Episodic memories make goal directed action selection context-aware and explainable

(January 8, 2026)

Yukun Yang and Wolfgang Maass

Institute of Machine Learning and Neural Computation, Graz University of Technology, Graz, 8010, Austria.

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

Abstract

Current AI tools are criticized for their lack of context-awareness and explainability. This makes their action selection sometimes undesirable for a given context, or even unsafe. In contrast, our brains can adjust their goal-directed action selection on the fly to the current context, and we can explain our action selection in terms of related experiences. This capability results from a fundamental difference between the way how brains and current AI tools store experiences. Whereas most AI tools transform experiences into parameter values, the brain also stores explicit representations in the form of episodic memories, and uses them for goal-directed action selection. We show that experimental data on neural codes for episodic memories in the brain suggest a method that allows us to port these capabilities into artificial devices. The resulting Episodic Neighbor Algorithm attains high task performance by creating a cognitive map from episodic memories, which provides a clear and explainable sense of direction for goal-directed action selection. Furthermore, this cognitive map can be reconfigured on the fly if the goal or context change, or when new experience is added. The resulting algorithm for goal-directed action selection requires only local rules for synaptic plasticity in shallow neural networks, and is therefore suitable for implementation in energy-efficient edge devices.

1 Introduction

The best current AI methods for goal-directed action selection are based on Deep Neural Networks (DNNs) or Large Language Models (LLMs). Suboptimal aspects of these tools is the large amount of training data which they require, their large power consumption, and their long latency in decision making. In addition, it is hard to quickly adjust them when the goal or contingencies change, or to optimize an action sequence for a specific context, such as driving when it snows, or taking a walk at night. In addition, their lack of explainability 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 for making autonomous driving and human-robot interaction safer. Substantial research efforts have already been invested into Explainable Reinforcement Learning (XRL) (Bekkemoen, 2024; Milani et al., 2024; Qing et al., 2022). But they have not yet solved the problem of making Reinforcement Learning (RL) explainable, partially because there is a lack of tools for intrinsically explainable RL.

Apparently the brain has found better solutions, because when we solve a problem we can usually adjust our strategy to the current context, and explain our action selection on the basis of concrete experiences. But the algorithms and data structures which the brain uses for that have remained opaque (Mattar and Lengyel, 2022). Concrete experiences are stored in the brain in the form of episodic memories (EMs). But how can algorithms make efficient use of a multitude of diverse EMs?

An interesting hint from experimental data are neurons that serve as indices for salient EMs, which fire during all phases of an episodic experience. For example, one has found neurons that fire throughout the presentation of a specific movie clip, both while encountering this episodic experience and during subsequent recall (Chettih et al., 2024; Gelbard-Sagiv et al., 2008; John et al., 2025; Kolibius et al., 2023; Purandare and Mehta, 2023; Tacikowski et al., 2024; Yaeger et al., 2025). These neural codes, which correspond to keys in LLMs, are complementary to other neural codes that encode the content (values) of EMs and their sequential structure. But it has remained open how they can be used for goal-directed decision making.

We show that this becomes possible if one considers a specific type of state-encoding and cognitive map which they generate. Cognitive maps have already been studied in many contexts in neuroscience and cognitive science. Cognitive maps of the brain enable navigation not only in spatial environments but also in abstract concept spaces (Behrens et al., 2018; Bottini and Doeller, 2020). It turns out that EMs give rise to a new form of cognitive map that provides a general solution to the problem of goal directed decision making. In contrast to previous approaches, such as reinforcement learning with a successor representation (Stachenfeld et al., 2017), a state is represented in our EM-based approach by the vector of indices for EMs in which this state occurred. This representation is inspired by the observation that if we think for example of a concrete restaurant, we immediately activate many EMs of visits to this restaurant. These EMs help us to plan a visit to it in a context-aware manner. For example, they suggest to first make a reservation if we want to visit it on a saturday, because this was useful in the past. Also, we can usually explain the decision to order a specific dish on the basis of EMs of previous visits where we liked this dish. Our results show that one can turn these intuitive ideas into a powerful algorithm for goal-directed decision making that captures these functional advantages of the use of EMs by the brain, i.e., context-awareness and explainability. Furthermore, we show that this algorithm is very flexible and can adjust on the fly to new goals, contingencies, and experiences.

Learning an index for a new EM requires a one-shot learning rule for synaptic plasticity. A rule which achieves that, BTSP (Behavioral Time Scale Synaptic Plasticity), had first been discovered in area CA1 of the rodent hippocampus (Bittner et al., 2017). This is the same brain area where indices for EMs were found both in the rodent and in the human brain (John et al., 2025; Kolibius et al., 2023; Zutshi et al., 2025). Very recently, BTSP has also been demonstrated in the neocortex of awake and behaving rodents (Xiao et al., 2025; Yaeger et al., 2025). The seconds-long integration window and fast learning of temporal contexts that is shown 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 the human brain. We show here that the simple model for BTSP from (Wu and Maass, 2025) suffices for creating a cognitive map from individual EMs that supports context-aware and explainable action selection. This demonstrates the validity of the conjecture from (Durstewitz et al., 2025) that BTSP provides new solutions to continual learning challenges in dynamically changing environments. The Episodic Neighbor Algorithm (ENA) for goal-directed action selection that we propose requires in addition to learning of indices for EMs only some standard method for learning predictions of possible consequences of an action. The name Episodic Neighbor Algorithm was chosen because it shares an important feature with the famous k Nearest Neighbor (k-NN) algorithm: It does not compress all learned knowledge into parameter values, but stores exemplars of salient experiences. However, in virtually all other aspects this algorithm is very different from the k-NN algorithm. For example, it operates in dynamic rather than static environment, and it employs self-supervised learning instead of supervised learning.

The ENA 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 goal can be seen as analogue to a query, and the content (observations) of an EM as values. Furthermore, both LLMs and the ENA apply some form of WTA (Winner-Take-All) for producing an output. But in contrast to LLMs, the ENA neither requires deep learning nor large amounts of data, and can be implemented in an energy-efficient sparse activity regime. Furthermore, the ENA offers in comparison with RL approaches both significant functional advantages in terms of explainability and context-awareness and more energy-efficient implementation options. In fact, it appears to be especially suited for implementation through on-chip plasticity and energy-efficient computations with sparse activity in neuromorphic chips (Kudithipudi et al., 2025; Zhang et al., 2020).

We will first explain the basic principles of the ENA in a generic setting for goal-directed action selection and problem solving: finding a short path from some start to some goal node in an abstract

graph. The ENA does not require any prior knowledge about the graph. It explores it instead in a self-supervised manner and records these experiences in EMs. We show in the subsequent section that it achieves close to optimal performance on fairly large random graphs. Subsequently we show that this also holds for stochastic graphs, i.e. for Markov Decision Processes (MDPs), where action outcomes are not deterministic and actions are in general not reversible. Finally we show that the ENA produces efficient foraging strategies when there are multiple goals, and that it can easily be transferred to continuous task domains.

2 Results

2.1 The embedding of observations into episodic memories provides a new representation of a state

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 goal directed action selection 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 underlying goal-directed decision making opaque for the human user.

We introduce here a radically different approach for the definition of a state in terms of the EMs in which it occurs. More precisely, the ENA 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 standard problem solving task (Russell and Norvig, 2020), finding a short path to a given goal node in 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. We show in Section A of the Supplement that it suffices in the end if different inputs to these neurons are approximately orthogonal.

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 ENA 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 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 producing 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

