# Description, Implementation & Analysis of the Tollman-Eichenbaum Machine

**Luke M. Hollingsworth**
Dept. of Computer Science
University College London
London, UK
luke.hollingsworth.21@ucl.ac.uk

**Clémentine C. J. Dominé**
Gatsby Computational Neuroscience Unit
University College London
London, UK
clementine.domine.20@ucl.ac.uk

**Rodrigo Carrasco-Davis**
Gatsby Computational Neuroscience Unit
University College London
London, UK
rodrigo.cd.20@ucl.ac.uk

**Andrew M. Saxe**
Gatsby Computational Neuroscience Unit
University College London
London, UK
a.saxe@ucl.ac.uk

September 22, 2022

## Abstract

Whilst both domains of experimental and computational neuroscience have advanced in leaps and bounds over the past 2 decades, the ease with which scientists in these domains collaborate is becoming increasingly limited. Both models and experiments in neuroscience are typically constrained to a small area of investigation, be it neurological or behavioral. This specificity can make it difficult to easily and reliably compare results between the two domains, with many influential models initially being applied to only a handful of experimental settings. Here we aim to describe and implement an influential model of hippocampal function, the Tollman-Eichenbaum machine (TEM), into a new standardised and open-source collaborative toolkit, NeuralPlayground (NPG). TEM models hippocampal (HPC) and entorhinal cortex (EC) function by generating abstract representations of it's environment and using these to predict future sensory observations; the model has been widely influential in the understanding of the HPC and EC, therefore it's implementation into an open-source toolkit will hopefully encourage prolific engagement and investigation. We aim to both expand on the model theory and offer a concise description of its implementation in our RL-based environment. In addition to it's implementation, we investigate some key features of the Tollman-Eichenbaum, and provide qualitative and quantitative insight into it's function and results. We hope that in doing so, we make valuable contributions to the understanding of TEM, whilst also proving the utility of our standardised framework.

## Acknowledgments

This paper was supervised primarily by Andrew Saxe, however a great deal of support was given by PhD students Clémentine Dominé and Rodrigo Carasco-Davis. They are also the owners of the collaborative framework which forms the context of the following thesis, however this framework is not publicly available at the time of writing.

# 1 Introduction

Despite its roots in ancient Egypt [1], neuroscience has emerged as an explicit domain of study over the past half a century, with most of the work being done in imaging and physiology. The world of *experimental neuroscience* has yielded many great advances in understanding of the brain; these range from how single gene defects contribute to neurological disorders [2], to the malleability's of the brain itself, even into adulthood [3]. Most of these discoveries were made in parallel to advancements in imaging technology, enabling us to peer deeper and more precisely into the inner workings of the brain.

In the past couple of decades, a similar advance in computational performance and theoretical understanding have driven a growth in the domain of *computational neuroscience*. Here, scientists use computational models to probe the foundations of complex, neurological systems. Our area of focus is the hippocampus (HPC) and neighbouring entorhinal cortex (EC), where these models seem to reproduced many well-known neural features, such as grid [4, 5, 6, 7] and place cells [8, 9], whilst contributing new theoretical understanding to these phenomena.

Many stable neural representations have been discovered in the hippocampus and entorhinal cortex, especially in the context of spatial navigation. A key neural feature that helps animals to represent space is the entorhinal grid cell [10]. These are place-modulated neurons that have multiple firing locations and thus create a periodic array that covers the entire available surface of an open 2D environment. Similar features include place cells in the hippocampus [8, 9], which become active when an animal is in a specific location within an environment, and band cells [11] which have periodic firing patterns composed of plane waves (or bands). These neural representations have also been observed to undergo a process of 'remapping' [12] when a new environment is presented to the animal, whereby the cortical map is reorganised to better represent the new environment.

Due to the only recent emergence of computational neuroscience, as well as the extent to which the field and expertise of its scientists differ from the experimental discipline, there is a lack of communication between the two. The specificity of both computational models and experimental results, as well as the vast number of each, makes objective and unbiased comparison of the two difficult. Whilst the importance of productive comparison between experimentalists and theorists is clear, the tools available to facilitate this are limited.

## 1.1 NeuralPlayground

Our new open-source framework, NeuralPlayground (NPG), offers a potential solution to the problem of these empirical comparisons, effectively introducing a toolkit for collaborative use. The toolbox encourages cross-collaboration between experimental and computational neuroscientists by offering a standardised environment through which they can compare their model to experiment, or vice versa. With NPG, we focus on functions of the hippocampus and entorhinal cortex with a primary interest in spatial navigation.

The basis of NeuralPlayground is a pair of classes that interact in an reinforcement learning-like paradigm. Reinforcement learning (RL) is a field of study in machine learning and neuroscience that investigates interactions between intelligent agents and a complex environment; typically the actions of these agents are chosen to maximise some notion of reward [13]. RL is of great interest to neuroscientists [14, 15] as it clearly maps to the physical interactions of animals and the environment in which they live; additionally, understanding the 'agent policy' can be a means of investigating the behavioural choices of the animal. We note that, whilst our framework is capable of including reward as part of the agent-environment interaction, the version of the model we introduce is not currently concerned with the notion of reward.

One of these classes contains all the information and operations performed by the agent, whilst the other contains all the information inherent in the environment. The path of an agent through the environment can be passed in directly or generated by the agent on-the-fly. At each step in this path, the agent makes an observation of the environment and with it, updates its internal representation of the environment. These internal representations are used at the end of training to produce various biologically-grounded neural features, such as the grid and place cells we are interested in here. Within NPG, we have already implemented models associated with spatial navigation such as the successor representation [16], and aim to build on this into the future.

By implementing a standardised two-class RL framework, models and results can be easily implemented and compared to one another. This means that models can be applied to experimental settings they have never seen before, in a systematic way that produces novel results, this is outlined graphically in fig. 1. The specific area of interest for NPG is the hippocampus and entorhinal cortex. There are many interesting neural features that are associated with these areas and few models are able to account for more than a handful of them. In building this framework, we hope to offer a platform for scientists, interested in the HPC and EC, to compare and analyse models with experimental observations. Here, we implement a recent model of HPC and EC function as an agent in NPG, clarifying and expanding upon the model theory, and apply it to novel environments.

