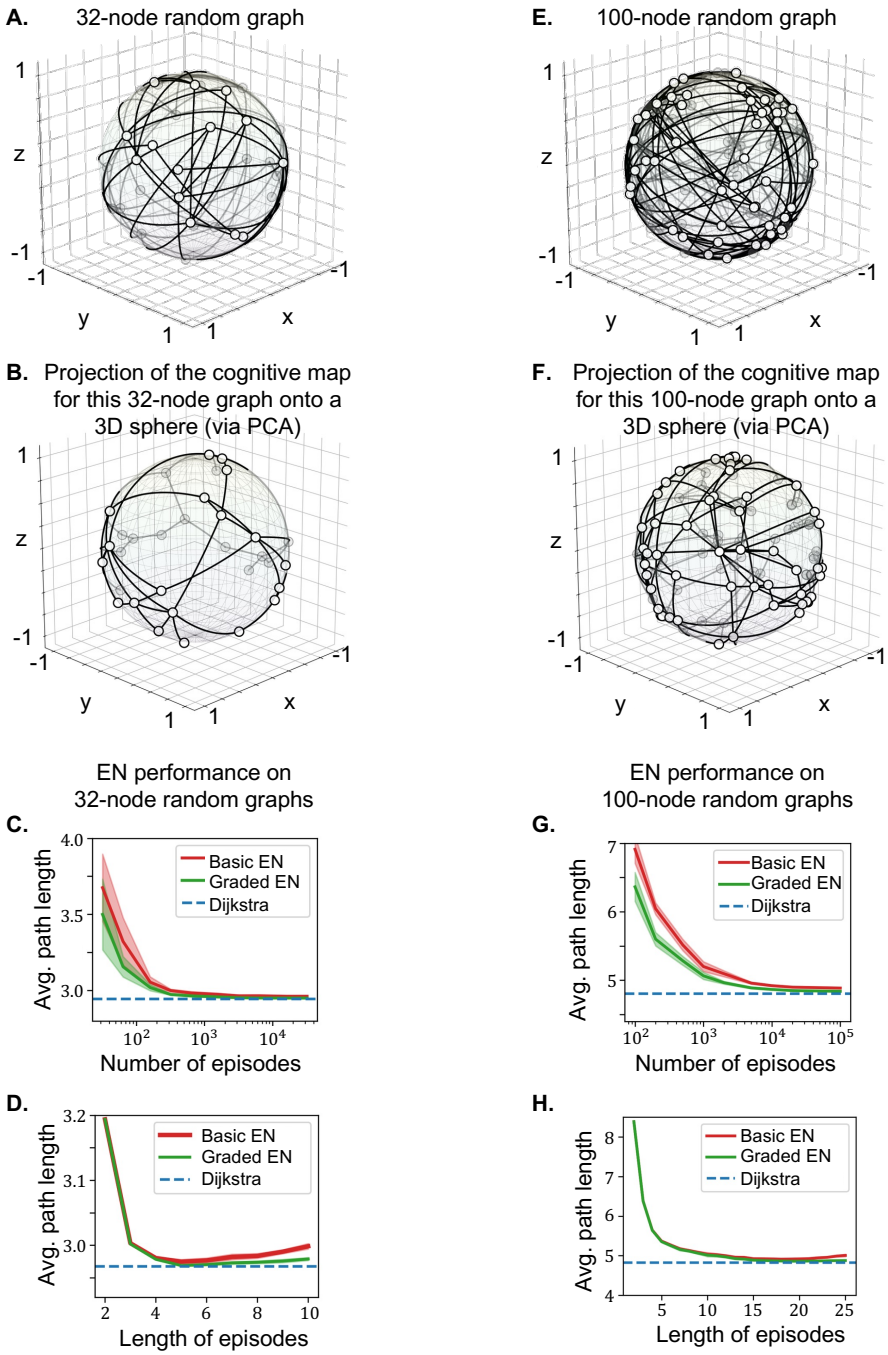


Fig. 2: **Performance and analysis of the EN-algorithm for planning in larger general graphs.** **(A)** A random graph with 32 nodes and edge probability 10%. **(B)** 3D projection of the cognitive map that was created through the embedding of the nodes of the graph into indices for EMs according to Fig. 1D. **(C)** Average path length for randomly chosen start and goal nodes produced by the EN-algorithm for this graph (red curve). Besides its basic version, we also depict the performance of a variant of the EN-algorithm (green curve) where neurons that provide indices for EMs respond stronger to items close to the middle of the episode ("Graded EN"). **(D)** Dependence of the average length of produced paths on the length of random walks (EMs) during exploration. **(E)** A random graph with 100 nodes and edge probability 3%. **(F - H)** Same as B - D but for this 100 node graph.

are mapped by the embedding $\mathbf{Q}$ to analog rather than binary vectors. Then the action selection mechanism of Fig. 1F prefers actions where the next state is connected by shorter episodes with the goal node, since these shorter episodes have a larger chance that either the next state or the goal

state is closer to the middle of the episode, thereby contributing more to the correlation between the corresponding vectors of analog values. The green curves in Fig. 2C, G show that the resulting "graded EN" algorithm improves planning performance in the case of relatively few exploration episodes, see Methods for details.

2.3 Context-aware and explainable planning

We show in the preceding and following sections that many types of planning and problem solving tasks that have been solved by RL algorithm can also be solved by the EN-algorithm. But besides its inherent flexibility to changes of the goal and its substantially lower computational complexity, the EN-algorithm enables simultaneous achievement of two important other benefits: Context-awareness of planning and explainability of plans. These benefits result from the fact that the EN-algorithm does not amalgamate all learnt knowledge into parameter values, but retains salient original experiences, including their context and specific advantages and disadvantages. These evaluations and annotations of EMs are carried out by several brain areas that are interconnected with area CA1, shown symbolically at the top of Fig. 3A. These internal evaluations of individual EMs are commonly used by our brain to selectively only recall particularly pleasant visits to a certain restaurant, or only those on rainy days, or only those where we were stuck with the bill. By adding such an evaluation module to our model, the EN-algorithm can selectively activate those EMs that contain the goal, but have in addition other properties that fit particularly well to the current planning context. This is shown in Fig. 3 for the simple graph from Fig. 1, where the 20 exploration episodes are colored differently in panel B, and partitioned arbitrarily into two classes ("contexts") according to their color, see panel D. Each of these contexts yields different cognitive maps, that are visualized in panels E and F via projections onto a 3D sphere. One sees that the task to plan a path from node 1 to node 2 yields in these two different context completely different solutions. The beauty is that this specialization of the cognitive maps to a given context does not require any computational overhead or new learning: One just has to constrain the activation of EMs by the current goal node to those EMs that fit into the desired context, as signaled by the evaluation module. Also, if the planning environment suddenly changes and some nodes are no longer available as transition points, e.g. if a road is blocked in a spatial navigation task, the EN-algorithm can inhibit on the fly those EMs which contain these nodes, as demonstrated in panels I, J, K of Fig. 3.

2.4 Context-aware and explainable planning in Markov decision processes (MDPs)

The EN-algorithm can also be applied if the outcomes of actions are uncertain, i.e., for Markov Decision Processes (MDPs), see Fig. 4A. To be precise, the standard definition of a MDP includes a specification of a reward function. Since the cognitive map of the stochastic graph which the EN-algorithm creates does not depend on any reward function, we can apply it already to pre-MDPs, which we define as MDPs without specification of a reward function. As in the deterministic case, the cognitive map for pre-MDPs which the EN-algorithm creates as data structure for learned experience facilitate inference, i.e., planning. Note that if there are actions with stochastic outcomes, the EN-algorithm has to produce for a given goal not just a single short path from the start to the goal, but a policy that provides an action selection for any node in the graph that might become a transition point. The quality of such a policy is commonly measured by the expected length of the resulting path to the goal.

Action selection in the presence of stochastic action outcomes requires even more sophisticated foresight than in the deterministic case, because a new tradeoff arises between short but uncertain sequences of actions selections in order to reach a given goal, and or longer but more reliable sequences. Fig. 4B illustrates this tradeoff for a hypothetical task where the goal is to get rich. In this case the longer but safer path resulting from studying and working hard has to be compared with the shorter but less certain path via purchase of a lottery ticket. The goal of action selection is to minimize the expected length of the path to the goal.

MDPs are in general asymmetric, i.e., actions can not necessarily be inverted. Hence we have to handle here as special case also the problem of navigation in directed deterministic graphs, where edges can in general only be traversed in one direction. Seemingly, this excludes the use of any cognitive map, since distances between states on a cognitive map form a metric and are therefore necessarily symmetric.