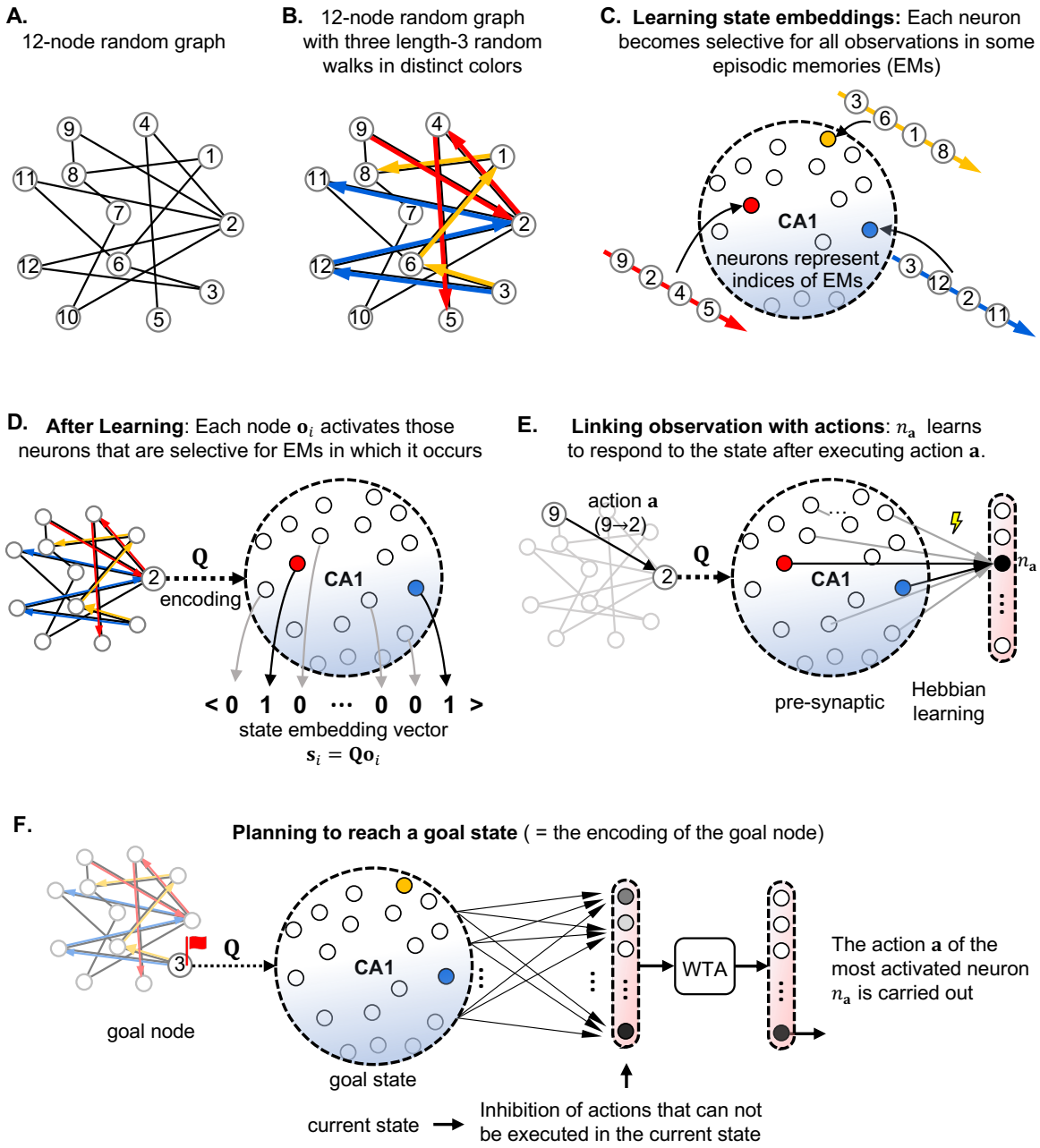


Fig. 1: **Network- and learning-architecture for the ENA. (A)** A small random graph, which serves as running example for this figure. **(B)** Three sample exploration trajectories indicated by three colors. **(C)** Population of neurons, corresponding to area CA1 in the brain, where each neuron becomes 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 ENA for online action selection 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.

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 ENA 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.

Finally, we want to point out that there exists an alternative method for goal directed action selection with the help of EMs. If one encodes an EM not just as sequence of nodes that the corresponding trajectories visits, but includes the actions (edge traversals) that it employs on the way, a simpler method for action selection is supported that does not require learning of synaptic weights from a representation of the next state after execution of an action, as indicated in Fig. 1E. Instead, a synaptic weight from an index neuron for an EM e to a neuron n_a that encodes an action a can be set to 1 if action a occurred in the corresponding trajectory. This corresponds in the brain to learning of backwards connections from index neurons in CA1 to neurons in the neocortex that encode content of the corresponding EM. This is assumed to take place during replay (Chen and Wilson, 2023). Goal directed action selection can then be carried out through these learned synaptic weights with the same method as indicated in Fig. 1F.

2.2 Cognitive maps resulting from episodic memories support a simple heuristic for reaching any goal

The quality of an action selection 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 ENA 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 ENA 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 producing 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 ENA 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 ENA 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 ENA

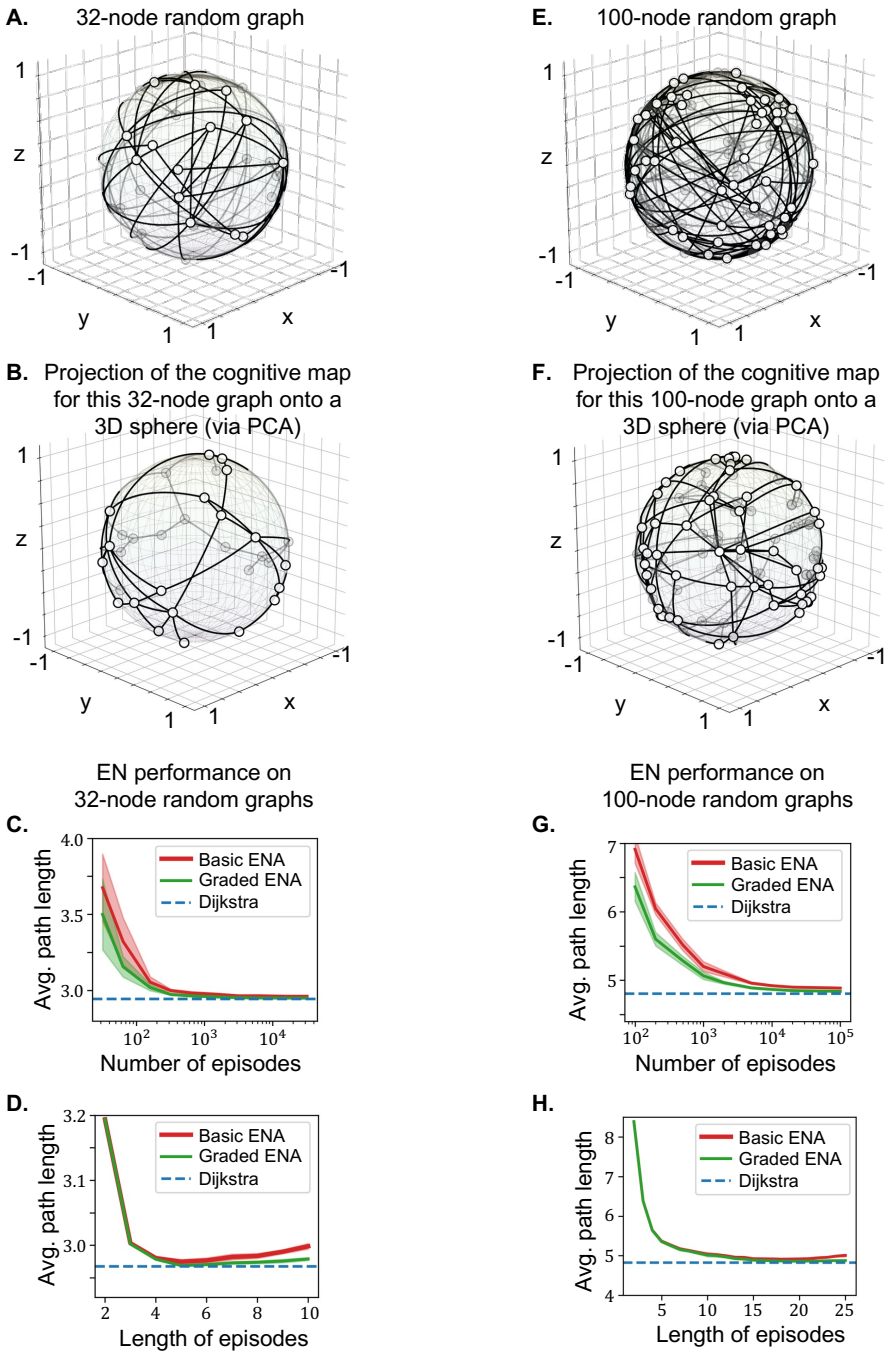


Fig. 2: **Performance and analysis of the ENA for action selection in larger general graphs.** **(A)** A random graph with 32 nodes and edge probability 10%. **(B)** 3D projection of the cognitive map that was created through the embedding of the nodes of the graph into indices for EMs according to Fig. 1D. **(C)** Average path length for randomly chosen start and goal nodes produced by the ENA for this graph (red curve). Besides its basic version, we also depict the performance of a variant of the ENA (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.

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 are mapped by the embedding **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 performance in the case of relatively few exploration episodes, see Methods for details.

2.3 The ENA enables context-aware and explainable decision making

We show in the preceding and following sections that many types of problem solving tasks that have been solved by RL algorithm can also be solved by the ENA. But besides its inherent flexibility to changes of the goal and its substantially lower computational complexity, the ENA enables simultaneous achievement of two important other benefits: Context-awareness of action selection and explainability of plans. These benefits result from the fact that the ENA 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 ENA can selectively activate those EMs that contain the goal, but have in addition other properties that fit particularly well to the current problem solving 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 produce 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 task 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 ENA 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 action selection in Markov decision processes (MDPs)

The ENA 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 ENA 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. In (Liu et al., 2024) the term controlled MDP was used for that, but this term would be rather un-intuitive in our context.