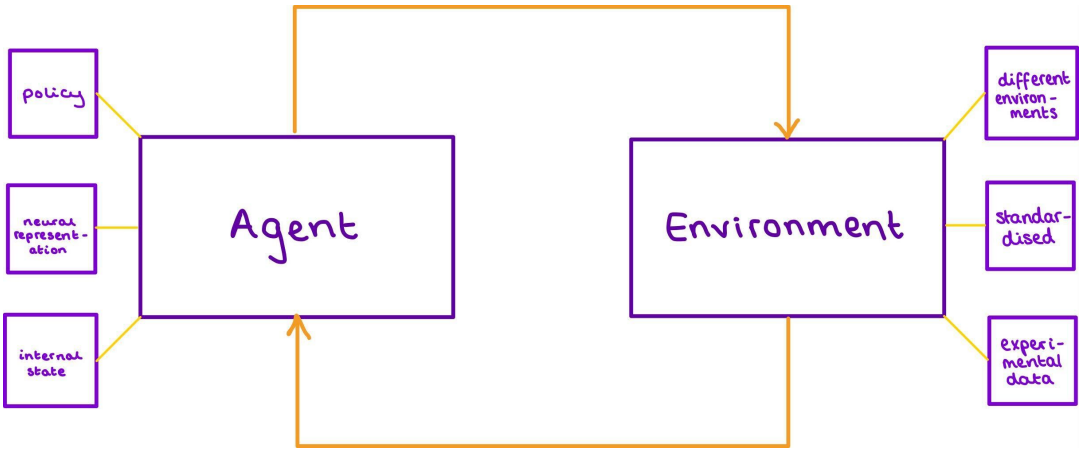


Figure 1: **Implementation Overview:** A high-level graphic of the interaction between environment and implemented model. Both interact at each time-step, with the environment offering various experimental arenas and data, and the agent reproducing neural phenomena of interest.

### 1.2 Literature Review

The original Tollman-Eichenbaum Machine (TEM) paper [4] and associated code both successfully reproduce neural features of interest and investigate more complex phenomena, such as object vector cells [17] and hierarchical environments [18]. The supplementary material also goes a way to describing the implementation of TEM in their code. The results of the model are described in great detail, and the links to biological plausibility are made clear.

TEM has been hugely influential in advancing the understanding of hippocampal and entorhinal function. With over 170 citations (as of 12$^{\text{th}}$ September 2022), it has informed publications across multiple disciplines from Psychology [19] to Machine Learning [20], not to mention the impact it has had in neuroscience. TEM describes the conjunction of two prominent schools of thought in the domain of HPC and EC function, namely Tollman's idea of a cognitive map that represents an abstraction of space, and Eichenbaum's view of the hippocampus as a locus of relational memory. The aggregation of these two theories under one theoretic framework has made TEM of distinct value to those interested in understanding HPC function. TEM has also been explored and expanded upon by other scientists interested in hippocampal function and cognitive abstraction [21].

However, due to it's complexity, the connection between theory and implementation is at time opaque. There are discrepancies in the general model methods (probabilistic vs. deterministic), as well as in the finer details of implementation, such as the weights that are trained and the specific losses that are handled throughout. The mathematics and naming scheme of TEM are also based on the presumption of a probabilistic model and therefore become confusing considering the implementation is strictly deterministic.

This paper aims to add the Tollman-Eichenbaum machine to our collaborative framework, NeuralPlayground, whilst also offering robust description and analysis of some key results. We also aim to offer a rigorous theoretical description that is consistent with our implementation. In doing so, we hope to provide those interested in TEM, and hippocampal function more generally, with a transparent and consistent parallel between theory and implementation, as well as a novel implementation that offers improved accessibility for modification and advancement.TEM has also garnered a lot of interest from those in the field of reinforcement learning [22, 20, 23]; we therefore hope to expand on the possibilities of future RL advancements to TEM, as a result of it's inclusion in our framework.

### 1.3 Contribution

#### 1.3.1 Description

The primary aim of this paper is to provide a transparent and robust implementation of the Tollman-Eichenbaum machine to our collaborative framework, NeuralPlayground. As a result of this, a large part of the following work is dedicated to outlining and detailing the function and operation of processes inherent in TEM. A functional description of core process is given both graphically and mathematically, and an effort is made to highlight areas that aren't described in detail in the original paper.

In our description of this influential model, we attempt to provide a robust parallel between theory and implementation, such that the reader is able to understand and interact with the model, as a feature of our open-source framework. We expressly outline the weights and variables used throughout the model, whilst also expanding upon

theory that has been until now, only implicitly referenced. The aim of this is to offer a guide to understanding and implementing TEM, that can be used in conjunction with our collaborative framework.

#### 1.3.2 Implementation

It should first be noted that here, we don't account for all the complex behaviour exhibited by TEM; we implement a simplified version that only deals with the most basic of environments (square 2D) and doesn't include shiny objects responsible for producing object vector cells. This simplified version still captures the fundamental functioning of the model whilst also offering a more transparent and easily understood interpretation of the theory and implementation.

By focusing on an implementation-based description of the Tollman-Eichenbaum machine, we are able to identify and clarify discrepancies between theory and code, and thus address them. We are also able to produce a code-base that is more accessible and therefore more easily investigated and improved. We hope that this will enable the large collection of scientists interested in TEM to make novel contributions to the understanding of the HPC and EC, by utilising TEM.

We are excited by the future possibilities offered by advancements to the Tollman-Eichenbaum machine, and we hope that this implementation will encourage and enable researchers to better understand the function of the hippocampus and surrounding systems. One possible extension would be to introduce an RL model that influences the action policy, based on the internal state of the agent, or other input. This might mirror the role of the dorsolateral striatum (DLS) in learning associations between actions and egocentric representations, via model-free response learning. Models of the relationship between the HPC and DLS have been modeled in this way [24], however TEM offers an extension by including abstract EC representations as a factor in learning hippocampal place cells and predicting future sensory observations.

#### 1.3.3 Analysis

In this paper, we also provide analytical results from studies of the original implementation of TEM. We deem these to be of importance in better understanding the key results from TEM, and therefore its function, as they highlight features that greatly influence TEM's performance. We also provide some results for which the reasoning is unclear, and therefore may require further investigation; the inclusion of some model features are vital for performance yet reflect design choices that are unclear or lack biological plausibility.

We perform several studies, focused on ablating features of the model we deem interesting and important. An investigation into key weights is done and the hierarchical structure, central to memory retrieval and discussed at length below, is explored in detail. We also highlight key features which see no mention in the original paper, yet are crucial for many of the important results of TEM to emerge.

## 2 The Tollman-Eichenbaum Machine

### 2.1 TEM & the Brain

The Tollman-Eichenbaum machine [4] is a model capable of generating representations that resemble some of the neural activity seen in the hippocampus and entorhinal cortex. The model takes inspiration from Tollman's theory of an internal representation, or cognitive map [25], and combines this with the relational memory of Eichenbaum [26]. TEM uses these concepts to both sequentially learn within an environment and to learn abstract features, common across different environments. Over the course of the agent's trajectory, it learns an abstract representation of the spatial structure it inhabits, and as a result, is able to make predictions from state-action pairs ($[apple, North]$), even if it has never seen this specific pair before.