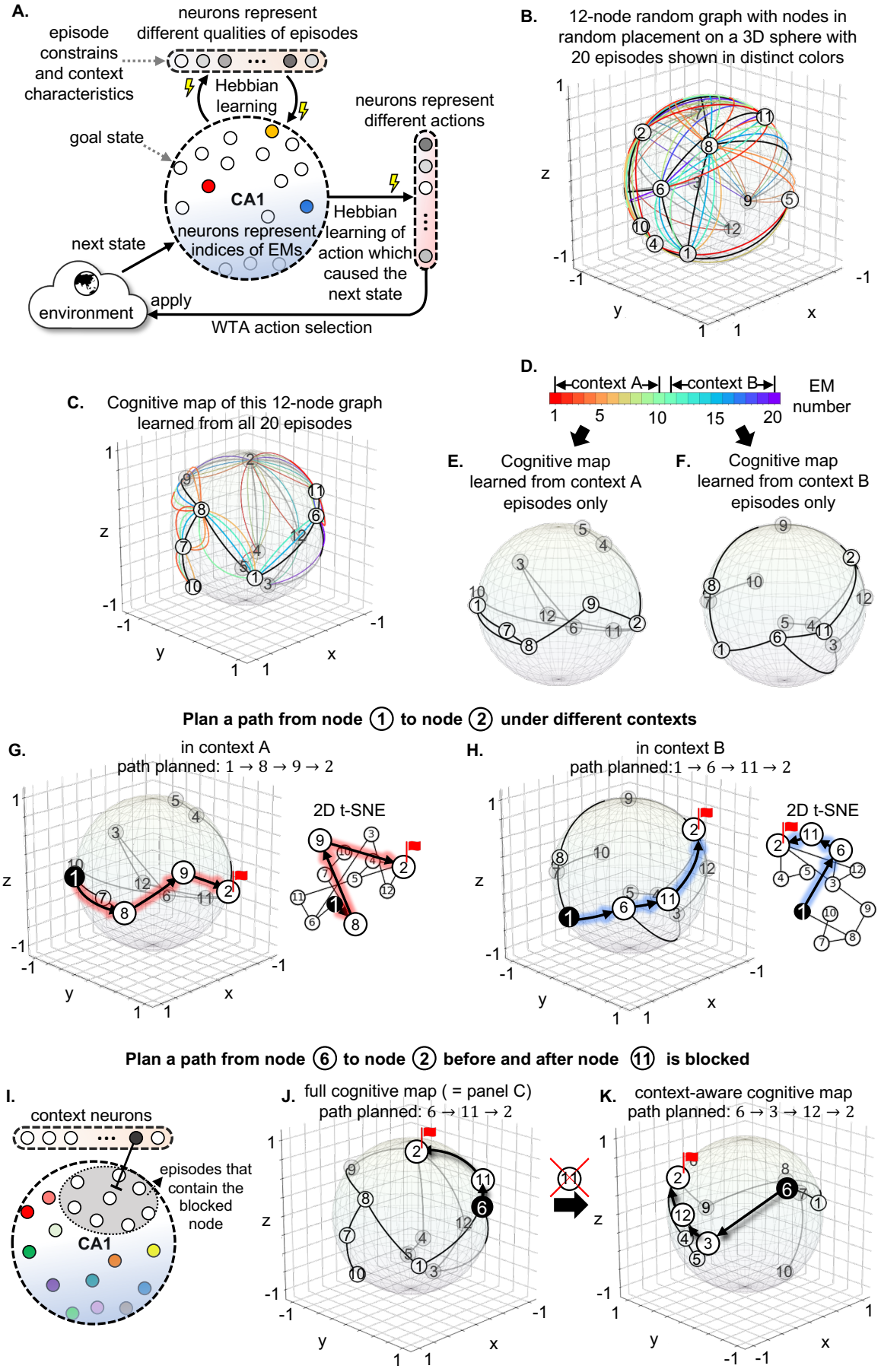

A.
episode constrains and context characteristics
neurons represent different qualities of episodes
Hebbian learning
goal state
neurons represent different actions
CA1
neurons represent indices of EMs
Hebbian learning of action which caused the next state
next state
environment
apply
WTA action selection
B.
12-node random graph with nodes in random placement on a 3D sphere with 20 episodes shown in distinct colors
C.
Cognitive map of this 12-node graph learned from all 20 episodes
D.
context A
context B
EM number
E.
Cognitive map learned from context A episodes only
F.
Cognitive map learned from context B episodes only
Plan a path from node 1 to node 2 under different contexts
G.
in context A
path planned: 1 → 8 → 9 → 2
2D t-SNE
H.
in context B
path planned:1 → 6 → 11 → 2
2D t-SNE
Plan a path from node 6 to node 2 before and after node 11 is blocked
I.
context neurons
episodes that contain the blocked node
CA1
J.
full cognitive map (= panel C)
path planned: 6 → 11 → 2
K.
context-aware cognitive map
path planned: 6 → 3 → 12 → 2

Fig. 3 *(previous page)*: **Context-aware planning through the EN-algorithm. (A)** Augmentation of the architecture through a network module at the top, which records the context of an EM and evaluates its quality from various perspectives. Synaptic connections from this network module to CA1 can selectively activate or inhibit EMs according to current context or constraints. **(B)** The same graph from Fig. 1A, with random node placement and all 20 exploration episodes shown in different colors. **(C)** An arbitrary partition of the 20 episodes into two contexts A and B. **(D)** The cognitive map for the graph from Fig. 1A, resulting from the embedding according to Fig. 1D. **(E)** 2D projection of the cognitive map resulting from the 10 episodes in context A. **(F)** Same for context B. **(G, H)** The same task, planning a path from node 1 to node 2, yields two different solutions in contexts A and B, indicated by projections of the cognitive maps for each context both for projections into 3D and 2D. **(I)** Another context (constraint): Node 11 can currently not be used for path planning, implemented by inhibiting the neurons in CA1 in which node 11 was occurring (these are the same neurons onto which node 11 is embedded according to Fig. 1D). **(J)** Task: Planning a path from node 6 to node 2. Note that the removed node 11 lies on the shortest path for this task. **(K)** Instantaneous re-planning with the updated cognitive map, where all EMs that contain node 11 are inhibited.

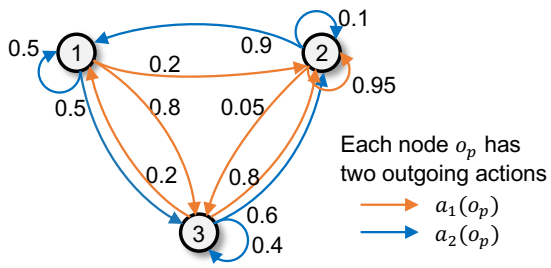
Hence geometric distances between two states on a cognitive map can not be indicative of the length of the shortest path from one to the other. But surprisingly, the same cognitive map as for undirected deterministic graphs does still provide a sense of direction to the goal, and therefore enables efficient action selection with foresight with the simple greedy heuristic ("move into the right direction"). We show that in the Supplement that this works for directed determnistic graphs, and consider here the general case of pre-MDPs with non-reversibel actions. Both the deterministic and the stochastic undirected case require a slight modification of the encoding of exploration episodes, since the direction of the sequence of nodes in an EM becomes now relevant. One easy way to handle this is to have a synaptic connection from a neuron $n(e)$ in CA1 to an action neuron n_a only if e is an episode that starts with this action a (from the current state). This implies that a given goal node o^* can only trigger selection of action a on a path to the goal via EMs that start with action a, and therefore go in the right direction, i.e., in the direction to
the goal node o^*. Fortunately, the action selection mechanism of Fig. 1F for the undirected case remains the same: WTA is applied to linear neurons n_a that represent possible actions a from the current node. This mechanism selects here that action a for which the empirical distribution of resulting next states has the largest correlation with the goal state. In other words, this mechanism chooses that action for which exploration episodes that start at this node have reached most often the goal node.

Note that this action selection mechanism is very easy to implement in a neural network, and can also be implemented through in-memory computing in a straightforward manner: by applying for each possible action at the current node a multiplication of the matrix $\mathbf{W}$ of synaptic weights with the vector that represents the goal state.

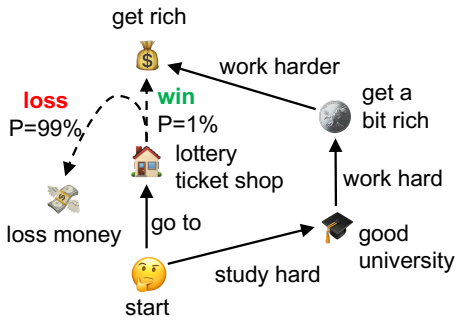
A gold standard for action selection in MDPs is more difficult to compute, since one cannot apply the Dijkstra algorithm in stochastic environments. But optimal policies can be computed via iterative RL methods such as value iteration (Sutton and Barto, 2018). We show in Fig. 4E that the EN-algorithm reaches the same performance level as value iteration. In addition it has the advantage that it produces already reasonable solutions with smaller number of computation steps as required for value iteration. This holds in spite of the fact that the EN-algorithm has to explore the graph on its own, whereas a perfect model of the MDP is provided to the value iteration algorithm (without any charge for the computing effort needed to produce that). In addition, value iteration has to be restarted from the beginning when the goal changes, whereas the EN-algorithm requires no extra computational effort for this adjustment. The Successor Representation (SR) variant of RL (Dayan, 1993) requires less computational effort for adjusting to a new goal, but according to Fig. 4E it requires overall more computational effort than the EN-algorithm or value iteration. The same experiment on a 32-node pre-MDP is shown in Fig. S2.