As in the deterministic case, the cognitive map for pre-MDPs which the ENA creates as data structure for learned experience facilitate inference, i.e., action selection. Note that if there are actions with stochastic outcomes, the ENA 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

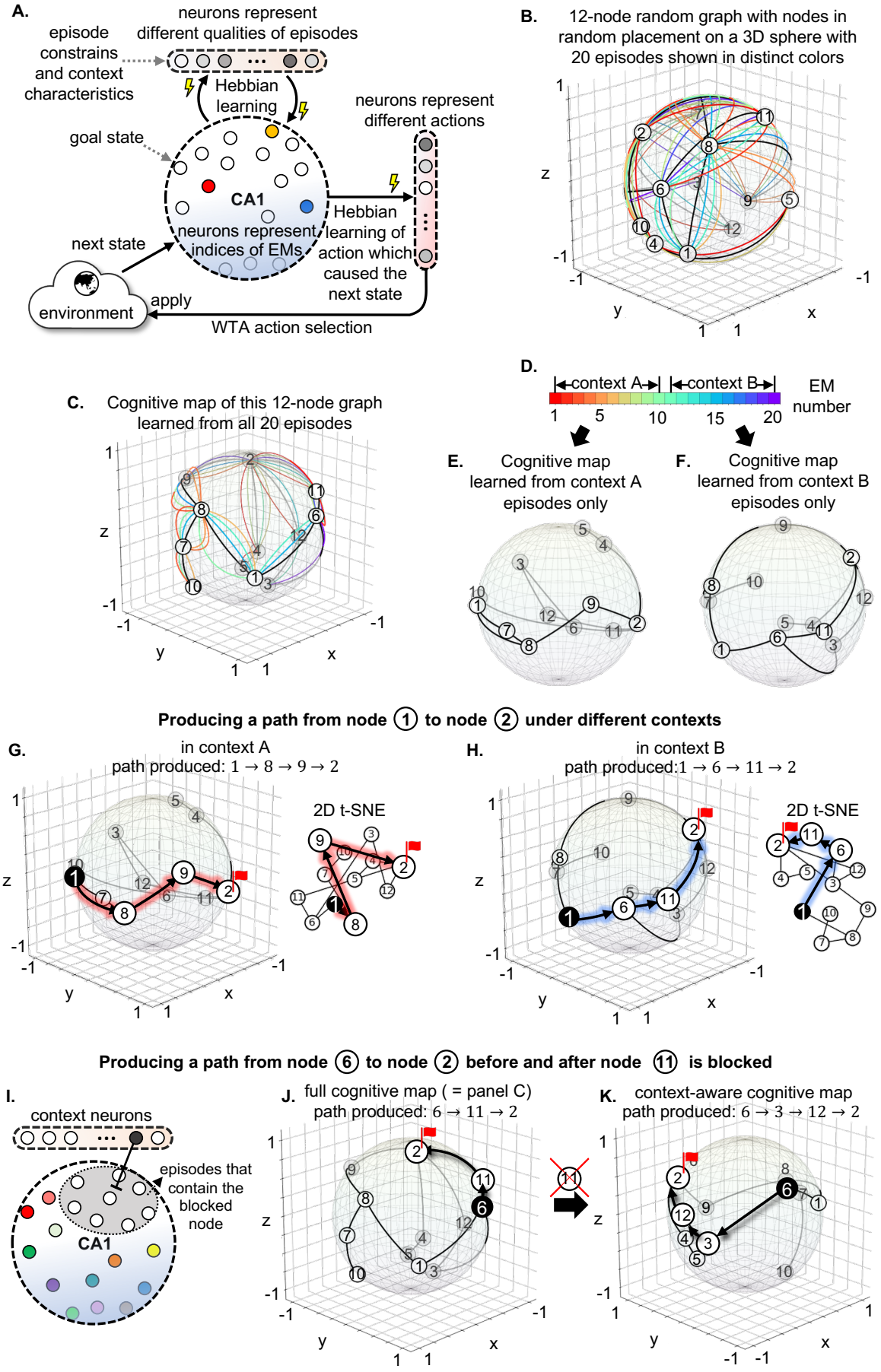

A.
episode constrains and context characteristics
neurons represent different qualities of episodes
Hebbian learning
goal state
neurons represent different actions
CA1
neurons represent indices of EMs
Hebbian learning of action which caused the next state
next state
environment
apply
WTA action selection
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
Producing a path from node 1 to node 2 under different contexts
G.
in context A
path produced: 1 → 8 → 9 → 2
2D t-SNE
H.
in context B
path produced:1 → 6 → 11 → 2
2D t-SNE
Producing 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 produced: 6 → 11 → 2
K.
context-aware cognitive map
path produced: 6 → 3 → 12 → 2

Fig. 3 *(previous page)*: **Context-aware action selection through the ENA. (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, producing 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 in a path, implemented by inhibiting the neurons in CA1 that represend indices for EMs in which node 11 was occurs (these are the same neurons onto which node 11 is embedded according to Fig. 1D). **(J)** Task: Producing a path from node 6 to node 2. Note that the removed node 11 lies on the shortest path for this task. **(K)** Instantaneous update of the cognitive map, where all EMs that contain node 11 are inhibited.

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.

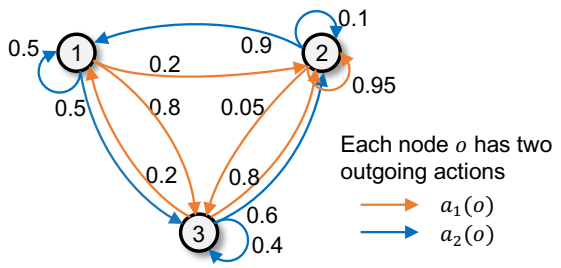
MDPs are in general asymmetric, i.e., actions can not necessarily be inverted. Hence we have to handle here as special case also the problem of navigation in directed deterministic graphs, where edges can in general only be traversed in one direction. Seemingly, this excludes the use of any cognitive map, since distances between states on a cognitive map form a metric and are therefore necessarily symmetric. 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 provides a sense of direction to the goal, and therefore enables efficient action selection with foresight with the simple greedy heuristic ("move into the direction of the goal"). But both deterministic directed graphs and pre-MDPs require a slight modification of the encoding of exploration episodes, since the direction of the sequence of nodes in an EM becomes now relevant. We describe in Methods how this can be handled. We discuss in Section B of the Supplement an application to the special case of deterministic directed graphs

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 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.

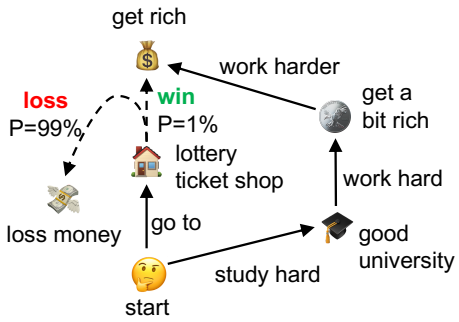
Note that this action selection mechanism is very easy to implement in a neural network, and can also be implemented through in-memory computing in a straightforward manner: by applying for each possible action at the current node a multiplication of the matrix $\mathbf{W}$ of synaptic weights with the 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 ENA 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 ENA 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 ENA 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 ENA or value iteration. Results of the same experiment for a 32-node pre-MDP are shown in Fig. S2.

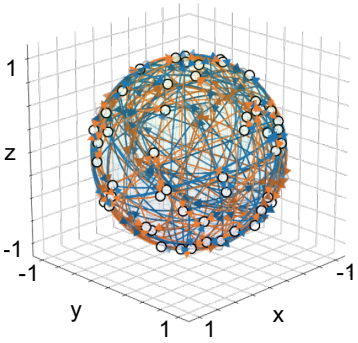
A. An example 3-node pre-MDP.



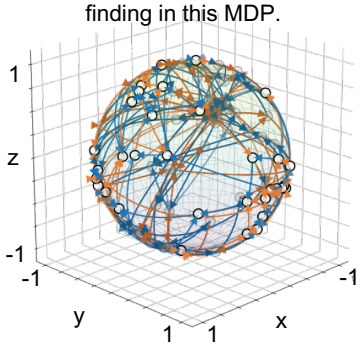
B. The shortest path is not always the best.



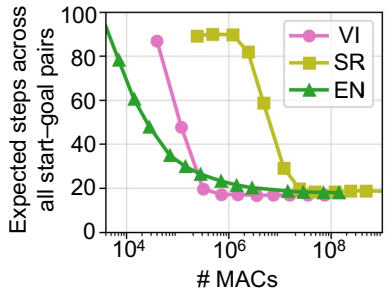
C. 100-node generic MDP with random node placement.



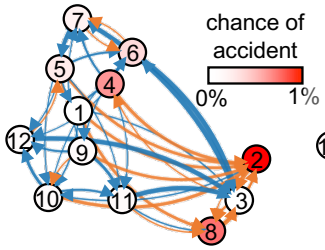
D. Cognitive map which the EN-algorithm employs for path finding in this MDP.



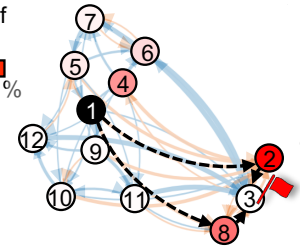
E. Performance comparison between the EN-algorithm and RL algorithm for this 100 node MDP.