The structure of TEM is biologically motivated, and reflects the interactions between key brain areas responsible for spatial navigation and memory. At the lowest level of the model, sensory observations $x$ mirror representations of the lateral entorhinal cortex (LEC) [27] whilst the most abstract representations $g$ are analogous to those found in the medial entorhinal cortex (MEC) [28]. These two representations are handled separately and are only every brought together to retrieve memories via the hippocampal (HPC) representation $p$.

Manns and Eichenbaum (2006) [29] described hippocampal representations as conjunctions between the abstract representations and sensory input from the MEC and LEC, respectively; thus, sensory observations are combined with EC representations to generate HPC place cells $p$, such that $p = g \cdot x$. Mirroring hippocampal synaptic potentiation, memories are rapidly stored in the weights $M$, between $p$, using a Hebbian learning rule [30].

TEM has been shown to reproduce some of these well-known neural features, as well being able to remap between environments. The abstract EC representation $g$ resembles grid cells and band cells, shown in fig. 2; by splitting $g$ into 5 temporally-filtered streams, one is able to generate grid cells on a variety of spatial scales (modules). The formation of memories with place-like representations also encourages remapping across environments. Again, these

place-like fields span multiple sizes and thus mirror the hierarchical composition of hippocampal place fields [31]. Similarly, TEM's HPC cells demonstrate remapping by not preserving their spatial correlation, but instead relocating under different environments.

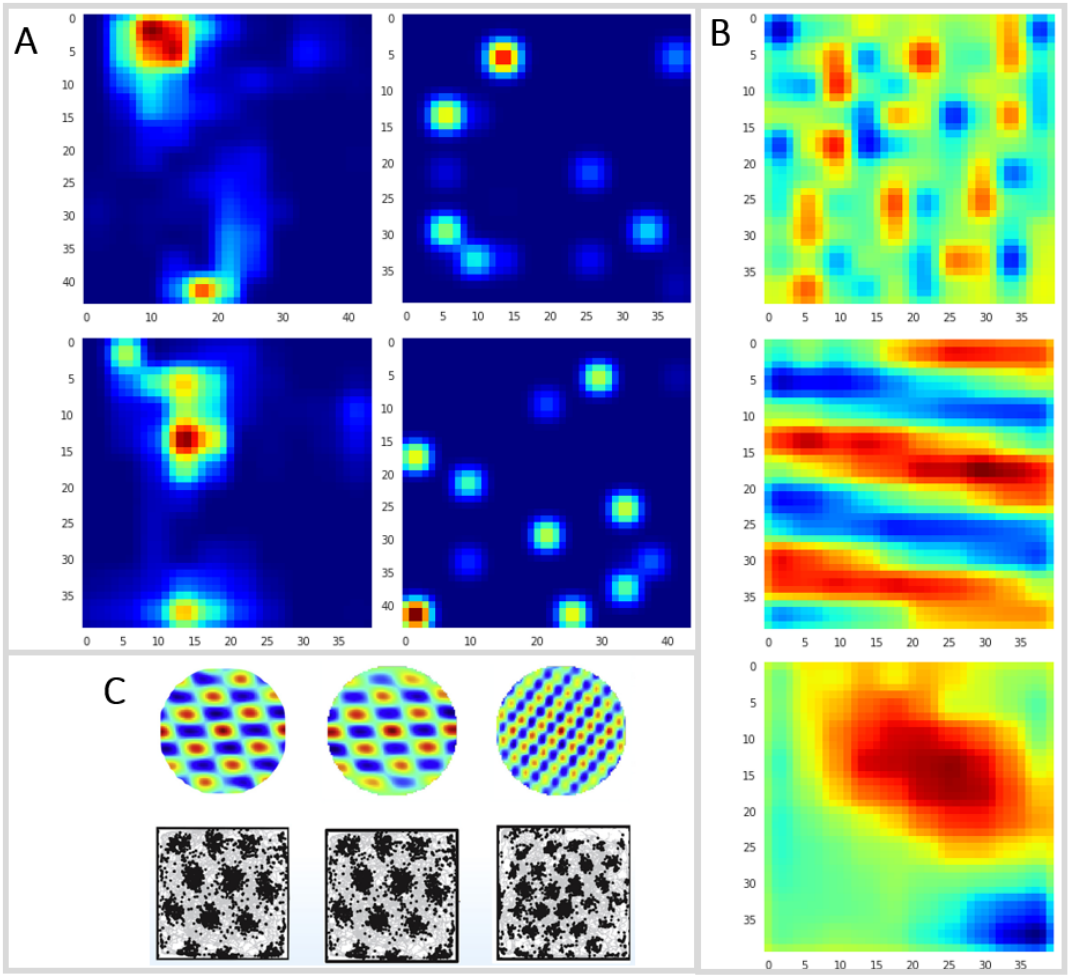


Figure 2: **TEM Cell Representations: A:** Learned memory representations $p$ resemble place cells with different field sizes. **B:** Structural representations $g$ at different spatial scales resemble grid cells. **C:** Autocorrelation data compared with real data [32, 33].

The Tollman-Eichenbaum machine has also been able to make predictions of neural observations where they have yet to be done experimentally. When applied to the lap-based experiment of Sun et al. 2020 [34], TEM showed hippocampal cells retaining lap specificity after remapping which match the results of experiment. The MEC cells produced by TEM also learn to represent the non-spatial task structure of being rewarded every 4 loops; these results have not yet been made experimentally but are predicted by this model.

### 2.2 Structure

In order to predict an upcoming sensory observation, $x$, TEM must build an internal representation of the space it is in. For this representation to generalise across environments, it helps for this representation to be invariant to the particular sensory objects of any one environment. In order to accomplish this, we view TEM at two different scales, each with different learning rules and network weights; these allow the model to both learn the structure of the environment it is in, as well as generalise to environments with a similar structure. The model also takes a great deal of inspiration from biological structure, and this results in an architecture that resembles some of the neural areas responsible for spatial abstraction and navigation.

Two main processes work together in TEM, one **generative** and one **inference-based**, in order to make predictions of sensory objects, as well as produce HPC and MEC representation. The generative and inference processes work together as part of an 'inner loop' that learns the structure of the current environment, enabling it to make accurate 'zero-shot' predictions of states it returns to. Figure 3 shows this structure in a simplified graphical form. Whilst the

generative process is responsible for generating predictions of sensory observations, using its abstract representation, the inference process is constantly updating this representation via the true observation.

The generative and inference processes operate at every time-step and are connected across time through a step-update process. We note that in the original paper, this process is subsumed by the generative model; however when implemented, the step update is separated and we believe it better described this way. Typically, generative models are responsible for making predictions of observations from latent variables, whilst inference models 'infer' latent variables from observations. The step update here does neither and instead translates between the latent variables at each time-step, by generating and maintaining the abstract MEC representation used in both generative and inference processes.