The EN-algorithm provides for the solution of path finding problems in MDPs two new qualities that have to the best of our knowledge not be achieved by any previous RL method: The search can instantly be adjusted to a given context, and each action selection can be explained in terms of prior experiences. We show this in Fig. 4G - I for the example of a pre-MDP in Fig. 4F, where each node has been assigned some probability for the occurrence of an accident when this node is passed. We assume that a new safety measure is enforced, where only paths to the goal are permitted that

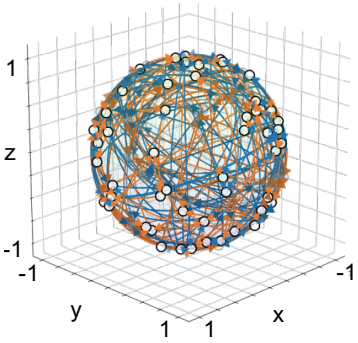
A. An example 3-node pre-MDP.



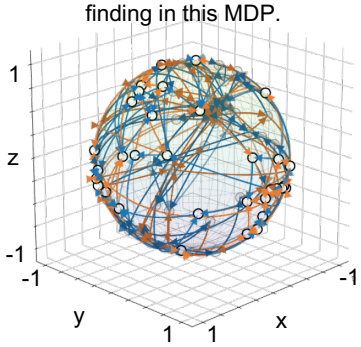
B. The shortest path is not always the best.



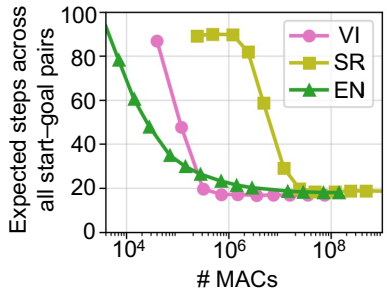
C. 100-node generic MDP with random node placement.



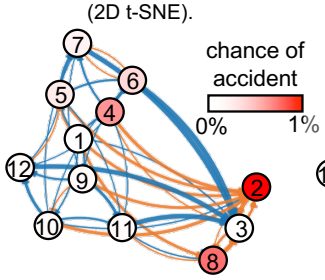
D. Cognitive map which the EN-algorithm employs for path finding in this MDP.



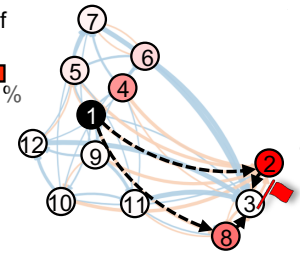
E. Performance comparison between the EN-algorithm and RL algorithm for this 100 node MDP.



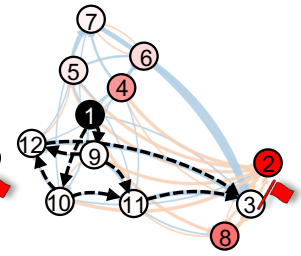
F. Cognitive map of a sample 12-node pre-MDP with arbitrary assignment of accident probabilities to nodes (2D t-SNE).



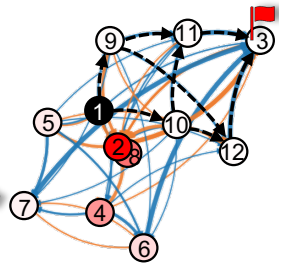
G. Paths from node 1 to 3 produced by the EN-algorithm without safety considerations.



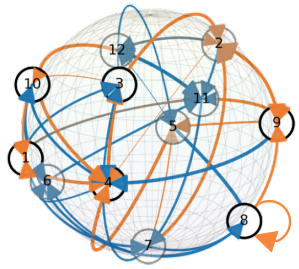
H. Paths from node 1 to 3 produced by the EN-algorithm with safety considerations.



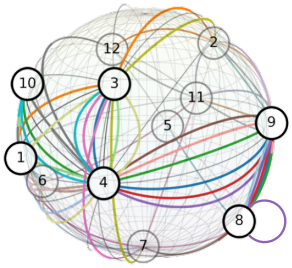
I. Cognitive map employed by the EN-algorithm for path finding with safety consideration.



J. A sample 12-node pre-MDP, with the two action options marked at each node



K. A set of 20 sample exploration episodes for this pre-MDP



L. Those 3 of the episodes that lead to the selection of the orange action at node 8

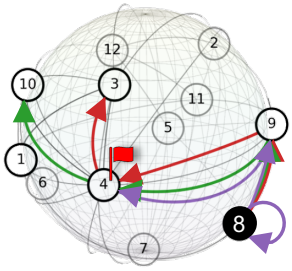


Fig. 4 *(previous page)*: **Context-aware and explainable planning of the EN-algorithm for MDPs. (A)** A sample 3-node pre-MDP with $A = 2$ actions (yellow, blue), each leading probabilistically to one of $K = 2$ successor states. A and K are fixed across examples, though EN generalizes to other settings. **(B)** The "get rich" example demonstrate the tradeoff between short but uncertain sequences of actions and longer but more reliable sequences. **(C)** A 100-node generic pre-MDP with random node placement. **(D)** Projection of the cognitive map of this 100-node pre-MDP onto a 3D sphere. **(E)** Comparison of value iteration (VI), successor representation (SR), and the EN-algorithm in terms of expected path length (averaged over 100 times path planning for each pair of start and goal nodes) as function of the total computational cost of the agent, measured by the number of multiply–accumulate (MAC) operations. **(F)** A 12-node pre-MDP. An arbitrary accident probability has been assigned to each node. **(G)** Paths from node 1 to node 3 generated by the policy resulting from the EN-algorithm. **(H)** Paths from node 1 to node 3 generated by the EN-algorithm in response to a requirement to avoid nodes with accident probability above 0.8%. **(I)** Cognitive map which the EN-algorithm employs for generating the action sequences (policy) indicated in panel H. It results from inhibiting all EMs that traversed nodes that were not sufficienlty safe.

traverse nodes whose accident probability is below some threshold (here chosen to be 0.8%). Fig. 4H depicts the paths from node 1 to node 3 that result from action sequences (or more precisely, the policy) which the EN-algorithm generates in view of this new safety measure. It can instantly adjust its planning to this new constraint by inhibiting indices for EMs that traversed nodes with higher accident probability, thereby switching instantly to the new cognitive map shown in Fig. 4I.

As for the deterministic case, also for MDPs each action selection of the EN-algorithm is supported by a concrete set of experiences (exploration episodes), which provide an explanation for its selection, see Fig. 4J - L for a visualization in the case of a small instance of the problem. The available set of EMs is depicted in Fig. 4K. At each node one has a choice between two actions, indicated by orange and blue arrows (each action with two possible outcomes). We consider in Fig. 4L the case that the orange action has been chosen at node 8, in order to reach node 4. This choice is supported by 3 prior experiences (exploration trajectories), whose colors are highlighted. Note that according to Fig. 1F each action selection is supported by a specific set of neurons in CA1 that provide indices for prior EMs, and an action selection can be explained in terms of those indices that are currently activated for a particular action selection and have non-zero weights to the winning neuron n_a (thereby contributing to its winning of the competition). This is valid also for stochastic action outcomes, where one has non-zero weights from all indices that define the next state for any of the possible outcomes of an action.

3 Discussion

Planning action sequence that are likely to trigger a sequence of events that satisfy our complex needs and wishes is an essential capability of our brain. It is also clear, that this is not only carried out through abstract brain computations on the basis of learnt probability tables, on which common RL algorithms are based. Furthermore, we are able to adjust our plans on the fly to new goals and contingencies, fit a plan to a given context, and recall previous experiences that provide a basis for our decisions. An understanding of these computational processes is needed both for modeling how brains plan and make decisions, but also in order to port similar capabilities into artificial agents. We show here that brain data on the omnipresence of EMs suggest an algorithm for decision making and planning that strongly differs from previously considered ones, see Fig. 1. This algorithm, that we call Episodic Neighbor (EN) algorithm because of its analogy to the famous k-Nearest Neighbor algorithm insofar that it does not compress learnt experiences completely into parameter values, but also maintains and uses learnt experiences in a more direct way. This turns out to be the key for achieving brain-like flexibility in planning, in particular the capability to tailor the plan to the current context, goals, and constraints, and also allows us to explain actions decisions and plans in the light of concrete previous experiences. But the EN-algorithm does not select and re-enact the closest experience, as the name "Episodic Neighbor" might suggests, but rather chooses the next action so that it is supported by the largest number of experiences that fit to the current goal and context.

RL-algorithms can be viewed as low-D projections of the substantially richer basis for planning which the EN-algorithm employs: In RL algorithms all experience is compressed into probability tables of values and probabilities. Apart from the resulting loss in flexibility and explainability, the

RL strategy to work with this numerical compression of rich experience, it also requires demanding computations for producing a decision on the next action, and a corresponding significant delay of this inference.