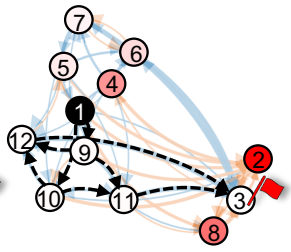
F. Cognitive map of a sample 12-node pre-MDP with arbitrary assignment of accident probabilities to nodes (2D t-SNE).



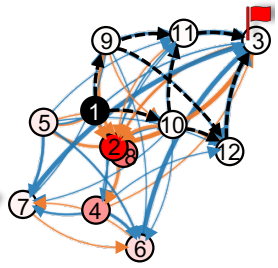
G. Paths from node 1 to 3 produced by the EN-algorithm without safety considerations.



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

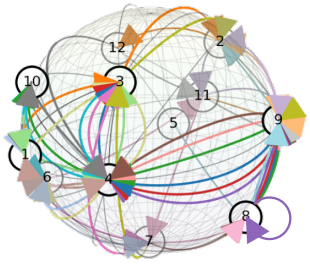


I. Cognitive map employed by the EN-algorithm for path finding with safety consideration.



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

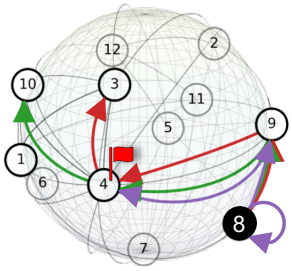


Fig. 4 *(previous page)*: **Context-aware and explainable action selection of the ENA for MDPs. (A)** A sample 3-node pre-MDP with $A = 2$ actions (yellow, blue), each leading probabilistically to one of $K = 2$ successor states. A and K are fixed across examples, though the ENA generalizes to other 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 ENA in terms of expected path length (averaged over 100 times 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. In the plot, the thickness of each directed edge represents this probability. **(G)** Paths from node 1 to node 3 generated by the policy resulting from the ENA. **(H)** Paths from node 1 to node 3 generated by the ENA in response to a requirement to avoid nodes with accident probability above 0.8%. **(I)** Cognitive map which the ENA employs for generating the action sequences (policy) indicated in panel H. It results from inhibiting all EMs that traversed nodes that were not sufficiently safe. **(J)** Learned cognitive map (via PCA) of a 12-node pre-MDP. Each node has two outgoing actions, colored orange and blue. **(K)** Available set of episodic memories (EMs). **(L)** When navigating from node 8 to node 4, the agent selects the orange action, which is supported by three episodes highlighted here in different colors.

The ENA 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 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 ENA generates in view of this new safety measure. It can instantly adjust its action selection 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 ENA 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.

2.5 Foraging

We considered so far only applications of the ENA to tasks where reaching a particular state or location was specified as behavioral target. But the EN-approach works just as well for other tasks, provided that the behavioral target can be defined in terms of a set $\mathbb{G}$ of EMs. The ENA generates then a sequence of actions that maximize correlation between the EM-based encoding of the resulting state and $\mathbb{G}$.

In particular, foraging tasks are supported, where a set of target points in a physical environment are given, and the target behavior is a short sequence of movements that visits each of them. This behavioral target is defined by the set $\mathbb{G}$ of all EMs where at least one of the target points was visited. An inherent algorithmic difficulty of value-function based RL solutions is that value functions

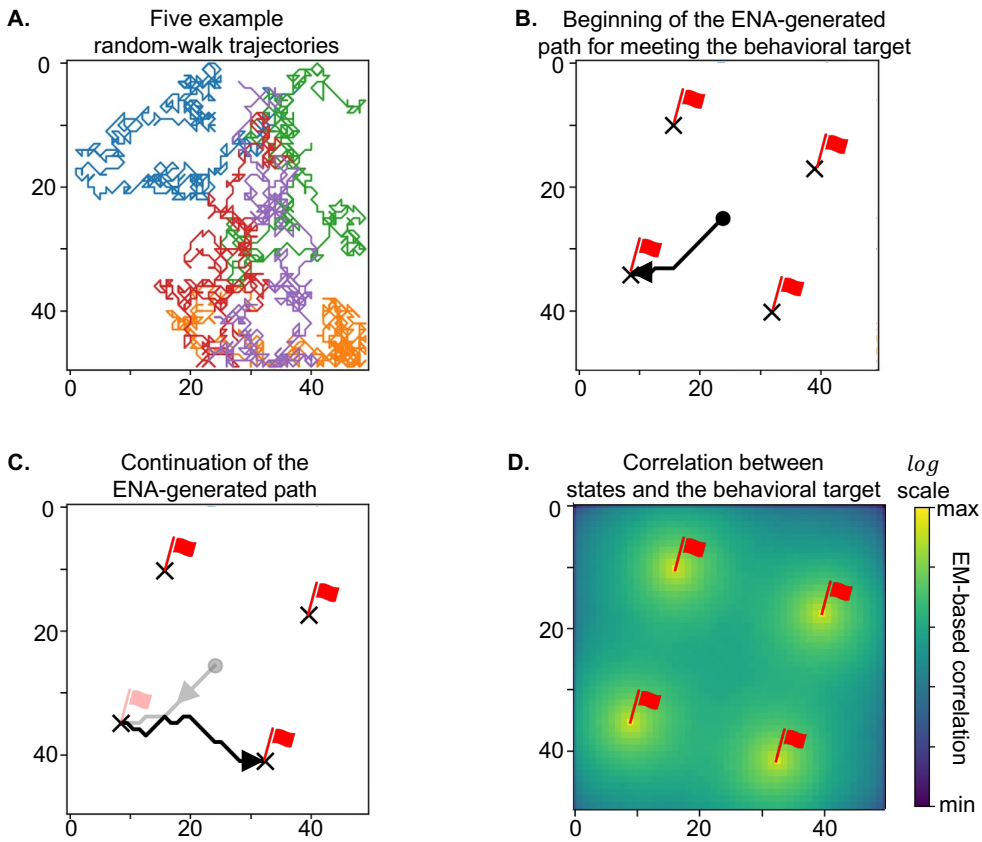


Fig. 5: **Application of the ENA to a foraging task. (A)** 5 samples of random exploration trajectories, from random starting points. Each trajectory, consisting of 500 steps, gives rise to an EM. **(B)** Foraging from a start point in the middle between the 4 target points. The movement trajectory that is generated by the ENA algorithn immediately moves away from the virtual attractor that the center between them generates in common value-based approaches. **(C)** After reaching the first target point, the ENA automatically produces a trajectory to another one of them, once EMs that contain the first one are discarded (or the index neurons for them are inhibited). **(D)** Structure of the cognitive map that is produced by correlations between locations (states) and the set G of EMs that define the behavioral target (compare with the cognitive maps shown in Fig. 2B, F, Fig. 3C, Fig. 4D). Locations between target points have low correlation.

arising from different target points are superimposed at locations in between target points, thereby attracting movements to these in-between locations.

We show in Fig. 5B that a straightforward application of the ENA, following a standard random exploration of the environment (see Fig. 5A), yields a trajectory that directly moves to one of the 4 target points. If one removes then those EMs from $\mathbb{G}$ that had visited this point, the trajectory that is generated by the ENA moves then on to the next target point. Hence it generates a meaningful foraging behavior.

The reason why the ENA does not fall into the trap of moving towards locations between several target points is illustrated in Fig. 5D: The EM-based correlation of these locations (states) with the behavioral target that is defined by $\mathbb{G}$ is fairly low at these in-between locations.

2.6 EM-based navigation in continuous environments

So far we considered for simplicity only environments and tasks where observations (states) and actions were defined by small discrete sets. But the ENA can just as well be applied to continuous environments and observations, and continuous actions. We demonstrate this in Fig. 6 for a continuous 2D environment with obstacles. More precisely, we consider there a fine discretizations of locations and actions, as required for emulation by a digital computer. Fig. 6A shows sample trajectories from random explorations, each representing a curve in the continuous 2D space. Each trajectory gives rise