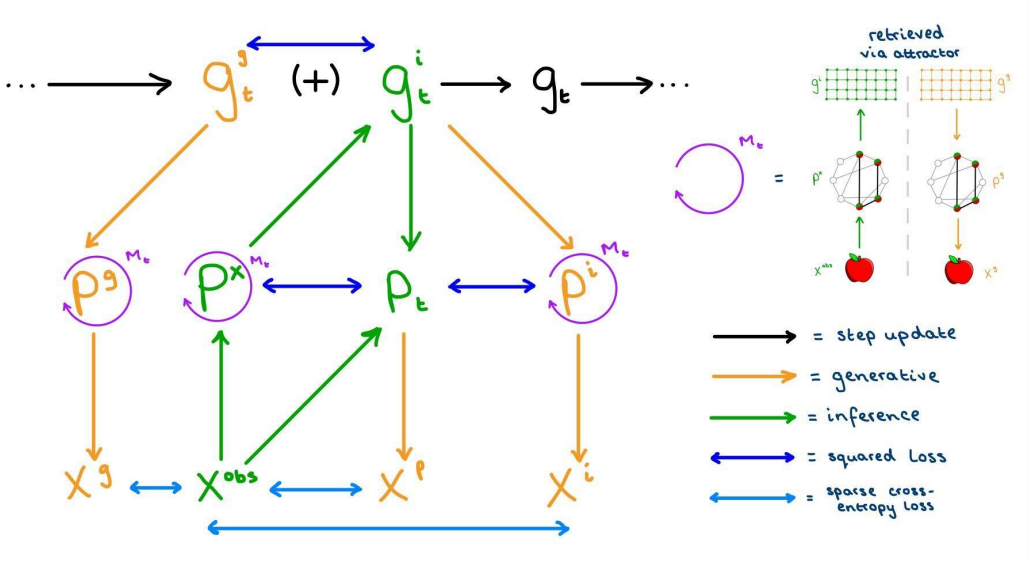


Figure 3: **TEM Basic Structure:** The generative and inference models, connected across each time-step by the step updates. Each directional arrow represents a fully-connected layer or MLP, with bidirectional arrows representing losses. Details of the weights of these layers are described in more detail in fig. 5. The top-right pane describes the dynamics of retrieval via the attractor network.

### 2.3 Initial Processing

Prior to any calculations, the sensory input $x$ and medial entorhinal representations $g$ must be manipulated so that they are of the right format to be passed into the hippocampus. The observed object $x^{obs}$ is first compressed using a 3-layer dense network, with trained weights $f_{decompress_1}$ and $f_{decompress_2}$. This converts the original 45-element one-hot encoded vector into a vector with 10 elements and is done for the sake of computational efficiency. The compressed sensorium are then temporally filtered and split to provide 5 equal streams of input to the hippocampus. The MEC representations $g$ are also split into 5 (unequal) streams and down-sampled to remove 2/3 of the data. Details of the equations used in these processes can be found in the generative section whilst details of the size and use of streams are given when they are used, in the inference section.

| Steps | Equation |
|---|---|
| Compress sensory object | $x_t^c = f_c(x_t)$ |
| Temporally filter | $x_t^f = (1-\alpha^f)x_{t-1}^f + \alpha^f x_t^c$ |

$f_c(\cdot) =$ compression via 3-layer dense network (1 hidden layer).
$f_{down}(\cdot) =$ down-sampling from first $\frac{1}{3}$ of $g$.
$f_d(\cdot) =$ decompression via 3-layer dense network (1 hidden layer).
details of the $attractor(\cdot)$ function is detailed below

Each stream is kept separate throughout training, having it's own network weights and biases, and are only re-combined within the attractor to retrieve hippocampal memories $p$. Within the attractor, the streams are stacked together and interact with one another in a hierarchically determined manner, discussed in detail in the implementation section. This hierarchy is only used when the attractor is retrieving memories from an input from the MEC,

when sensory observations from the LEC are passed into the attractor, each frequency interacts equally with every other module. Nevertheless, it has a pervasive impact on the performance of the model and therefore is of great influence to the results, as discussed later.

The reasoning for this hierarchical interaction of streams in the entorhinal cortex is to encourage faster learning. By allowing higher frequency modules to interact with lower frequencies, one enables the agent to use a course-level understanding of the environment to influence it's understanding and a finer scale. This is also a biologically-inspired design choice, effectively producing modules that exhibit grid cells at different spatial scales, which is consistent with observations across both grid cells [32] and place cells [33]. We note however, that the forced nature of these discrete frequency modules is not a biologically plausible implementation, as this aforementioned spatial scale should emerge naturally.

### 2.4 Step Update

TEM learns sequentially, step-by-step, in one environment after another. As this proceeds, the agent generates an internal representation of its environment that more closely approximates it's environments structure. The core of this internal representation is $g$, or the abstract representation of grid cells. We believe TEM to be best described using a step-update module that links the aforementioned generative and inference models. The step update is responsible for updating the MEC representations, $g^g$ and $g^i$, produced by both models respectively.

For $t = 0$, a generative prior with set standard deviation is used as an initialisation for the entorhinal representation, $g$. At each time step thereafter, the generative model takes as input the previous $g_{t-1}$ and the action taken $a_t$. The action-dependent transition matrix is computed, via a series of 3 dense layers (one hidden layer), denoted as $W_a$. The size of this hidden layer is chosen to be $d_{hidden} = 15$ in order to optimally relate the opposing directions of movement (i.e. East and West are related (opposite), as are North and South, but not North and East).

This path-integrated $g^g$ is then used as a component in the inference of $g^i$. A hypothesis $g^i$ is inferred directly from memory $p_t$ and combined with $g^g$ to generate the final candidate for the EC representation $g$. Each $g$ has a weighting $\sigma$ associated with it, which determines how heavily each influences the final MEC representation. This is done such that early on in the trajectory, the inferred $g^i$ has a reduced effect and the path-integration takes charge; the reasoning for this is simply that the accuracy of early memories is particularly poor. A combined scaling weight $\sigma$ is generated from the sigma for both inference and path integration, $\sigma_i$ and $\sigma_g$ respectively...

$$\log(\sigma) = -\frac{1}{2}\log\left[\frac{1}{\sigma_i^2} + \frac{1}{\sigma_g^2}\right] \tag{1}$$

... and this is used to calculate $g$, which is what is ultimately used to plot the grid cells you see in fig. 2.

$$g = \sigma^2\left[\frac{g_i}{\sigma_i^2} + \frac{g_g}{\sigma_g^2}\right] \tag{2}$$

A mask is then applied to this matrix and it is multiplied by the $g_{t-1}$; this mask has a hierarchical nature and serves the purpose of further enforcing the hierarchy, something that is not discussed in the original paper. The result is then added to the old entorhinal representation to update it, and the whole thing is passed through a leaky ReLU activation function, that thresholds at $\pm 1$, $f_g(\cdot)$ as follows:

$$g_t = f_g(g_{t-1} + W_a g_{t-1}) \tag{3}$$

Path integration involves making predictions of preceding cognitive maps, without the use of any external input (imagine finding your way around your kitchen in the dark). We believ this process should be considered separately from the generative model and is implemented so in both the original code-base as well as in our framework. We note that in the original paper, path integration is probabilistic, with a distribution on $g$ being sampled in order to generate the next $g^g_{t+1}$. The theory in the paper is consistent with this Bayesian approach yet the implementation remains strictly deterministic. This may lead to confusion surrounding the mathematical equations as well as the variables that are used throughout TEM; for clarity, we outline both equations and variables in a manner that parallels their implementation in the code.