Also on the implementation level the EN-algorithm places more brain-like demands on the structure and use of neural networks that implement it, that can be seen as dual to the demands of common RL-methods. The EN-algorithm favors implementations through online-computations in shallow sparsely active neural networks, where neurons operate in an event-based sparse activity regime and produce binary messages ("spikes") instead of analog values. Its action selection requires, apart from a WTA operation, just a vector-matrix multiplication, and hence can be implemented in a single step on analog hardware with crossbars of memristors. Furthermore, the EN-algorithm only requires self-supervised online learning with simple local rules for synaptic plasticity, which can be implemented through on-chip plasticity of memristor arrays. The underlying sparse coding strategy of the EN-approach makes it especially suited for implementation on large neuromorphic systems, such as the ones that are currently emerging (Kudithipudi et al., 2025; Zhang et al., 2020). In Sec. D of the Supplement we are presenting an alternative implementation strategy of the EN-algorithm that is suitable for smaller memristor crossbars with a substantial number of conductance values of each memristor, as discussed in (Li et al., 2017).

In contrast to the EN-approach, RL algorithms require complex re-computations of a numerical value functions when a new experience is added, or when the goal changes. One reason is that for RL the value function and corresponding policies have to the updated in tandem, whereas the EN-algorithm can wait with the generation of policies until after learning, and only needs to generate policies step by step, in an online manner when decisions are needed.

We have shown in Fig. 2 that the EN-algorithm achieves approximates optimal performance for a planning environment that is commonly used for benchmarking planning algorithms (Russell, 2010): Finding short paths to a given goal nodes in a general undirected graph. In the Supplement (Fig. S1) we show that this also holds for directed graphs. In Fig. 4 we have shown that this also holds for planning in stochastic environments. The fundamental difference between the EN-algorithm and RL methods lies in the data structures which they employ for encoding learnt knowledge. We show that a rich repertoire of EMs can easily be organized into cognitive maps, which enable fast online action selection with foresight of the next action. Furthermore, we have shown in Fig. 3 that this cognitive map can be adjusted on the fly to reflect the current context. Furthermore, it remains clear which EMs support a specific action selection (see last panel of Fig. 4), thereby providing an explanation or interpretation of this action selection in the light of past experiences and their context.

An essential ingredient of the EN-algorithm are neurons that provide indices for EMs, i.e., which fire most of the time during recall of an EM. Such neurons had apparently first been found in the human brain by (Gelbard-Sagiv et al., 2008) for the case of EMs that were created by watching a movie clip. Related data from the human brain were reported by (John et al., 2025; Tacikowski et al., 2024). For the rodent brain experimental data on the emergence of neurons through BTSP that respond to a sequence of places on a track are closely related (Bittner et al., 2017; Wu and Maass, 2025). Characteristic for these indices or keys for EMs in the brain is that they respond to most members of the sequence that make up the EM, i.e., to a multitude of queries in the language of transformers (Vaswani et al., 2017). BTSP is a synaptic plasticity mechanism that has a several seconds long window of plasticity, and does therefore appear to be especially useful for creating neurons that fire during imprinting and recall of an EM. Essential for that is that neural codes for individual phases of the EM are encoded by sparse and largely orthogonal neural codes, as reported for the brain for example in (Sun et al., 2025). EMs are offline replayed during sharp wave ripples with a 10-fold speedup (Chen and Wilson, 2023), thereby moving also longer experiences into the plasticity window of a BTSP-like mechanism. A first application of BTSP in neuromorphic hardware has already been explored in (Galloni et al., 2024).

Whereas these neurons that fire during imprinting and recall of an EM were so far largely seen as by-product or indicator for the activation of an EM trace, the EN-algorithm suggests that they are likely to play an important functional role for planning and problem solving, since they enable an integration of knowledge from multiple related EMs through shallow neural networks. Further experimental studies of both the human and the rodent brain are needed in order to elucidate there functional role, for which this study provides a concrete hypothesis. In addition, more knowledge is needed about selective consolidation, replay, and other mechanisms that create neurons that fire throughout an extended experience. Also the relation between EMs and schemas that bundle several related EMs into a more abstract brain representation gains additional relevance from the functional

perspective of using both types of representation of learnt knowledge for planning and problem solving. We also want to clarify that neurons that represent indices of EMs are not the only representations of EMs. For example, other representations are needed to recall and replay individual frames of an EM. But so far the EN algorithm has no need for them.

A common feature of the EN-algorithm and transformers (Vaswani et al., 2017) is that both learn through self-supervised learning. Furthermore, both implement action selection by taking the maximum over dot products of target states $\mathbf{s}^* = \mathbf{Q}\mathbf{o}^*$ ("queries") with possible next states that result from currently available actions ("keys"). However, in contrast to transformers, the EN-algorithm can already plan well with few experiences, whereas transformers tend to require large sets of training data. Also the ease to produce context-aware and explainable decisions by the EN-algorithm can so far not be reproduced by LLMs because they amalgamate all learned knowledge into parameter values.

4 Methods

4.1 Mathematical description of the EN-algorithm

We consider a task space that is defined by some graph with N_o nodes. These nodes represent observations o , which may include external and internal signals in real-world applications, or clusters/classifications of such signals that represent clusters of functionally equivalent observations, e.g. all observations made at a particular spatial location. For simplicity, we represent each observation by a 1-hot code with an N_o-dimensional binary vector. One could just as well use sparse binary codes, which would correspond in the brain to a sparse activity pattern of neurons in areas CA3 and EC that are presynaptic to neurons in CA1 that encode indices for EMs (see (Wu and Maass, 2025) for references to biological data).

Learning

We assume that N_s exploration episodes have been been carried out in this task space. Each consists of a sequence of nodes, where each subsequent pair in the sequence is connected by an edge of the graph. One creates for each of these episodes e a linear neuron $n(e)$ that assumes non-zero values only for nodes on this trajectory, by setting the corresponding binary synaptic weights from neurons that are active in one of these nodes to neuron $n(e)$ to 1. This corresponds to a 1-shot application of the simplified BTSP rule from (Wu and Maass, 2025):

$$\mathbf{Q}_{e,o} = \begin{cases} 1 & \text{if node } o \text{ appears in episode } e, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Note that in the brain there is in general not just one but several neurons that fire for all phases of an episode, but it suffices for the EN-algorithm to have just one such neuron.

We then embed the N_o nodes of the graph by an embedding $\mathbf{Q} \in \{0,1\}^{N_s \times N_o}$ into the N_s-dimensional state space that is created by the N_s episode specific neurons $n(e)$. $\mathbf{Q}$ can be defined as a matrix that maps each node (observation) o, which is encoded by a 1-hot vector $\mathbf{o}$, onto the set of neurons $n(e)$ for which o occurs in the episode e. In particular, also a given target node o^* is mapped by $\mathbf{Q}$ to a target state $\mathbf{s}^* = \mathbf{Q}\mathbf{o}^*$ in the state space.

The only other synaptic plasticity that is needed is for learning a matrix $\mathbf{W} \in \{0,1\}^{N_a \times N_s}$ that maps states onto a population N_a of neurons n_a that each encode an action a in the sense that activation of n_a can trigger execution of action a. More precisely, the matrix $\mathbf{W}$ represents weights for synaptic connections from neurons $n(e)$ to neurons n_a that assume value 1 whenever e is an exploration episode in which the node occurs which results from execution of action a, i.e., after traversing the the edge that is encoded by a:

$$\mathbf{W}_{a,e} = \begin{cases} 1 & \text{if the node reached after executing action } a \text{ appears in episode } e, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The rule updates all incoming weights of neuron n_a in one-shot. From a biological perspective one can interpret learning of $\mathbf{W}$ as application of a Hebbian learning rule that enables neurons n_a to become mirror neurons, i.e., respond to an observation of the outcome that results from action a.

Planning

During planning an action sequence to reach the target state $\mathbf{s}^* = \mathbf{Q}\mathbf{o}^*$, the current utility value $u(a)$ of each action a, represented by a one-hot vector $\mathbf{a}$, is computed through a single-layer forward pass. Specifically, the goal state $\mathbf{s}^*$ is projected through the incoming weights ($\mathbf{Wa}$) of the ath action neuron n_a, and the resulting total input is given by the scalar product:

$$u(a) = (\mathbf{Wa})^\top \mathbf{s}^*. \tag{3}$$

Actions that cannot be executed in the current state are assumed to be inhibited (Stöckl et al., 2024), so that they cannot compete in the resulting WTA competition that chooses the action with largest current utility.

Context-awareness

We introduce a population of context neurons whose activations form the context vector $\mathbf{c} = (c_1, \ldots, c_{N_c})^\top$. Associations between context and index neurons for EMs are stored in bidirectional weight matrices $\mathbf{C}^{\text{CA1}\leftarrow\text{ctx}} \in \mathbb{R}^{N_s \times N_c}$ and $\mathbf{C}^{\text{ctx}\leftarrow\text{CA1}} \in \mathbb{R}^{N_c \times N_s}$, both initialized to zero.