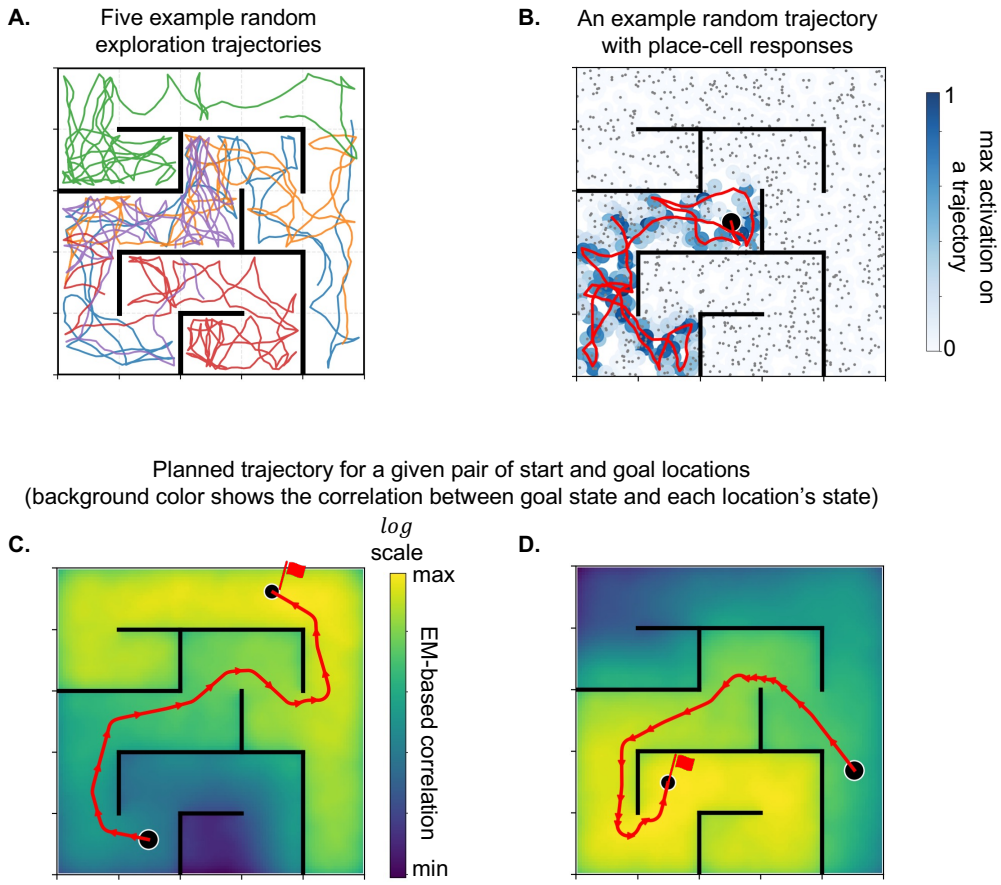


Fig. 6: **Application of the ENA to navigation in a continuous 2D environment. (A)** 5 samples of randomly generated exploration trajectories with local reflection at obstacles, each consisting of 400 steps with step size 0.1. **(B)** Each exploration trajectory is encoded as an EM by the set of place fields (from some finite pool) that are crossed by the trajectory. **(C, D)** Two sample paths from given starting points to given goal points that are generated by the ENA from the EMs that had been acquired during exploration.

to an EM by activating a subset of pre-defined place cells whose place fields overlap with the visited locations (Fig. 6B).

Each point p in the continuous 2D space is then encoded by a sparse vector of non-negative continuous values, instead of the sparse binary vector indicated in Fig. 1D, with one coordinate for each EM. These continuous values encode for each EM how close the corresponding trajectory has come to point p. An implicit cognitive map is defined for this continuous environment by correlations between these sparse continuous-valued vectors (see Fig. 6C, D), analogously as before by correlations between binary vectors. It encodes for each pair of points in the 2D space how often trajectories that are encoded by EMs have come close to both of them.

We define a set of actions in this continuous 2D space by a large randomly selected set of straight point-to-point movement primitives. The ENA produces then for any such movement primitive for which the current location in 2D is close to its start point an estimate of the utility of it. It is defined by the correlation of the vector that defines the behavioral target and the vector that is defined by the end-point of this movement primitive. We show in Fig. 6C-D two examples of movement trajectories from a given starting point to a goal point that are produced in this way by the ENA. More precisely, we show here a smoothed version of the generated sequence of movement primitives (see Fig. S3 for the unsmoothed version) as they are typically generated when they carried out by some physical devices Thus we see that the ENA can just as well be applied for navigation in continuous physical environments. It employs a representation of actions by neurons that fire for specific intended or actual movements to specific endpoints. Maps of such neurons have been found in several brain areas (Andersen and Buneo, 2002; Barnaveli et al., 2025; Gordon et al., 2023; Hatsopoulos et al., 2007).

3 Discussion

In contrast to most currently available RL algorithms, humans are able to adjust their plans on the fly when the goal or context change. In addition, they can usually explain in terms of concrete EMs why they have made a particular decision. Hence currently available RM methods do not capture the full capabilities of the brain for goal-directed decision making. Our study shows that inspiration from the brain, in particular from index neurons for EMs, gives rise to a functionally powerful action selection method, sketched in Fig. 1, that offers brain-like flexibility and explainability. One can understand its function through an analysis of the cognitive map that it creates for any task space, where the distance between two states is quantified in terms of the number of EMs in which they co-occur. This cognitive map provides a transparent strategy for goal-directed action selection: move to that next state that has minimal distance to the goal state, or in other words, the closest episodic neighbor to the goal state in the resulting cognitive map. This cognitive map has the nice property that it can be adjusted on the fly when the current context makes particular EMs more salient, or when new EMs are added (continual learning). Hence we propose to add the ENA to the repertoire of existing methods for solving RL problems. In contrast to the functionally most powerful RL methods such as policy gradient, the ENA does not require deep learning, or any form of offline training. Hence it cannot compete with RL methods that have been designed to reach a single well-defined goal, such as winning a game (Silver et al., 2017). But it offers new benefits for task environments that are arguably closer to those that humans face, where goals, contingencies, and experiences are continuously changing and rational explanations for action selections are often needed. We have shown that the ENA can handle also tasks with uncertain action outcomes, with continuous states and actions, and with multiple ad-hoc goals. Further work will be needed to determine the full range of its applications and its limitations. Note that catastrophic forgetting does not occur in this approach since, instead of overwriting previously learnt knowledge, new neurons are used for encoding new EMs. A limitation for continual learning occurs when one runs out of neurons for that. But there are obvious brain-inspired methods for avoiding that by reusing neurons that served as indices for EMs that turned out to be not relevant, or by condensing several EMs into a scheme (Stickgold and Walker, 2013).

We have shown in Fig. 2 that the ENA exhibits near optimal performance for a standard benchmark task for action selection (Russell, 2010): Finding short paths to any 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 action selection in stochastic environments, and in Fig. 6 that the ENA is also able to produce trajectories to a goal in a continuous 2D environment. In fact, it does not even require a specific goal. It suffices to distinguish some of the EMs as paradigms for future behavior. In this way one can for example model experimental data on foraging hidden reward sites (AbdelRahman et al., 2025).

An essential ingredient of the ENA 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). Also the fMRI data of (Tarder-Stoll et al., 2024) suggest that there exist substantial numbers of neurons in the human hippocampus and higher cortical areas that are active both for a given frame of an EM and for adjacent frames. For the rodent brain we have in addition experimental data which elucidate the emergence of such neurons through 1-shot learning via BTSP (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 has a several seconds long window of plasticity, and can therefore create neurons that fire for several seconds during imprinting and recall of an EM. This works especially well if neural codes for individual frames of an EM are encoded by sparse and largely orthogonal neural codes, as reported for example in (Sun et al., 2025). Offline replay during sharp wave ripples provides a 10-fold speedup (Chen and Wilson, 2023), thereby moving also longer experiences into the plasticity window of BTSP. Very recently it has been shown that not only pyramidal cells in the hippocampus, but also on layer 5 of the neocortex are subject to BTSP, and as a result also fire throughout an EM (Yaeger et al., 2025). The ENA gives rise to the hypothesis that the resulting neural codes for EMs on layer 5 in the neocortex, many of which are known to project to downstream motor areas, implement the EM-based action selection mechanism scheme of Fig. 1E. On the side we want to clarify that neurons that encode indices for EMs are not likely to be the only representations

of EMs. Other representations support recall and replay of individual frames of an EM, and support further uses of EMs.

In contrast to goal-directed decision making methods from RL, the ENA can be implemented through simple online-computations in shallow sparsely active neural networks. In particular, indices for EMs can easily be learnt via one-shot learning through a simple form of BTSP (Wu and Maass, 2025), provided that the corresponding inputs to the index neuron are approximately orthogonal. This tends to be the case in higher brain areas (Sun et al., 2025), and can be achieved in artificial devices through sparse coding, possibly in combination with random bits. The action selection of the ENA 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 ENA only requires self-supervised online learning with simple local rules for synaptic plasticity, which can be implemented through on-chip plasticity of memristor arrays. A first application of BTSP in neuromorphic hardware has already been explored in (Galloni et al., 2024). 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 ENA that is suitable for smaller memristor crossbars with a substantial number of conductance values of each memristor, as discussed in (Li et al., 2017).

The fundamental difference between the ENA and RL methods lies in the data structures which they employ for encoding learnt knowledge. RL methods compress learnt experience into parameter values, whereas the ENA retains records of some of the experiences themselves. In that sense, the data structures that RL-algorithms employ can be viewed as low-dimensional projections of the substantially richer basis for problem solving which the ENA employs. Apart from the resulting loss in flexibility and explainability, the RL strategy to work compress experience into parameter values also requires more time and computational effort to extract a decision from this data structure. In addition, most RL algorithms require complex re-computations when a new experience is added, or when the goal changes. The reason is that value functions and corresponding policies have to the updated in tandem, whereas the ENA can wait with the generation of policies until after learning, and only needs to generate policies step by step, when needed. Decisions of a policy that results from generic RL methods are typically not explainable. This also holds for model-based RL and RL based on the successor representation. Some remedies have been investigated in the burgeoning field of Explainable RL (XRL). Besides explanations in terms of decision trees, one also has worked on methods to extract training trajectories to which a policy decision is sensitive (Deshmukh et al., 2023). But this approach required LSTM networks and transformers.