#### 2.4.1 Generative Model

A generative model typically makes predictions of observations based on latent variables. Here, the generative model makes predictions of sensory objects $x$ from latent variables of the medial entorhinal cortex, $g$. For a correct prediction of $x$ to be made, a memory $p$ must be retrieved that correctly links the current MEC representation with a sensory observation. This is done via an attractor network, which is described graphically in fig. 3 and will be discussed in detail later.

To fit the size of the HPC, $g$ is first down-sampled to one third of it's original size and then multiplied by a fixed weight $W_{repeat}$ that copies $g$ until it has the same dimensions as $p$. Down-sampling is done in the pursuit of computational efficiency, and also encourages the learning of more general abstract representations. Once this processing is complete, $g$ is passed into the attractor network, where a corresponding memory $p$ is retrieved.

As mentioned above, until this point the representations of TEM have been kept in 5 separate streams, each with it's own network weights and learned representations. Whenever a representation, be it abstract of sensory, is passed into the attractor, these streams are stacked together and interact in a specific way. In this case of input from the MEC, streams interact hierarchically, in a manner identical to the mask discussed in the previous section, with higher-frequency streams interacting with lower-frequency streams but not the other way round. The reverse process is discussed in the section detailing the attractor dynamics.

Memories $p^g$, retrieved from the HPC, are then tiled using another fixed weight $W_{tile}$ to the dimensionality of sensory observations and passed through a $softmax(\cdot)$ function. The resulting sensory predictions are decompressed using a similar 3-layer dense network that is used to compress sensory observations $x^{obs}$. In order to produce a suitable loss function, multiple $x$ are generated, using various EC and HPC representations.

| Steps | Equation |
|---|---|
| Path Integration | $g_t^g = f_g(g_{t-1}^i + f_{W_a}(g_{t-1}^i)$ |
| EC input to the HPC | $\tilde{g}_t = W_{repeat} f_{down}(g_t^g)$ |
| Retrieve memory | $p_t^g = attractor(\tilde{g}_t, M_{t-1})$ |
| Sensory Prediction | $x_t^g = f_x(W_{tile}^T p_t^g)$ |

$f_g(\cdot) =$ activation thresholding at $\pm 1$
$f_{down}(\cdot) =$ down-sampling from first $\frac{1}{3}$ of $g$
$f_x(\cdot) = softmax(f_d(\cdot))$
$f_d(\cdot) =$ decompression via 3-layer dense network (1 hidden layer)
details of the $attractor(\cdot)$ function is detailed below

#### 2.4.2 Inference Model

The **inference model** is responsible for making predictions of $g$, using the true sensory observation $x_{obs}$, via an attractor network. It also updates the Hebbian weights $M_t$, which are analogous to the hippocampal memories, using this inferred $g^i$ and $x_{obs}$. The inference model makes these predictions of $g$ using a combination of the memory inferred from the observed sensory object, $p^x$, and the path-integrated $g^g$.

The inference model contains two main components, the first of which is responsible for inferring the medial entorhinal representation, $g$. The first step here is to pass the sensory input to the hippocampus. This involves compressing the observed sensory object $x$ and then splitting it into multiple parallel streams. Each stream can learn different weights and so represents the world at different scales, each stream is temporally smoothed separately using the following equation

$$x_t^f = (1 - \alpha^f)x_{t-1}^f + \alpha^f x_t^c \tag{4}$$

This compressed and filtered observation is normalised by passing it through a rectified linear activation $f_n(\cdot)$ and then a unit normalisation. The result of this is scaled by a learned factor of $w_p$ and passed into a tiling function. This tiling function is equivalent to multiplying by the fixed weight $W_{tile}$ that increases the dimension to match that of the hippocampus.

$$\tilde{x}_t = W_{tile} w_p f_n(x_t^f) \tag{5}$$

This input to the hippocampus can be used to retrieve a corresponding memory of where the agent is, i.e. it's abstract representation $g$. The sensory input is passed into an attractor network which returns the corresponding hippocampal representation, $p_t^x$; this in turn allows the inference of our entorhinal representation. This inference is simply the application of a fully-connected network to the retrieved memory $f_p(p^x)$, followed by an activation that thresholds at $\pm 1$.

The second component of the inference process is to infer a hippocampal representation from $g^i$ and $x_{obs}$ to update Hebbian weights. Similarly to before, the EC representation has to be scaled to the size of the hippocampus, this is done by splitting it into frequencies again, sub-sampling $g$ to remove $\frac{2}{3}$ of the entries and multiplying it by the fixed weight $W_{repeat}$ such that $\tilde{g}_t = W_{repeat} f_{down}(g_t)$. The MEC representation and LEC input are then joined by simply multiplying them together and the result is passed through a leaky ReLU activation that is additionally thresholded at $\pm 1$.

We note here that, similarly to the step update above, the mathematics used to describe this inference process is Bayesian in nature. The problem of inference is therefore discussed using a variational autoencoder (VAE) framework which may cause confusion considering the implementation is again deterministic. In the section 2.4.2 below, we aim to outline the equations as they are used in the implementation.

| Steps | Equation |
|---|---|
| Sensory input to HPC | $\tilde{x}_t = W_{tile} w_p f_n(x_t^f)$ |
| Retrieve memory | $p_t^x = attractor(\tilde{x}_t, M_{t-1})$ |
| Infer EC | $g_t = C\left(g_t^i = f_g(f_p(p_t^x)), g_t^g\right)$ |
| EC input to HPC | $\tilde{g}_t = W_{repeat} f_{down}(g_t)$ |
| Infer HPC | $p_t^i = f_p(\tilde{g}_t \cdot \tilde{x}_t$ |
| Form memory | $M_t = hebbian(M_{t-1}, p_t^i)$ |

$f_n(\cdot) =$ ReLU + L2 normalisation

$$C(g^i, g^g) = \sigma^2 \left(\frac{g^i}{\sigma_i^2} + \frac{g^g}{\sigma_g^2}\right); \left[\frac{1}{\sigma_i^2} + \frac{1}{\sigma_g^2}\right]^{-1}$$

$f_p(\cdot) =$ leaky ReLU activation (thresholding at $\pm 1$)

both the $attractor(\cdot)$ and $hebbian(\cdot)$ functions are outlined in more detail below

### 2.5 Optimisation

The training of TEM can be separated into two recurrent loops, with one nested inside the other. At the highest level are the 'slow' updates of *network weights* and nested within these are the fast *Hebbian weight* updates. The former weights are what update the fully-connected layers between different representations within TEM whilst the latter are confined to the attractor.