During exploration, when an episode is stored, one can model this through BTSP, where exactly one CA1 neuron –indexed by i – receives a plateau potential (see (Wu and Maass, 2025). The symmetric Hebbian updates add contextual tags to this episode by strengthening only the connections involving i:

$$\Delta\mathbf{C}_{ij}^{\text{CA1}\leftarrow\text{ctx}} = \Delta\mathbf{C}_{ji}^{\text{ctx}\leftarrow\text{CA1}} = c_j. \tag{4}$$

Let $\mathbf{s}^* \in \{0,1\}^{N_s}$ denote the goal-related activation over episodes (before context selection). The desired context $\mathbf{c}$ gates $\mathbf{s}^*$ by elementwise multiplication ($\odot$) with the context-to-CA1 projection:

$$\tilde{\mathbf{s}}^* = \left(\mathbf{C}^{\text{CA1}\leftarrow\text{ctx}}\mathbf{c}\right) \odot \mathbf{s}^*. \tag{5}$$

The resulting plans remain explainable: the plan is based on real episodes that the agent collected before, and any selected episode can be traced to explicit, context tags via nonzero entries of $\mathbf{C}$.

The utility computation is the same as before, but uses the $\tilde{\mathbf{s}}^*$:

$$u(a) = (\mathbf{Wa})^\top \tilde{\mathbf{s}}^*. \tag{6}$$

4.2 Details to planning in general graphs in Fig. 2

We created random instances of deterministic graphs by specifying the number of nodes and a connection probability between any pair of nodes. The edge probability was set to 10% for the 32-node graph and to 3% for the 100-node graph.

For each deterministic graph, the EN-algorithm learns the environment from exploration episodes. We uniformly sample an equal number of random walks of length L ($L = 5$ in Fig. 2C, and $L = 16$ in Fig. 2G) starting from each node. For example, in a 32-node graph, sampling 10 trajectories per node results in $N_s = 32 \times 10 = 320$ episodes ($N_s = 3.2 \times 10^4$ in Fig. 2D, $N_s = 10^5$ in Fig. 2H). At each step of a trajectory, the agent randomly selects one of the current node's outgoing edges with equal probability and moves to the next node. A trajectory of length L contains L edges, corresponding to $L + 1$ nodes. Some nodes may appear more than once within an episode; however, we only record whether a node occurs (1) or not (0). Multiple occurrences of the same node within a trajectory do not affect its embedding.

As a baseline comparison, we employ the Dijkstra graph search algorithm. Dijkstra's algorithm is an offline planning method that requires unrolling the entire planning sequence before executing the first move. It generates the shortest path between any given start node and goal node.

Both matrices $\mathbf{Q}$ and $\mathbf{W}$ are initialized to zeros. During planning, an affordance gating mechanism is applied prior to the WTA process to inhibit actions that cannot be executed (see (Stöckl et al., 2024) for details).

4.2.1 Graded EN

To model the case where episodic neurons respond more strongly to nodes occurring near the center of a trajectory, we introduce a graded activation profile along each episode. For an episode e of length

L, let $t \in \{1, \ldots, L\}$ denote the temporal index of a node within the episode. The activation weight $g(t)$ assigned to that node is defined by a Gaussian function centered at the middle of the trajectory:

$$g(t) = \exp\left(-\frac{(t - L/2)^2}{2\sigma^2}\right), \tag{7}$$

where the standard deviation σ controls the width of the bell shape. In our implementation, we set $4\sigma = L$, so that the Gaussian effectively spans the entire episode length, and automatically adjusts when the trajectory length L updates.

Each node occurrence within an episode contributes to its embedding in $\mathbf{Q}$ proportionally to $g(t)$ rather than as a binary indicator:

$$\mathbf{Q}(i, p) = \begin{cases} g(t) & \text{if node } \mathbf{o}_p \text{ appears in episode } e \text{ at time } t, \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

Thus, nodes near the middle of an episode receive higher activation values, while nodes at the beginning or end contribute less. This graded mapping leads to a continuous-valued $\mathbf{Q}$ matrix and improves planning performance when only a limited number of exploration episodes are available (see Fig. 2C, G), and when the trajectory length is long (see Fig. 2D, H).

4.2.2 3D Spherical PCA Projection

To visualize the relational geometry of node embeddings, i.e, of cognitive maps, we applied a 3D Spherical PCA projection. PCA is widely used for embedding visualization (Jolliffe, 2011), and L2-normalization, as in word2vec, makes cosine similarity reflect angular distance (Mikolov et al., 2013).

Specifically, the high-dimensional embedding matrix $\mathbf{Q}$ was first projected onto its first three principal components using Principal Component Analysis (PCA). The resulting three-dimensional vectors $\mathbf{x}_i$ were then normalized to unit length: $\tilde{\mathbf{x}}_i = \mathbf{x}_i / \|\mathbf{x}_i\|_2$, so that only their directional (angular) information was preserved. This normalization maps all nodes onto the surface of a unit sphere, enabling angular distance (or equivalently, cosine similarity) to reflect the correlation structure among nodes. Intuitively, nodes that frequently co-occur within the same episode exhibit more similar embedding directions, resulting in smaller angular separations on the sphere.

4.3 Pre-Markov Decision Processes (pre-MDPs)

We model environments as discrete stochastic graphs that correspond to Markov decision processes (MDPs) without a reward function, here referred to as pre-MDPs. Such structures describe only the stochastic state transitions that are caused by actions, and serve as platforms on which different goal-directed behaviors can later be planned when given a goal.

Each pre-MDP consists of N_o nodes (states) and A actions per node, resulting in $N_a = N_o \times A$ distinct actions.

Each node o is represented by a one-hot observation vector $\mathbf{o} \in \{0, 1\}^{N_o}$ here only the oth element is 1 and all others are 0s. Similarly, each action a is also defined by a one-hot action vector $\mathbf{a} \in \{0, 1\}^{N_a}$ with the ath element set to 1 and all others are set to 0s.

These definitions specify the stochastic environment used by all algorithms in the study. Since no reward function is assigned, the same learned transition structure can be reused for planning toward any desired goal.

We have created generic pre-MDPs for our experiments in the following way: For each action a among N_a actions, we randomly select K successor nodes o_i among the N_o nodes. Transition probabilities to these K successors are assigned by sampling unnormalized weights $w_i \sim \mathcal{U}(0, 1)$ and normalizing:

$$P(o_i \mid a) = \frac{w_i}{\sum_{j=1}^{K} w_j}. \tag{9}$$

Carrying out this procedure for all actions yields a sparse transition tensor $P \in \mathbb{R}^{N_o \times N_a}$ that defines a probabilistic graph.

4.3.1 Mathematical description of the EN-algorithm for pre-MDPs

For each episode i, we update the state embedding matrix $\mathbf{Q} \in \{0,1\}^{N_s \times N_o}$ by the same rule as Eq. (1) as for deterministic graphs.

The $\mathbf{W} \in \{0,1\}^{N_a \times N_s}$ matrix is updated by having a synaptic connection from a neuron $n(e)$ in CA1 that represents an EM e to an action neuron n_a only if $n(e)$ is the index for an EM e that starts with this action a_q:

$$\mathbf{W}(a,e) = \begin{cases} 1 & \text{if action } a \text{ initiated episode } e, \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

The action selection mechanism remains the same as in the deterministic case. The utility value $u(a)$ of each action a is computed through a single-layer forward pass. Specifically, the goal state $\mathbf{s}^* = \mathbf{Q}\mathbf{o}^*$ activates all episodic neurons $n(e)$ whose corresponding episodes e contain the node o^*. This goal representation is then projected through the incoming weights ($\mathbf{Wa}$) of the action neuron n_a. Here $\mathbf{a}$ is the one-hot vector of action a. The incoming weights of n_a indicate which episodes were initiated by a. The resulting total input is given by the scalar product: $u(a) = (\mathbf{Wa})^\top \mathbf{s}^*$. Then, a WTA mechanism applied to the linear neurons n_a, which represent possible actions a from the current node, selects the action a for which the exploration episodes initiated by this action have most frequently reached the goal node.

4.4 Details to planning in pre-MDPs in Fig. 4

We created random pre-MDPs with N_o nodes, where N_o is specified for each experiment. We allow $A = 2$ actions per node. Each action leads to $K = 2$ possible next nodes. A and K are fixed across examples, though EN generalizes to other settings.

Episodes were generated by random walk exploration in the MDP. Each episode starts from a uniformly sampled node $s^0 \in \{1, \ldots, N_o\}$ and runs for a fixed length L. At each time step t, an action $a^t \in \{1, \ldots, A\}$ is chosen uniformly at random, and the next state s^{t+1} is drawn from $P(\cdot|s^t, a^t)$. This procedure ensures that, in expectation, each action is applied equally often at every node, introducing no sampling bias when later comparing which action most frequently lead the agent toward the goal.