We want to emphasize that the idea to employ EMs in machine learning is actually already quite old (Hassabis et al., 2017). But most work in this direction, especially numerous studies of episodic control, from (Lengyel and Dayan, 2007) to (Li et al., 2023), have focused on reducing the amount of training, rather than trying to achieve brain-like goal-flexibility, context-awareness, or explainability.

A common feature of the ENA 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 ENA can already produce decent solutions with limited experience, whereas transformers tend to require large sets of training data. Also the ease to produce context-aware and explainable decisions by the ENA 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 ENA

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, as long as these are approximately orthogonal. This would correspond to neural coding 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, and (Sun et al., 2025) on related experimental 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 ENA 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.

Action selection

During producing 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:

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

The resulting plans remain explainable: action selection 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 $\tilde{\mathbf{s}}^*$:

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

4.2 Details to action selection 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 ENA 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 action selection method that requires unrolling the entire solution path 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 action selection, 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 action selection 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 action selection 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 ENA 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 action selection 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 the ENA 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 action selection, $V(s)$ 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 ENA 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, the production of paths 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 framework of (Dayan, 1993).

4.4.2 Multiply–accumulate (MAC)-Based Estimates of the Computational Complexity of Problem Solving 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, often implemented as a single fused multiply–add (FMA) instruction, corresponding to 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-MDP, 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 the ENA, 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.

In addition, VI lacks the flexibility to adapt its learned policy to new goals. Each goal defines a unique reward distribution over the map, requiring the agent to recompute the value function from scratch for every new goal. To ensure fair comparison of computational overhead across algorithms, and to support goal-directed navigation as enabled by both the Successor Representation (SR) and the ENA, I count the total computation time for VI as the sum of all separate value table computations for each goal.

For 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 $(\text{MACs})_{\text{SR}} \approx 2N_o L$ per episode.

For the learning process of an ENA 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 the ENA 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 the ENA 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, the ENA 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 the ENA 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 path length in Fig. 4E, we averaged results over 100 runs for each possible (start, 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 $L = 3$.

4.5 Further details to foraging (multi-goal navigation) in a 2D discrete environment, see Fig. 5

We evaluated the agent in a two-dimensional discrete navigation task designed to probe sequential, multi-goal path generation based on experience acquired through unguided exploration.

Environment and actions.

The environment is a square grid of size 50×50, with integer-valued coordinates $(r, c) \in \{0, \ldots, 49\}^2$. Each grid cell corresponds to a unique state. At every time step, the agent deterministically selects one of four actions: *up*, *down*, *left*, or *right*, resulting in a transition to one of the four von Neumann neighboring cells. If an action would move the agent outside the grid, the position is reflected at the boundary, ensuring that all transitions remain within the environment.

Exploration phase.

During training, the agent explores the environment through unguided random walks. We sample 10^5 exploration episodes, each starting from a uniformly random grid location and consisting of a trajectory of fixed length (500 steps). At each step, one of the four actions is chosen uniformly at random.

For each episode, we record the set of grid locations visited at least once during the trajectory. Temporal order and visitation frequency within an episode are not retained; instead, each episode is represented as a binary indicator over grid locations, or equivalently an OR over grid locations. The learning process is as in the standard ENA.

Goal configuration.

After exploration, the agent is evaluated on a foraging task. The starting location is fixed at the center of the grid, $(24, 24)$. Four target locations are placed at equal Euclidean distance from the start and arranged approximately on a circle surrounding it, resulting in a symmetric configuration in which all target points are equidistant from the initial state. The behavioral target (corresponding to the goal in the preceding demonstrations) is defined by the set of all EMs that visit any of the four target points. Once one of these target points is reached, the EMs in which this point had been visited are removed (or equivalently, the corresponding index neurons are inhibited).

Utility definition.

Utilities are defined as in the case of a single goal point, but for a different set of EMs. In this case the total utility of an action is the sum of the utilities that result from specifying a single one of the four target points as the goal. Target points that have not yet been reached are termed *active*.

Greedy action selection and sequential target removal.

Starting from the current location, the agent constructs a path using a greedy action-selection policy. At each step, the four neighboring states are evaluated and the agent selects the action leading to the state with the highest utility. Ties are broken randomly. This process continues until an active target is reached or a maximum step limit is exceeded.

4.6 Further details to the ENA in a continuous environment, see Fig. 6

Maze settings

The environment is a bounded continuous square domain $[0, L] \times [0, L]$ with $L = 5.0$. Movement is constrained by a set of axis-aligned internal walls. The agent's state is its continuous position $\mathbf{x}_t = (x_t, y_t) \in \mathbb{R}^2$. A proposed displacement is rejected if the straight-line segment between the current and proposed position intersects any wall segment or boundaries.

Exploration trajectories.

To generate exploration trajectories for forming EMs, the agent performs quasi-random walks in continuous space. The next atomic movement step is generated in dependence of the previous one, in order to induce smooth trajectories. This can be implemented as follows. Each episode begins at a sampled starting location and then evolves for a fixed number of steps ($T = 400$ in our experiments) with step size $\Delta = 0.1$. At each time step, the agent maintains a heading angle θ_t and updates it with Gaussian noise,

$$\theta_{t+1} = \theta_t + \epsilon_t, \qquad \epsilon_t \sim \mathcal{N}(0, \sigma_\theta^2), \tag{21}$$

with $\sigma_\theta = 0.3$. The resulting displacement is

$$\mathbf{x}_{t+1}^{\text{prop}} = \mathbf{x}_t + \Delta\,(\cos\theta_{t+1}, \sin\theta_{t+1}), \tag{22}$$

When the proposed movement would result in a collision with a wall or boundary, a new head direction is sampled uniformly from the 180° range that points away from the obstacle. We generated a large pool of such episodes (in simulation, this process can be carried out in parallel on GPU), spanning a grid of starting locations, to obtain broad coverage of the maze.

Place-cell population code.

We represent each continuous position by the activity of a population of place cells. We sampled $P = 1000$ place cell centers $\{\boldsymbol{\mu}_p\}_{p=1}^{P}$ uniformly over the environment. The activation of place cell p at position $\mathbf{x}$ is defined as an exponentially decaying function of Euclidean distance. Further, we set activations lower than a threshold $\vartheta_p = 0.1$ equal to 0:

$$a_p(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \boldsymbol{\mu}_p\|}{\lambda}\right), \qquad a_p(\mathbf{x}) \leftarrow 0 \;\text{ if } a_p(\mathbf{x}) < 0.1, \tag{23}$$

where λ is a length scale (set to $\lambda = 0.15$). To reflect occlusion by maze walls, we further apply a visibility constraint: if the straight-line segment between $\mathbf{x}$ and $\boldsymbol{\mu}_p$ intersects any wall, then $a_p(\mathbf{x})$ is set to zero. The resulting observation $\mathbf{o}$ is the sparse continuous-valued population vector

$$\mathbf{o} \;=\; (a_1(\mathbf{x}), \ldots, a_P(\mathbf{x}))^\top \in \mathbb{R}^P. \tag{24}$$

In our experiments, we used $P = 1000$ place cells and treat $\mathbf{o}_t$ as a 1000-dimensional observation.

Note that in the rodent the place cells that are activated by a trajectory may be especially created through BTSP with a likely center on the trajectory (Bittner et al., 2017).

Learning.

For each exploration episode e (a trajectory $\mathbf{x}_{1:T}^{(e)}$), we define its spatial fingerprint in terms of maximal place-cell activity that it evokes by the following vector:

$$\mathbf{o}^{(e)} = \big(\max_{t\in\{1,\dots,T\}} a_1(\mathbf{x}_t^{(e)}),\ \dots,\ \max_{t\in\{1,\dots,T\}} a_P(\mathbf{x}_t^{(e)}) \big)^\top. \tag{25}$$

We then instantiate one *index neuron* per episode, which stores this vector in its incoming synaptic weights from the place-cell population. Concretely, letting $\mathbf{Q}_e \in \mathbb{R}^P$ denote the weight vector for the index neuron corresponding to episode e, we apply a one-shot update:

$$\mathbf{Q}_e = \mathbf{o}^{(e)}. \tag{26}$$

This is a direct one-shot instantiation of the simplified BTSP mechanism: an episode triggers a single synaptic imprint that binds the episode to the pattern of maximal place-cell activity associated with that episode, without iterative learning or backpropagation.

Actions are defined by pairs of points in continuous 2D, that define the beginning and end point of a movement primitive. We used as points a sampled a set of $N_w = 500$ points $\{\mathbf{z}_i\}_{i=1}^{N_w} \in \mathbb{R}^2$ uniformly at random from the free space of the maze.