TEM can thus be seen as one large ANN containing all the constituent processes described above. As these processes are all differentiable, training of 'slow' network weights is done via back-propagation and the passing of gradients through this large ANN. Hebbian weight updates constitute a process of their own, within this larger update process, and are thus seen as a 'fast' inner loop which learn the structure of a particular environment. Sensory predictions, and their associated losses, are used to update both the slow loop directly, and the fast loop via the inference of correct memories.

This distinction of slow and fast weight training is useful in understanding the primary results of the Tollman-Eichenbaum machine. Whilst network weights are responsible for generalisation across changing environments, and therefore the grid-like representations of the MEC, the fast Hebbian weights reflect learning within a particular world. This learning relies on accurate memory-retrieval, and therefore reproduces representations associated with the hippocampus and is key to accurate zero-shot prediction.

#### 2.5.1 Network weights

These are what allow the agent to generalise across environments, and improve the speed of learning with each training iteration. They are updated, via back-propagation through time (BPTT), every time the agent reaches the end of a block of 25 steps. A full list of the trained network weights is given in table 1, we note that these weights differ to those originally described in the TEM paper.

The updating of these weights corresponds to an 'outer loop' that allows the agent to learn across environments, and thus improve speed and accuracy of future predictions. Figure 4 shows this simplified outer loop, starting with the prior of $g$ at the first time-step, $t = 0$.

#### 2.5.2 Attractor Dynamics

TEM is able to extract 'memories', given either a sensory observation or EC representation. Retrieving memories enables the agent to infer where it is, based on what it sees, or vice versa. The mechanism by which memories are retrieved utilises an associative memory system called an *attractor network*. This memory system uses the fast Hebbian updates mentioned above to make improving predictions of future sensory observations, the mechanism of this update is detailed in this section.

The memory $p^x$ is represented by a stationary point, or attractor, reached by the network and represents the most likely pair of sensory observation $x$ and abstract representation $g$. These point-attractor networks are essentially

| **Weight** | **Function** | **Model** | Name in Code |
|---|---|---|---|
| $w_x$ | scales from $p$ to $x$ | generative | w_x |
| $f_{decompress}$ | revert from two-hot to one-hot encoding | generative | f_decompress_1/2 |
| $W_{tile}*$ | scales between $x$ to $p$ | inference | tf.tile($\cdot$) |
| $w_p$ | scales from $x$ to $p$ | inference | w_p |
| $f_{compress}$ | weights for compression of $x$ | inference | f_compress_1/2 |
| $w_\sigma$ | weights for inference $g_t^i$ | inference | sig_p2g |
| $w_\mu$ | weights for inference of $g_t^i$ | inference | mu_p2g |
| $\alpha^f$ | temporally filter $x$ | inference | w_smooth_freq |
| $W_{repeat}*$ | scale from $g_g$ to $p$ | step update | tf_repeat_axis_1($\cdot$) |
| $\sigma_g$ | contribution of $g_t^g$ to $g_t^i$ | step update | sig_g2g |
| $\mu_g$ | weights for the second layer of $W_a$ MLP | step update | mu_g2g |
| $d_g$ | weights in first layer of $W_a$ MLP | generative | d_mixed_g2g |
| $\mu_{prior}^g$ | weights of $g$ prior | step update | mu_g_prior |
| $\sigma_{prior}^g$ | bias of $g$ prior | step update | logsig_g_prior |

Table 1: Network weights ($*$ indicates fixed weights).

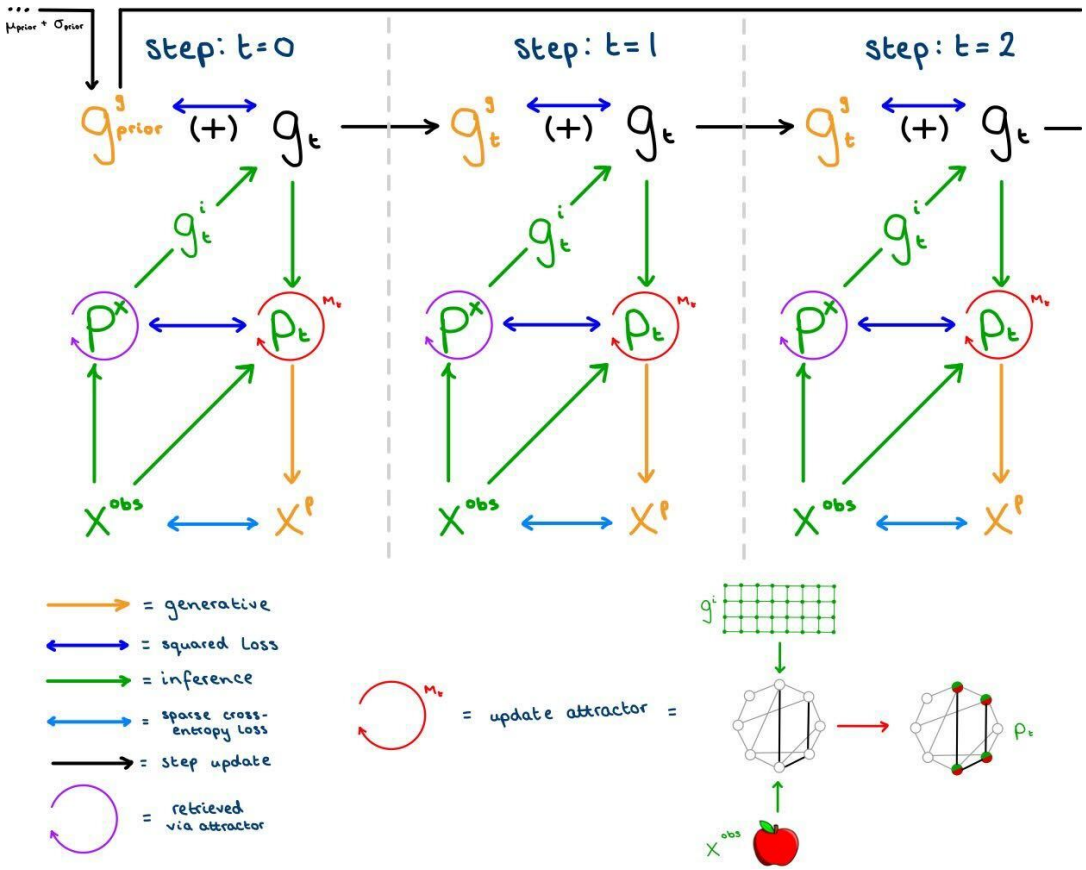


Figure 4: **TEM Loop Structure:** A simplified version of the TEM structure is given for each time-step, ignoring some of the additional variables that are generated for use in the calculation of losses. At the start of each chunk of 25 steps, a prior on $g^{prior}$ is used instead of the path-integrated $g^g$; this prior is learnt via the network weights and biases $\mu_{prior}^g$ and $\sigma_{prior}^g$ via BPTT, at the end of each chunk. The bottom-right pane shows the process of updating the attractor network weights.