4.4.1 Implementation of RL-algorithms for comparison

Value iteration

Our implementation is a direct transcription of the pseudo code in (Sutton and Barto, 2018), Section 4.4 ("Value Iteration"), which iteratively updates the state-value function according to the Bellman optimality equation:

$$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)\big[r(s,a,s') + \gamma V_k(s')\big]. \tag{11}$$

We used a reward convention that assigns $r(s,a,s') = -1$ for each non-goal transition and $r(s,a,s') = 0$ when entering the goal state, so that the optimal value function corresponds to the negative of the expected number of steps required to reach the goal. The discount factor was set to $\gamma = 0.99$, which maintains numerical stability while effectively approximating an undiscounted shortest-path objective over long planning horizons. The algorithm loops over all states in each iteration, assuming that the complete transition probability matrix $\mathbf{P}$ is known in advance. Iterations are repeated for a fixed number of sweeps K, and the greedy policy is extracted after each update as

$$\pi(s) = \arg\max_a \sum_{s'} P(s'|s,a)\big[r(s,a,s') + \gamma V(s')\big]. \tag{12}$$

For goal-conditioned planning, $V(s)$ is recomputed from scratch for each new goal state. Because it explicitly accesses $\mathbf{P}$, value iteration has a computational advantage over model-free methods such as the Successor Representation and the EN-algorithm that must estimate transition statistics from experience.

Successor representation

For the Successor Representation (SR) baseline, we followed the original temporal-difference formulation introduced by Dayan (1993). The SR matrix $\mathbf{M}$ encodes the expected discounted future state occupancies as:

$$\mathbf{M} = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{o}_{p_t} \middle| \mathbf{o}_{p_0}\right], \tag{13}$$

where p_t denotes the index of the state visited at time t, and $\mathbf{o}_{p_t}$ is its corresponding one-hot vector. Thus, each row $\mathbf{M}(\mathbf{o}_p)$ represents the expected discounted visitation frequencies of all future states when starting from state p. The matrix satisfies the fixed-point relation $\mathbf{M} = \mathbf{I} + \gamma P_\pi \mathbf{M}$, where P_π is the transition matrix under policy π.

We implemented the standard on-line one-step temporal-difference (TD) learning rule:

$$\mathbf{M}(\mathbf{o}_p) \leftarrow \mathbf{M}(\mathbf{o}_p) + \big[\mathbf{o}_p + \gamma \mathbf{M}(\mathbf{o}_{p'}) - \mathbf{M}(\mathbf{o}_p)\big], \tag{14}$$

where $\mathbf{o}_p$ and $\mathbf{o}_{p'}$ correspond to the one-hot encodings of the current and next states, respectively. The behavior policy during training was uniformly random over actions to ensure sufficient exploration.

After convergence, planning toward a goal state was performed by defining a goal-specific reward vector $\mathbf{r}$ (with $r_p = 0$ at the goal and -1 elsewhere) and computing the corresponding value vector:

$$\mathbf{v} = \mathbf{M}\mathbf{r}. \tag{15}$$

A greedy one-step look-ahead policy was then derived from $\mathbf{v}$ using the same procedure as in the VI baseline. This implementation corresponds exactly to the canonical SR learning and planning framework of Dayan (1993).

4.4.2 Multiply–accumulate (MAC)-Based Estimates of the Computational Complexity of Planning Algorithms

The dominant arithmetic operation in neural and neuromorphic learning systems is the multiply–accumulate (MAC) operation (Sze et al., 2020), which computes weighted sums $\sum_i w_i x_i$. Counting MACs provides a simple and hardware-independent measure of computational cost. Each MAC performs one multiplication and one addition, typically fused into a single hardware instruction (FMA), where one FMA equals two floating-point operations (FLOPs) (NVIDIA Developer, 2023). Hence, reported MAC counts can be directly converted to FLOPs if desired.

For Value Iteration (VI), one training sweep updates the value of every state once according to the Bellman optimality equation:

$$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)\big[r(s,a,s') + \gamma V_k(s')\big]. \tag{16}$$

Each state update iterates over all A possible actions, and for each action computes the expected future value $\sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V_k(s')]$, which forms a dot product between the transition vector $P(s'|s,a)$ and the term $[r(s,a,s') + \gamma V_k(s')]$. Assuming that each (s,a) pair has on average k nonzero successor states (with $k = 2$ in our experiments), the total computational cost is:

$$(\text{MACs})_{\text{VI}} \approx (N_o A k) \text{ per sweep over every state,} \tag{17}$$

reflecting all Bellman updates across states, given full access to the transition matrix $\mathbf{P}$.

We assumed in Fig. 4E that a perfect model $\mathbf{P}$ of the pre-MDP is known and can be used by VI. If VI also has to explore the pre-MXDP, like the other two algorithms, additional computations are required to update the estimate of the model during exploration. For N_s nodes and A actions, applying each action 10^3 times to obtain reliable transition probabilities yields $N_s A \times 10^3$ parameter updates. As in EN, each parameter write is counted as one effective MAC for parity, introducing extra computational overhead to VI. Moreover, learning from an estimated $\mathbf{P}$ may cause deviations from the theoretical optimal policy.

For the Successor Representation (SR), each transition applies the one step TD update:

$$\mathbf{M}[s,:] \leftarrow \mathbf{M}[s,:] + \alpha\big(\mathbf{e}_s + \gamma \mathbf{M}[s',:] - \mathbf{M}[s,:]\big). \tag{18}$$

This involves two vector-scalar multiplications over all N_o elements: one for computing $\gamma \mathbf{M}[s', :]$ and another for $\alpha(\cdot)$ during the weight update, resulting in:

$$(\text{MACs})_{\text{SR}} \approx (2N_o) \text{ per step the agent moves,} \tag{19}$$

per environment transition. When trained on an episode with length L, the total $(\text{MACs})_{\text{SR}} \approx 2N_oL$ per episode.

For the learning process of an EN agent, each episode updates a single row of the $\mathbf{Q}$ matrix that represents the incoming weights of a memory neuron. The update marks all unique nodes which occurred in that trajectory. The update simply writes binary values to mark whether a node appeared in the episode, without any arithmetic computation.

For comparability with SR and VI, we count each memory write in EN as one effective MAC. This provides a conservative estimate, since a single MAC internally performs three memory reads and one memory write, making it more computationally demanding than EN's binary updates. Using this convention places EN on the same computational scale as the arithmetic-based algorithms while slightly overestimating its true cost. Hence, the compute cost per episode for learning the $\mathbf{Q}$ matrix is approximated by the number of unique nodes visited, approximately proportional to the trajectory length $L + 1$. In addition, EN needs to mark an episode's initiating action by updating matrix $\mathbf{W}$ according to Eq. (10), which contributes to 1 additional MAC per episode. In total, the computation cost of EN is:

$$(\text{MACs})_{\text{EN}} \approx (L + 2) \text{ per episode.} \tag{20}$$

In experiments, this value is recorded statistically for each episode during learning and added to the MAC counter.

4.4.3 Other detailed settings

The cognitive map in Fig. 4D was trained with $N_s = 10^5$ episodes of length $L = 10$. To evaluate the expected planning path length in Fig. 4E, we averaged results over 100 planning runs for each possible $(\text{start}, \text{goal})$ pair.

In the 12-node pre-MDP example shown in Fig. 4F–I, the accident probability of each node was sampled from $\text{safety}_i \sim 0.1\% + 0.9\% \times \text{Beta}(a, b)$, with $(a, b) = (1.2, 8)$ for all $i \notin 2, 8$, where $\text{Beta}(a, b)$ is a Beta distribution. The probabilities of nodes 2 and 8 were fixed at 1%, making them two low-safety nodes. The cognitive map was generated from 1000 random-walk episodes, each of length $L = 3$. The cognitive map of the pre-MDP example in Fig. 4J–L was generated from 20 random-walk episodes with the same episode length $L = 3$.

5 Acknowledgments

We would like to think Christoph Stoeckl for inspiring discussions. This research was partially supported by the National Science Foundation of the USA (EFRI BRAID project 2318152) and the Austrian Science Fund (FWF) (10.55776/COE12). The authors also gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer JUWELS[1] at Jülich Supercomputing Centre (JSC).

References

Behrens, T.E., T.H. Muller, J.C. Whittington, S. Mark, A.B. Baram, K.L. Stachenfeld, and Z. Kurth-Nelson. 2018. What is a cognitive map? organizing knowledge for flexible behavior. *Neuron 100*(2): 490–509 .

Bekkemoen, Y. 2024. Explainable reinforcement learning (xrl): a systematic literature review and taxonomy. *Machine Learning 113*(1): 355–441 .

Bittner, K.C., A.D. Milstein, C. Grienberger, S. Romani, and J.C. Magee. 2017. Behavioral time scale synaptic plasticity underlies ca1 place fields. *Science 357*(6355): 1033–1036 .

Bonini, L., C. Rotunno, E. Arcuri, and V. Gallese. 2022. Mirror neurons 30 years later: implications and applications. *Trends in cognitive sciences 26*(9): 767–781 .

Bottini, R. and C.F. Doeller. 2020. Knowledge across reference frames: Cognitive maps and image spaces. *Trends in Cognitive Sciences 24*(8): 606–619 .