An action a is defined as a straight-line displacement from one point to another. Two of the sampled points $\mathbf{z}_i$ and $\mathbf{z}_j$ are connected by an action (movement primitive) if and only if the straight-line segment between them does not intersect any wall. This visibility constraint ensures that each action corresponds to a geometrically feasible movement in the continuous environment.

Each action a is associated with a dedicated action neuron n_a as in the discrete case. We denote by $\mathrm{end}(a) \in \mathbb{R}^2$ the location that is reached by executing action a, i.e., the target point of that action. The corresponding state embedding of this outcome is given by $\mathbf{Qo}(\mathrm{end}(a))$, where $\mathbf{o}(\mathrm{end}(a))$ is the place-cell population activity at location $\mathrm{end}(a)$.

The learning rule of n_a's incoming weights is identical to Eq. (2): Each action neuron n_a mirrors its resulting location's state embedding into its incoming weights in one-shot:

$$\mathbf{W}_a = \mathbf{Qo}(\mathrm{end}(a)) \tag{27}$$

Embedding of query locations.

Given a query location $\mathbf{x}^*$ (e.g., a goal position), we compute its place-cell activity vector $\mathbf{o}(\mathbf{x}^*)$ using the same place-cell and visibility model. We then embed $\mathbf{x}^*$ in the space of episode indices by projecting through the embedding matrix:

$$\mathbf{s}(\mathbf{x}^*) = \mathbf{Qo}(\mathbf{x}^*). \tag{28}$$

Similarity between a query embedding and other maze locations is computed via dot products in this episode-index space, yielding a spatial similarity map that highlights regions supported by similar episodic evidence. This forms the background utility values in Fig. 6C-D.

Action selection.

Starting from its current position, the agent repeatedly performs the following steps. (i) It maps its continuous position $\hat{\mathbf{x}}_t$ to the nearest starting point $\mathbf{z}_i$ of an action that is directly visible from $\hat{\mathbf{x}}_t$, which determines the set of candidate actions. (ii) An affordance module further gates these actions by vetoing any action whose resulting straight-line movement from $\hat{\mathbf{x}}_t$ would intersect a wall, or the ending location is too close to a wall (distance less than 0.1). (iii) Among the remaining actions, the agent selects the one with the highest utility, computed as the inner product between the action neuron's weights and the goal-related state embedding, $\mathbf{Ws}(\mathbf{x}^*)$. For simplicity, the agent starts its movement from $\hat{\mathbf{x}}_t$ directly, instead of the near-by starting point of this action. (iv) To ensure smooth execution, the agent advances only a fractional distance toward the selected action's end point, executing $\Delta\mathbf{x}_t = \eta\,(\mathrm{end}(a) - \hat{\mathbf{x}}_t)$, with $\eta = 0.5$, while continuously enforcing collision constraints. If the goal location becomes directly visible from $\hat{\mathbf{x}}_t$, the agent moves straight to the goal.

Trajectory smoothing.

For visualization and for producing physically plausible continuous motion, we post-processed the piecewise-linear path with a smoothing procedure that enforces gradual changes in heading while respecting walls. See supplementary section E for details.

5 Acknowledgments

We would like to thank 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

AbdelRahman, N.Y., W. Jiang, L.T. Coddington, S. Gong, J.T. Dudman, and A.M. Hermundstad. 2025. Composing trajectories for rapid inference of navigational goals. *bioRxiv*: 2025–09 .

Andersen, R.A. and C.A. Buneo. 2002. Intentional maps in posterior parietal cortex. *Annual review of neuroscience 25*(1): 189–220 .

Barnaveli, I., S. Viganò, D. Reznik, P. Haggard, and C.F. Doeller. 2025. Hippocampal-entorhinal cognitive maps and cortical motor system represent action plans and their outcomes. *Nature Communications 16*(1): 4139 .

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 .

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

Chettih, S.N., E.L. Mackevicius, S. Hale, and D. Aronov. 2024. Barcoding of episodic memories in the hippocampus of a food-caching bird. *Cell 187*(8): 1922–1935 .

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

Deshmukh, S.V., A. Dasgupta, B. Krishnamurthy, N. Jiang, C. Agarwal, G. Theocharous, and J. Subramanian. 2023. Explaining rl decisions with trajectories. *arXiv preprint arXiv:2305.04073* .

Durstewitz, D., B. Averbeck, and G. Koppe. 2025. What neuroscience can tell ai about learning in continuously changing environments. *Nature Machine Intelligence*: 1–16 .

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 .

Gordon, E.M., R.J. Chauvin, A.N. Van, A. Rajesh, A. Nielsen, D.J. Newbold, C.J. Lynch, N.A. Seider, S.R. Krimmel, K.M. Scheidter, et al. 2023. A somato-cognitive action network alternates with effector regions in motor cortex. *Nature 617*(7960): 351–359 .

Hassabis, D., D. Kumaran, C. Summerfield, and M. Botvinick. 2017. Neuroscience-inspired artificial intelligence. *Neuron 95*(2): 245–258 .

Hatsopoulos, N.G., Q. Xu, and Y. Amit. 2007. Encoding of movement fragments in the motor cortex. *Journal of Neuroscience 27*(19): 5105–5114 .

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 .

Lengyel, M. and P. Dayan. 2007. Hippocampal contributions to control: the third way. *Advances in neural information processing systems* 20 .

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 .

Li, Z., D. Zhu, Y. Hu, X. Xie, L. Ma, Y. Zheng, Y. Song, Y. Chen, and J. Zhao. 2023. Neural episodic control with state abstraction. *arXiv preprint arXiv:2301.11490* .

Liu, G., M. Tang, and B. Eysenbach. 2024. A single goal is all you need: Skills and exploration emerge from contrastive rl without rewards, demonstrations, or subgoals. *arXiv preprint arXiv:2408.05804* .

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* .

Milani, S., N. Topin, M. Veloso, and F. Fang. 2024. Explainable reinforcement learning: A survey and comparative review. *ACM Computing Surveys 56*(7): 1–36 .

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 .

Purandare, C. and M. Mehta. 2023. Mega-scale movie-fields in the mouse visuo-hippocampal network. *Elife* 12: RP85069 .

Qing, Y., S. Liu, J. Song, Y. Zhou, K. Chen, H. Wang, and M. Song. 2022. A survey on explainable reinforcement learning: Concepts, algorithms, challenges. *arXiv preprint arXiv:2211.06665* .

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* .

Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature 550*(7676): 354–359 .

Stachenfeld, K.L., M.M. Botvinick, and S.J. Gershman. 2017. The hippocampus as a predictive map. *Nature neuroscience 20*(11): 1643–1653 .

Stickgold, R. and M.P. Walker. 2013. Sleep-dependent memory triage: evolving generalization through selective processing. *Nature neuroscience 16*(2): 139–145 .

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 .

Tarder-Stoll, H., C. Baldassano, and M. Aly. 2024. The brain hierarchically represents the past and future during multistep anticipation. *Nature Communications 15*(1): 9094 .

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 .

Xiao, K., Y. Li, B.J. Sullivan, G. Li, and J.C. Magee. 2025. Rapid neocortical network modifications via dendritic plateau potential induced plasticity. *bioRxiv*: 2025–11 .

Yaeger, C.E., R. Mojica Soto-Albors, W. Liu, A. Beltramini, and M.T. Harnett. 2025. Plateau potentials are instructive signals for behavioral timescale synaptic plasticity in the neocortex. *bioRxiv*: 2025–11 .

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 .

Zutshi, I., A. Apostolelli, W. Yang, Z.S. Zheng, T. Dohi, E. Balzani, A.H. Williams, C. Savin, and G. Buzsáki. 2025. Hippocampal neuronal activity is aligned with action plans. *Nature*: 1–9 .

Supplementary Information for

Episodic memories make goal directed action selection context-aware and explainable

Yukun Yang and Wolfgang Maass
Institute of Machine Learning and Neural Computation, Graz University of Technology
yukun.yang@tugraz.at, wolfgang.maass@tugraz.at

Table of Contents

A. Non-orthogonal observations 28
B. Directed graphs 29
Figure S1. Performance of EN for goal-directed action selection in directed graphs. .... page 29
C. MDPs 30
Figure S2. Comparison of VI, SR, and EN on a 32-node pre-MDP. ..... page 30
D. Variation of the model suitable for memristor implementation 30
E. Continuous environment 31
Figure S3. Unsmoothed sequence of action primitives.

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 for the ENA. In this section, we explain how our model can tolerate 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:

$$(\mathbf{Q})_i = \bigvee_{j=1}^{L} \mathbf{o}_j \quad \text{if observation } \mathbf{o}_j \in \text{ the } i^{\text{th}} \text{ trajectory.} \tag{29}$$

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:

$$\mathbf{s} = f(\mathbf{Q} \cdot \mathbf{o}). \tag{30}$$

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.