Hopfield networks [35] and follow many of the same dynamical rules; therefore, using a Hebbian learning rule, the network learns to associate a presented stimulus with the memory that is it's closest match. A graphical description of this process is given in the top-right pane of fig. 3.

**Hebbian weights** help the attractor networks learn to associate $x$ or $g$ with the correct memory $p$. A memory is stored as the Hebbian weights, $M$, of a network once it has reached a stationary state. These updates operate on a different time-frame to the rest of the weight updates in TEM and should therefore be discussed in detail separately. Whilst most network weights in TEM are updated after each batch of 25 steps, the Hebbian weights are updated at

every step. This is what enables the agent to learn the structure of an environment and therefore to make correct predictions.

Another important factor to consider here is the quality of retrieved memories. Given the fact that these Hebbian weights are reset at the start of each environment, early memories are by nature less accurate than those retrieved later on. To correct for this, the work of Ba et al. (2016) [36] is used to attend to the resent past using fast weights. These fast weights also fit within the Hopfield paradigm by utilising an outer product rule to store hidden activity vectors, $h(t)$. This ensures that the agent preferentially uses recent memories in it's predictions over the memories produced early on. The details of this will be discussed in the implementation section.

The weights are updated using the Hebbian learning rule outlined by Ba et al. (2016)...

$$\mathbf{M}(t) = \lambda \mathbf{M}(t-1) + \eta \mathbf{h}(t)\mathbf{h}(t)^T \tag{6}$$

... where $\mathbf{M}$ denote the memories, $\mathbf{h}(t)$ are the hidden vectors at time $t$ and $\lambda$ and $\eta$ are constant rates for forgetting and learning respectively. The hidden states $\mathbf{h}(\tau)$ of TEM differ to that of the Ba et al. paper in that two different hidden vectors are used. We denote them as $\mathbf{m}(\tau)$ and $\mathbf{n}(\tau)$ such that $\mathbf{m}(t)\mathbf{n}(t)^T = (\mathbf{p} - \mathbf{p}^x)(\mathbf{p} + \mathbf{p}^x)^T$ where $\mathbf{p}$ are the memories produced from the inference model and $\mathbf{p}^x$ are those extracted, by the attractor, at each hidden step.

This ensures that the attractor never embeds a memory if it has seen it before (and therefore $\mathbf{p} = \mathbf{p}^x$), encouraging it to learn the whole environment equally. Memories are retrieved at every step in the same way as in Ba et al. (2016)...

$$\mathbf{M}(t+1) = \lambda^t \mathbf{M}_0 + \eta \sum_{\tau=1}^{\tau=t} \lambda^{\tau - t}(\mathbf{p}^x - \mathbf{p})(\mathbf{p}^x + \mathbf{p})^T \tag{7}$$

... where $\tau$ is the iteration of the attractor and $\eta$ is a decay term. This number of iterations depends on the stream being processed, and ranges from 1 and 5 for the lowest and highest stream in the hierarchy, respectively. The purpose of using this learning rule is to attend to the past memories in proportion to their scalar product with the current memory; this encourages recent memories to have a disproportionately strong impact on the updates, and ignores early memories which are by nature less accurate.

The details of the inner loop of TEM, including network and Hebbian weights, are given in fig. 5 below.

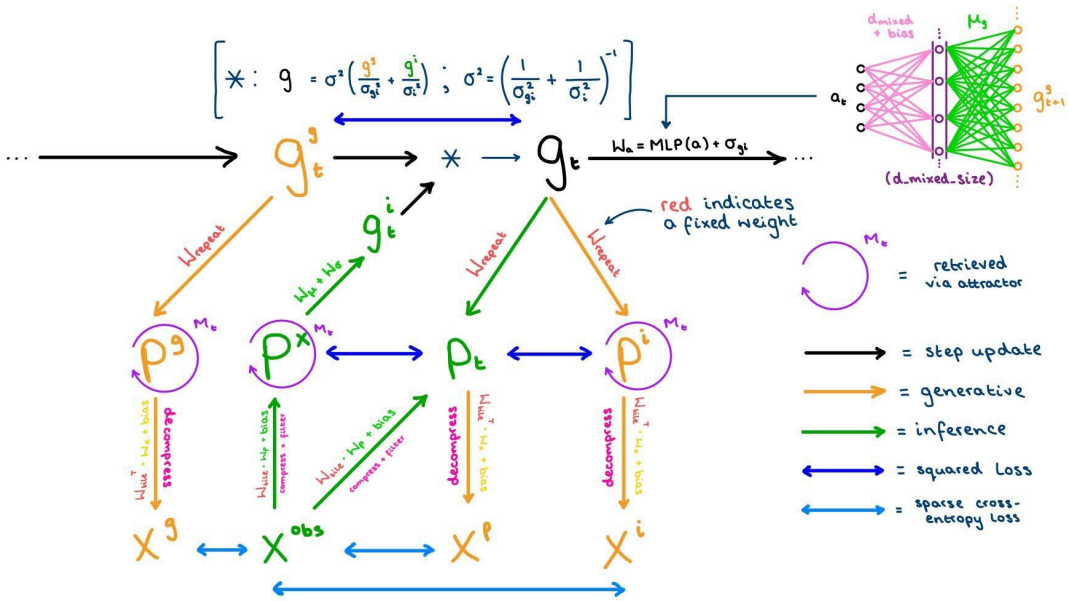


Figure 5: **TEM Structure:** The generative and inference models, connected via the step update, are described in greater detail. Annotations along each arrow denote the weights that are associated with the each fully-connected layer or MLP. A detailed diagram of the weight-dependent MLP used in path integration is given in the top-right. The equation that combines $g^g$ and $g^i$ to give the final EC representation $g$ is given at the top.

### 2.6 Losses

The original Tollman-Eichenbaum machine is considered as a probabilistic model, and therefore the discussion of optimisation primarily focuses on the problem of maximising an evidence lower bound (ELBO) on $\ln p_\theta(\mathbf{x}_{\leq T}|\mathbf{a}_{\leq T})$ where $\theta$ denotes the generative or inference model. This is solved using a variational autoencoder framework [37].