Burkart, N. and M.F. Huber. 2021. A survey on the explainability of supervised machine learning. *Journal of Artificial Intelligence Research* 70: 245–317 .

Buzsáki, G. 2019. *The brain from inside out.* Oxford University Press.

Chen, Z.S. and M.A. Wilson. 2023. How our understanding of memory replay evolves. *Journal of Neurophysiology 129*(3): 552–580 .

Dayan, P. 1993. Improving generalization for temporal difference learning: The successor representation. *Neural computation 5*(4): 613–624 .

Elston, T.W. and J.D. Wallis. 2025. Context-dependent decision-making in the primate hippocampal–prefrontal circuit. *Nature Neuroscience 28*(2): 374–382 .

Fried, I. 2022. Neurons as will and representation. *Nature Reviews Neuroscience 23*(2): 104–114 .

Galloni, A.R., Y. Yuan, M. Zhu, H. Yu, R.S. Bisht, C.T.M. Wu, C. Grienberger, S. Ramanathan, and A.D. Milstein. 2024. Neuromorphic one-shot learning utilizing a phase-transition material. *Proceedings of the National Academy of Sciences 121*(17): e2318362121 .

Gelbard-Sagiv, H., R. Mukamel, M. Harel, R. Malach, and I. Fried. 2008. Internally generated reactivation of single neurons in human hippocampus during free recall. *Science 322*(5898): 96–101 .

John, T., Y. Zhou, A. Aljishi, B. Rieck, N.B. Turk-Browne, and E.C. Damisah. 2025. Representation of visual sequences in the tuning and topology of neuronal activity in the human hippocampus. *bioRxiv*: 2025–03 .

Jolliffe, I. 2011. Principal component analysis, *International encyclopedia of statistical science*, 1094–1096. Springer.

Kolibius, L.D., F. Roux, G. Parish, M. Ter Wal, M. Van Der Plas, R. Chelvarajah, V. Sawlani, D.T. Rollings, J.D. Lang, S. Gollwitzer, et al. 2023. Hippocampal neurons code individual episodic memories in humans. *Nature human behaviour 7*(11): 1968–1979 .

Kudithipudi, D., C. Schuman, C.M. Vineyard, T. Pandit, C. Merkel, R. Kubendran, J.B. Aimone, G. Orchard, C. Mayr, R. Benosman, et al. 2025. Neuromorphic computing at scale. *Nature 637*(8047): 801–812 .

Li, C., D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang, et al. 2018. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nature communications 9*(1): 2385 .

Li, C., M. Hu, Y. Li, H. Jiang, N. Ge, E. Montgomery, J. Zhang, W. Song, N. Dávila, C.E. Graves, et al. 2017. Analogue signal and image processing with large memristor crossbars. *Nature electronics 1*(1): 52–59 .

Mattar, M.G. and M. Lengyel. 2022. Planning in the brain. *Neuron* .

Mikolov, T., K. Chen, G. Corrado, and J. Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* .

Milstein, A.D., Y. Li, K.C. Bittner, C. Grienberger, I. Soltesz, J.C. Magee, and S. Romani. 2021. Bidirectional synaptic plasticity rapidly modifies hippocampal representations. *Elife* 10: e73046 .

Mukamel, R., A.D. Ekstrom, J. Kaplan, M. Iacoboni, and I. Fried. 2010. Single-neuron responses in humans during execution and observation of actions. *Current biology 20*(8): 750–756 .

NVIDIA Developer. 2023. Matrix multiplication background user's guide. Technical documentation.

Perez-Cerrolaza, J., J. Abella, M. Borg, C. Donzella, J. Cerquides, F.J. Cazorla, C. Englund, M. Tauber, G. Nikolakopoulos, and J.L. Flores. 2024. Artificial intelligence for safety-critical systems in industrial and transportation domains: A survey. *ACM Computing Surveys 56*(7): 1–40 .

Rawas, S. 2024. Ai: the future of humanity. *Discover artificial intelligence 4*(1): 25 .

Russell, P.N. 2010. Artificial intelligence: a modern approach by stuart. *Russell and Peter Norvig contributing writers, Ernest Davis...[et al.]* .

Russell, S. and P. Norvig. 2020. Artificial intelligence: A modern approach, 4th edition. *Pearson Series in Artifical Intelligence* .

Stöckl, C., Y. Yang, and W. Maass. 2024. Local prediction-learning in high-dimensional spaces enables neural networks to plan. *Nature Communications 15*(1): 2344 .

Sun, W., J. Winnubst, M. Natrajan, C. Lai, K. Kajikawa, A. Bast, M. Michaelos, R. Gattoni, C. Stringer, D. Flickinger, et al. 2025. Learning produces an orthogonalized state machine in the hippocampus. *Nature 640*(8057): 165–175 .

Sutton, R.S. and A.G. Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

Sze, V., Y.H. Chen, T.J. Yang, and J.S. Emer. 2020. How to evaluate deep neural network processors: Tops/w (alone) considered harmful. *IEEE Solid-State Circuits Magazine 12*(3): 28–41 .

Tacikowski, P., G. Kalender, D. Ciliberti, and I. Fried. 2024. Human hippocampal and entorhinal neurons encode the temporal structure of experience. *Nature 635*(8037): 160–167 .

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, and I. Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 .

Waydo, S., A. Kraskov, R.Q. Quiroga, I. Fried, and C. Koch. 2006. Sparse representation in the human medial temporal lobe. *Journal of Neuroscience 26*(40): 10232–10234 .

Wu, Y. and W. Maass. 2025. A simple model for behavioral time scale synaptic plasticity (btsp) provides content addressable memory with binary synapses and one-shot learning. *Nature communications 16*(1): 342 .

Yang, Y. and W. Maass. 2025. A parsimonious model for learning order relations provides a principled explanation of diverse experimental data. *bioRxiv*: 2025–03 .

Zhang, Y., P. Qu, Y. Ji, W. Zhang, G. Gao, G. Wang, S. Song, G. Li, W. Chen, W. Zheng, et al. 2020. A system hierarchy for brain-inspired computing. *Nature 586*(7829): 378–384 .

Supplementary Information for

Episodic memories enable powerful algorithms for goal-directed decision making that are context-aware and explainable

Yukun Yang and Wolfgang Maass
Institute of Theoretical Computer Science, Graz University of Technology
yukun.yang@tugraz.at, wolfgang.maass@tugraz.at

Table of Contents

A. Non-orthogonal observations 21
B. Directed graphs 22
Figure S1. Performance of EN for planning in directed graphs. ..... page 22
C. MDPs 23
Figure S2. Comparison of VI, SR, and EN on a 32-node pre-MDP. ..... page 23
D. Variation of the model suitable for memristor implementation 23

A Non-orthogonal observations

In abstract graph experiments, each node is encoded as a one-hot observation vector, such that all observations are orthogonal with each other. However, this is not a strict requirement. Especially in biological brain, the sensory information are largely non-orthogonal. In this section, we explain how our model can tolerant non-orthogonal inputs.

A.1 Non-orthogonal code generation

On the 32-nodes abstract graph environment, we generated for each node a 10^4-dimensional binary vector with sparsity $= 0.5\%$ following the data 0.2% to 1% measured in the human medial temporal lobe (Fried, 2022; Waydo et al., 2006).

The degree of orthogonality is adjustable: The indices of ones are adjusted either to overlap with the dimensions of other items' ones, increasing the average cosine similarity, or to shift overlapping ones to different dimensions, reducing the cosine similarity. For a given target average cosine similarity, this process is iterated for a sufficiently long time to achieve a numerically optimal coding configuration.

A.2 Training

The training still follows the same rule, except for the inputs are no longer one-hot vectors.

A trajectory contains L observations. All observation neurons that fire within a trajectory will induce a LTP on the incoming weights of a state neuron (with index i) that is selected by the plateau potential:

$$(\mathbf{Q})_i = \bigvee_{j=1}^{L} \mathbf{o}_j \quad \text{if observation } \mathbf{o}_j \in \text{ the } i^{\text{th}} \text{ trajectory.} \tag{21}$$

In above, the $\bigvee$ represents dimension-wise "OR" operation between binary vectors: the result on a dimension is one if any observation $\mathbf{o}_j$ has a one on this dimension.

A.3 State embedding

Non-orthogonality poses a challenge for our model. When an observation lies outside of a trajectory but shares some "1" entries with an in-trajectory observation, the off-trajectory node still activates the corresponding state neuron—and that neuron ideally shouldn't respond. This undesired activation injects noise into the model.

Crucially, if only a subset of an observation's ones overlaps with another, then the state neurons encoding the true trajectory observation should exhibit higher activations than those encoding the spurious overlap. To enforce this, we introduce a nonlinearity $f(\cdot)$ and replace the original state-embedding formula $\mathbf{s} = \mathbf{Q} \cdot \mathbf{o}$ with:

$$\mathbf{s} = f(\mathbf{Q} \cdot \mathbf{o}). \tag{22}$$

Specifically, we use a max-thresholding function that preserves only the maximal entries (zeroing out all others). This effectively suppresses the unwanted activations and eliminates noise. One can implement this also with a suitable global inhibition.