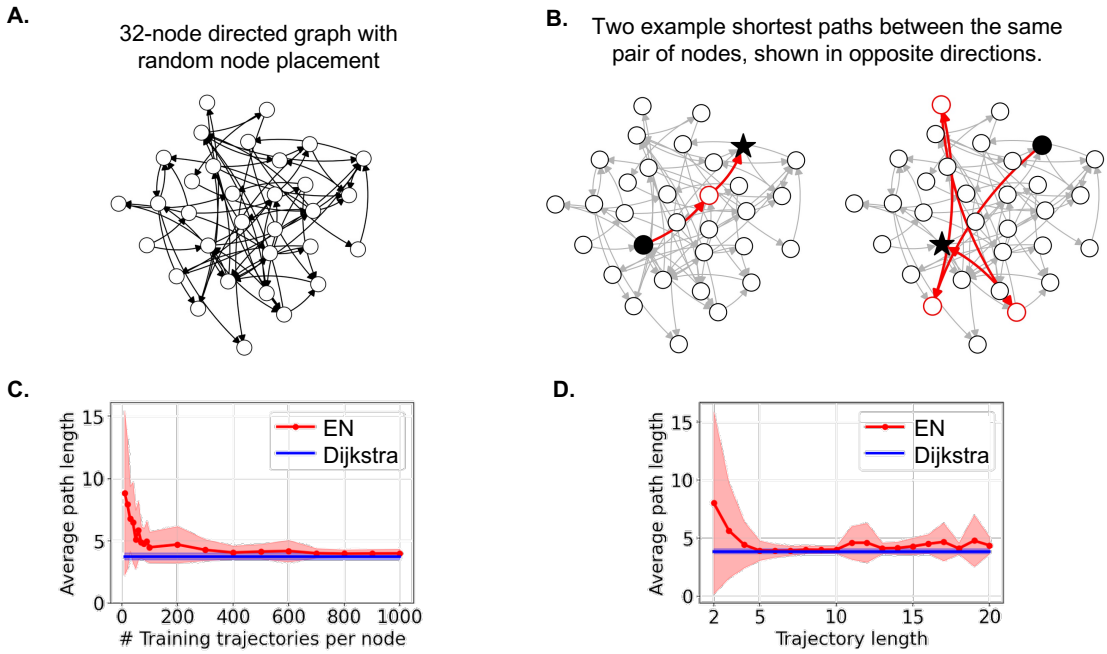


Fig. S1: **Performance of EN for goal-directed action selection in directed graphs. A.** An example random directed graph. **B.** Two example shortest paths for a pair of nodes in opposite directions. The trajectories can be very different when switching start and goal nodes. **C.** and **D.** show mean ± 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.

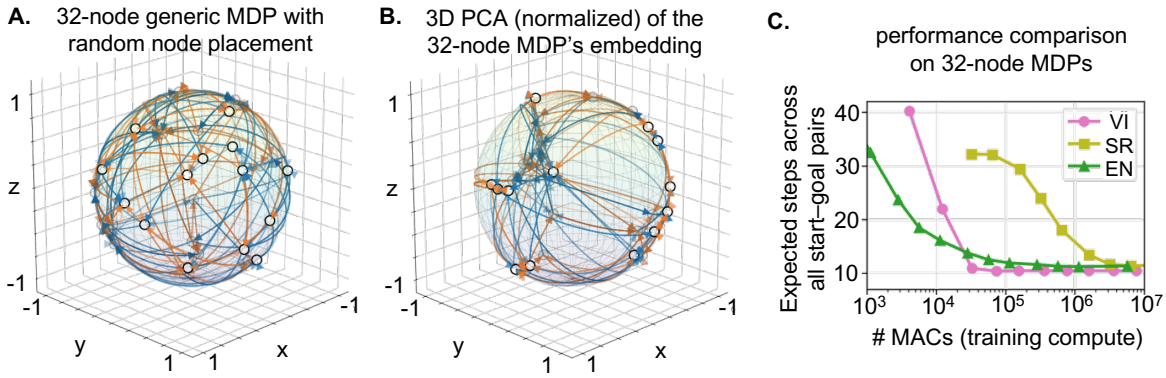


Fig. S2: **Comparison of VI, SR, and EN on a 32-node pre-MDP. (A)** A 32-node generic pre-MDP with random node placement. **(B)** Projection of the cognitive map of this 32-node pre-MDP onto a 3D sphere. **(C)** Comparison of value iteration (VI), successor representation (SR), and the EN-algorithm in terms of expected path length (averaged over 100 times 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:

$$\mathbf{Q}'_{ij} \leftarrow \mathbf{Q}_{ij} + 1 \quad \text{if node } i \text{ occurs before } j \text{ in a trajectory.} \tag{31}$$

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 goal-directed action selection, 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.

E Continuous environment

The produced trajectories consist of sequences of movement primitives, as shown in Fig. S3. In this section, we explain in detail how we smoothed the trajectories and produced the plots in Fig. 6C-D.

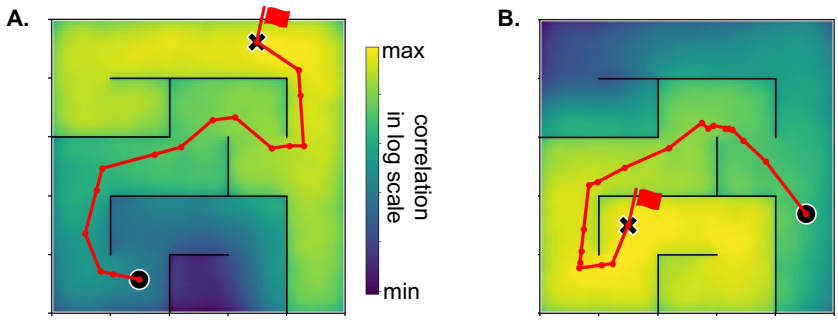


Fig. S3: **Unsmoothed sequence of action primitives (A),(B)** The unsmoothed version of the ENA-generated trajectories from Fig. 6C-D respectively.

Let $\{\mathbf{x}_k\}_{k=1}^{K}$ denote the sequence of all visited points on a trajectory. We generate a dense continuous trajectory $\{\hat{\mathbf{x}}_t\}_{t\geq 0}$ by integrating a unicycle-like heading dynamics with bounded angular velocity.

We maintain a current position $\hat{\mathbf{x}}_t \in \mathbb{R}^2$ and heading angle $\theta_t \in (-\pi, \pi]$. At each integration step, we define the current target point as $\mathbf{x}_\ell$ (initialized to $\ell = 2$) and compute the desired heading

$$\theta_t^\star = \operatorname{atan2}\big((\mathbf{x}_\ell - \hat{\mathbf{x}}_t)_y,\ (\mathbf{x}_\ell - \hat{\mathbf{x}}_t)_x\big). \tag{32}$$

The heading is updated using a proportional controller with an explicit bound on angular change per step:

$$\theta_{t+1} = \theta_t + \operatorname{clip}(\kappa \operatorname{wrap}(\theta_t^\star - \theta_t),\ -\Delta\theta_{\max},\ +\Delta\theta_{\max}), \tag{33}$$

where $\operatorname{wrap}(\cdot)$ maps angles to $(-\pi, \pi]$. In all experiments we set the turning gain to $\kappa = 0.45$ and the maximum allowed heading change to $\Delta\theta_{\max} = 10^\circ$ per integration step. These values ensure smooth curvature while allowing the trajectory to track moderately sharp turns when required by maze geometry.

Given the updated heading, a forward Euler step of fixed spatial length δ is proposed:

$$\hat{\mathbf{x}}_{t+1}^{\text{prop}} = \hat{\mathbf{x}}_t + \delta\,(\cos\theta_{t+1},\ \sin\theta_{t+1}). \tag{34}$$

We set the integration step size to $\delta = 0.03$, which provides sufficient spatial resolution relative to the maze size ($L = 5.0$) while keeping the computational cost low.

A point $\mathbf{x}_\ell$ is considered reached once the proposed position satisfies

$$\|\hat{\mathbf{x}}_{t+1}^{\text{prop}} - \mathbf{x}_\ell\| < \varepsilon, \tag{35}$$

with tolerance $\varepsilon = 0.04$. The target index is then advanced to $\ell \leftarrow \ell + 1$.

To enforce obstacle compliance, any proposed step that would cross a wall segment or exit the environment bounds is rejected. Upon rejection, we perform a local angular search around the current heading by testing candidate headings

$$\theta_{t+1} \pm m\,\Delta\theta_{\text{avoid}}, \qquad m = 1, \ldots, M, \tag{36}$$

where the avoidance increment is $\Delta\theta_{\text{avoid}} = 4^\circ$ and the maximum number of trials is $M = 60$. The first candidate producing a collision-free step of length δ is accepted. If no collision-free direction is found, the step size is reduced multiplicatively as $\delta \leftarrow 0.7\,\delta$ (with a lower bound of 10^{-4}) and the update is retried.

This smoothing procedure preserves straight segments in free space, enforces bounded curvature near turns, and yields collision-free continuous trajectories that closely follow the point sequence. For visualization, we annotate each connection between points with an arrow indicating the local direction of motion at the midpoint of the corresponding smoothed segment, computed from finite differences of $\{\hat{\mathbf{x}}_t\}$.

This receding-horizon procedure yields a continuous executed trajectory while keeping goal-directed action selection local and computationally lightweight.