A.4 Result

All other parts are identical to the original rule. In experiments, the model can tolerant any non-orthogonal codes, as long as no two observations share the same code.

With the average pairwise cosine between observations = 0.2 (highly non-orthogonal as compared with data from human MTL in 0.02 to 0.08 (Yang and Maass, 2025)), and trained on $N_s = 1000$ trajectories with each contains $L = 10$ nodes. The EN agent on average takes 3.114 steps on a 32-node random graph, only 1.81% worse than Dijkstra's 3.058 steps.

B Directed graphs

Directed graph is a special case of a pre-MDP where the outcome probabilities of all actions are 0 or 1, and importantly, for which one can still compute the gold standard shortest path via Dijkstra.

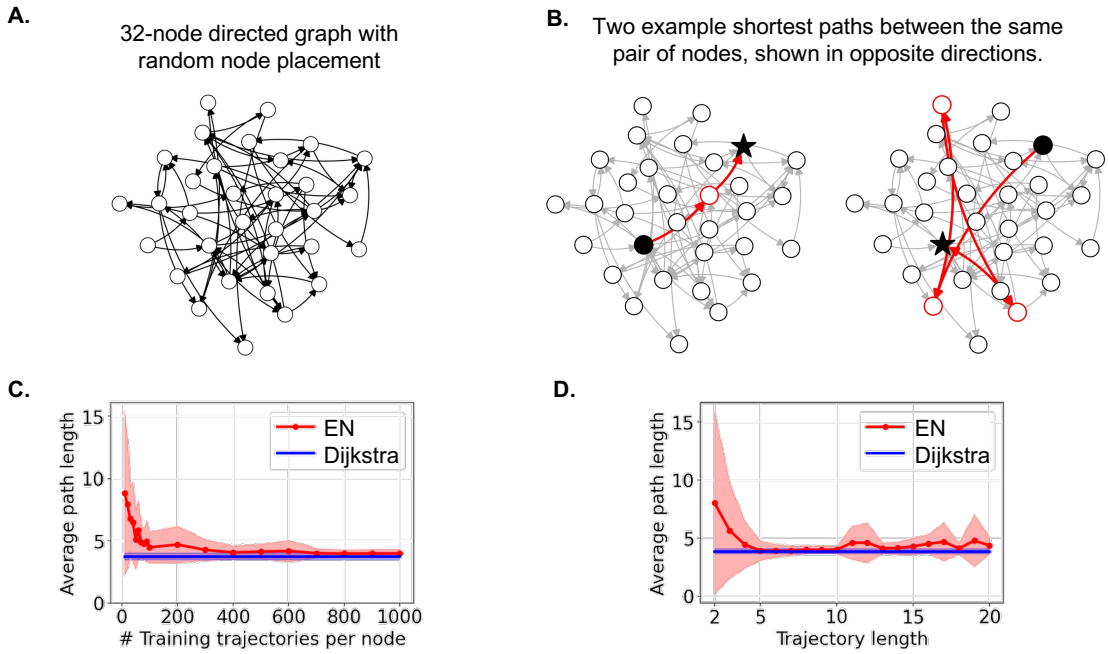


Fig. S1: **Performance of EN for planning in directed graphs. A.** An example random directed graph. **B.** Two example shortest paths for a pair of nodes in opposite directions. The trajectories can be very different when switching start and goal nodes. **C.** and **D.** show mean ± standard deviation over 10 randomly generaetd directed graphs. In D, one sees that the model already has near-optimal performance when the agent is trained on $\tilde{4}00$ episodes per node (with trajectory length = 6). In E, one sees that the model has a very stable near-optimal performance for trajectory length ranging from 5 to 10 (with 1,000 trajectories per node).

Across 10 random directed graphs, our model's average path length is **3.903**—only 4.51% longer than Dijkstra's 3.734—using 1000 training trajectories per node (length 6).

C MDPs

We repeat the experiment in Fig. 4C-E on a 32-node MDP. One can see from Fig. S2C that the required computation is roughly an order of magnitude smaller for the 32-node pre-MDP compared with that in the 100-node pre-MDP in Fig. 4E.

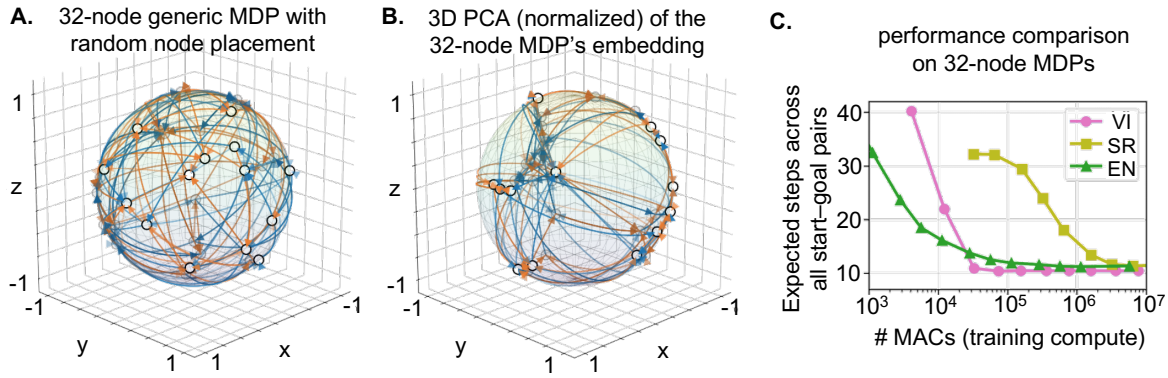


Fig. S2: **Comparison of VI, SR, and EN on a 32-node pre-MDP. (A)** A 32-node generic pre-MDP with random node placement. **(B)** Projection of the cognitive map of this 32-node pre-MDP onto a 3D sphere. **(C)** Comparison of value iteration (VI), successor representation (SR), and the EN-algorithm in terms of expected path length (averaged over 100 times path planning for each pair of start and goal nodes) as function of the total computational cost of the agent, measured by the number of multiply–accumulate (MAC) operations.

D Variation of the model that is suitable for implementation on smaller arrays of memristors that have many conductance levels

The previously described model can be implemented with memristors that have just 2 conductance levels. But there are also memristors that can assume many different conductance levels, see (Li et al., 2017). A remaining bottleneck for implementation on currently existing neuromorphic hardware is then the maximal size of the memristor crossbar array. There and in (Li et al., 2018) a 64 x 128 array was used. The dimensions of the binary weight matrix that was learnt in our preceding model were determined by the number of nodes in a graph task (or observations in the general case), and the number of trajectories during exploration. The latter was in general substantially larger, see Fig. 3G.

We describe here an alternative implementation that can be learnt by a smaller quadratic memristor crossbar whose dimension is determined just by the number of nodes. In other words, $N_s = N_o$, and the matrix $\mathbf{Q}$ becomes square. Importantly, only positive weights are required. Furthermore, one can expect that performance decays gracefully when these weights are not precisely learnt.

We describe this directly for the more challenging task of learning navigation in a directed graph. Obviously, the navigation problem in arbitrary directed graphs with 64 node is a really challenging task, but can in principle be learnt with the memristor crossbars described in (Li et al., 2018, 2017).

We propose the following learning rule: for a given observed trajectory of observations (nodes), we consider all pairs $(\mathbf{o}_i, \mathbf{o}_j)$ such that $\mathbf{o}_i$ occurs before $\mathbf{o}_j$ in the sequence. Each such pair triggers a weight update:

$$\mathbf{Q}'_{ij} \leftarrow \mathbf{Q}_{ij} + 1 \quad \text{if node } i \text{ occurs before } j \text{ in a trajectory.} \tag{23}$$

In this formulation, the input $\mathbf{o}_i$ opens the plasticity window for the ith state neuron, and all subsequent inputs $\mathbf{o}_j$ (with $j > i$) induce LTP (long-term potentiation).

During inference or planning, we simply use all values from the column of the goal node in the state embedding matrix—selected by a one-hot observation vector—as a value function: $\mathbf{g} = \mathbf{Q}\mathbf{o}^*$, representing $V_{\mathbf{g}}(\mathbf{s})$. A greedy policy for reaching a goal $\mathbf{g}$ moves from any current state $\mathbf{s}_i$ to the adjacent state $\mathbf{s}_j$ for which $(\mathbf{g})_j$ is maximal among all available neighbors of $\mathbf{s}_i$.

One may notice that the proposed rule is closely related to the learning rule of SR, in that it also counts how often the agent moves from one node to another during exploration. However, there are some differences that appear to be important for neuromorphic applications:

The learning process does not require a "learning rate", unlike the Successor Representation learned via temporal difference (TD). In other words, weights are stored as natural numbers, and there is minimal requirement on precision. No discount factor $\gamma \in [0, 1]$ is used. The effect of discounting is instead approximated by limiting the trajectory length (i.e., a window size), effectively truncating continuous trajectories into overlapping segments. A longer window corresponds to a higher effective discount factor.