Whilst it is noted in the paper that in implementation, the problem of optimisation is strictly deterministic and thus there is no need for this Bayesian complexity, a satisfactory description of the additional weight generation is missing. We hope to clarify this aspect of the optimisation. Losses are calculated at each level of TEM representation; there are comparisons made between abstract EC representations $g$, conjunctive HPC representations, $p$ and sensory observations and predictions $x$. Each of these losses are highlighted in blue in fig. 5.

To calculate the accuracy of sensory predictions, three additional hypotheses for the next seen object are generated. One is generated from the inferred memory $p_t$ whilst two more are generated directly from path-integrated and inferred EC representations, $g^g$ and $g^i$, respectively. All three of these predictions are compared to the ground-truth observed $x^{obs}$ and summed to give the total sensory loss $L_x$. The HPC memory that is used to generate $x_{gi}$ is also compared with $p_t$ to give a memory loss $L_p$. An EC loss, $L_g$, is also calculated from the path-integrated and inferred $g$s. We note that in the original paper, losses are described as being generated by current and previous EC representations $g_t$ and $g_{t-1}$. In reality sensory predictions $x^g$ and $x^i$ are generated from the current $g^g$ and $g^i$ respectively.

$$L_x = L_{x_g} + L_{x_p} + L_{x_{gi}} \tag{8}$$

$$L_p = L_{p_t} + L_{p_g} \tag{9}$$

Losses on both hippocampal memories $p$ and MEC representations $g$ are *squared error* losses, shown below in eq. (10). The losses calculated between sensory predictions are determined using a sparse cross-entropy equation, given in eq. (11). Whilst squared error simply uses the difference between each variable, cross-entropy compares each prediction with a truthful value, in this case $x^{obs}$.

$$L_p = \frac{1}{2}\sum_{c=1}^{C}(p_c - p_c^x)^2 \tag{10}$$

$$L_x = \sum_{c=1}^{C} x_c^{obs} \cdot \log(y_c) \tag{11}$$

In the above equations, $C$ is the size of each batch, such that $x^{obs} \in \mathbb{R}^C$. The cross-entropy loss uses the softmax equation to produce logits of predicted observations such that $y_c = \frac{\exp(x_c)}{\sum_{d=1}^{C}\exp(x_d)}$.

## 3 Environment

### 3.1 TEM Environment

TEM operates in a custom built grid-world, that varies in size and shape depending on the parameters chosen. Whilst for these purposes, we are only interested in a simple 2D square, TEM is capable of generalising across many different environments such as hexagonal and hierarchically organised graphs. Regardless of the environment, TEM uses a continuous trajectory through a discrete grid world in order to make observations of it's environment.

The method by which these trajectories are generated uses a pair of adjacency and transition matrices. The former of these contains information on which states neighbour each other, whilst the latter gives the probabilities with which the agent can move between available states. These are generated using details of the environment, such as width and height, as well as from the agent, i.e. it's policy. In the case of TEM , this policy is random and all of these processes of environment generation are done within the main `run` file.

Each state, or node, within this grid world is associated with a number between 0 and $\max(n_states)$. The agent uses the number unique to a position in it's environment in order to observe an object, represented as a one-hot encoded vector ($[0, 1, 0, ...]$ for apple). This enables the model to put together a map that associates actions with observations, thus make predictions on what it will see next. A graphical example of how TEM moves through and between different environments is given below in fig. 6.

TEM is also capable of a host of additional behaviours that aren't included in this implementation. One can also introduce 'shiny' objects to the agent's environment, that act as attractors, influencing TEM's trajectory. The inclusion of these objects encourages the model to reproduce object-vector-cell like representations, expanding the collection of neural phenomena that TEM is able to account for. The spatial environments that we are primarily concerned with can also be replaced by hierarchical, non-spatial, environments such as family trees. TEM is able to navigate these and perform transitive inference, understanding the relationships between nodes of the graph (for example, the brother of my mother is my uncle). These additional features are compelling, and we hope they are able to be added to this implementation in the future.

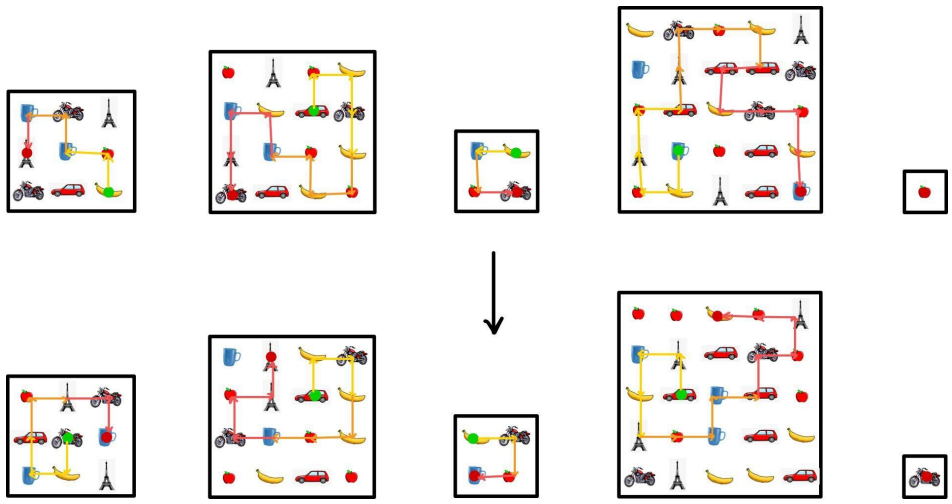

Figure 6: **TEM square grid world:** TEM starts at (●), moves from one discrete state to the next, observing the object associated with that state at each time-step and finishes at (●). The diagram shows two training iterations, each containing environments of differing sizes and containing randomly distributed sensory objects. The agent trajectory is made up of smaller 'chunks' of steps (yellow → red), it is at the end of each of these steps that the network weights are updated.

### 3.2 NeuralPlayground Environment

Similar to the original TEM, our environments are simple 2-dimensional squares, divided into smaller evenly-sized states. Each of these states contains a sensory observation (apple, chair, motorcycle etc.), that is encoded as a one-hot vector. At each time-step, the agent makes discrete steps in the environment and sees the object that corresponds with the state it is in. Whilst the policy that this agent follows is random, similar to that of TEM's random walk, one could conceivably introduce a policy that was dependent on the internal state of the agent. This is possible because, unlike the original implementation, this environment can update the policy of the agent in real-time, allowing the output of the agent to influence it's action-making policy.

This differs from the original TEM environment by being a (near) continuous space. The steps the agent makes are variable and thus can be reduced arbitrarily, to approximate a continuous mouse trajectory. The 2D arenas are also easily customisable, to produces more complex architectures such as T-mazes or merging rooms (see fig. 7). These environments can also be initialised with data from an experiment; one can simply replace the policy of an agent with the positional data from a corresponding experiment.

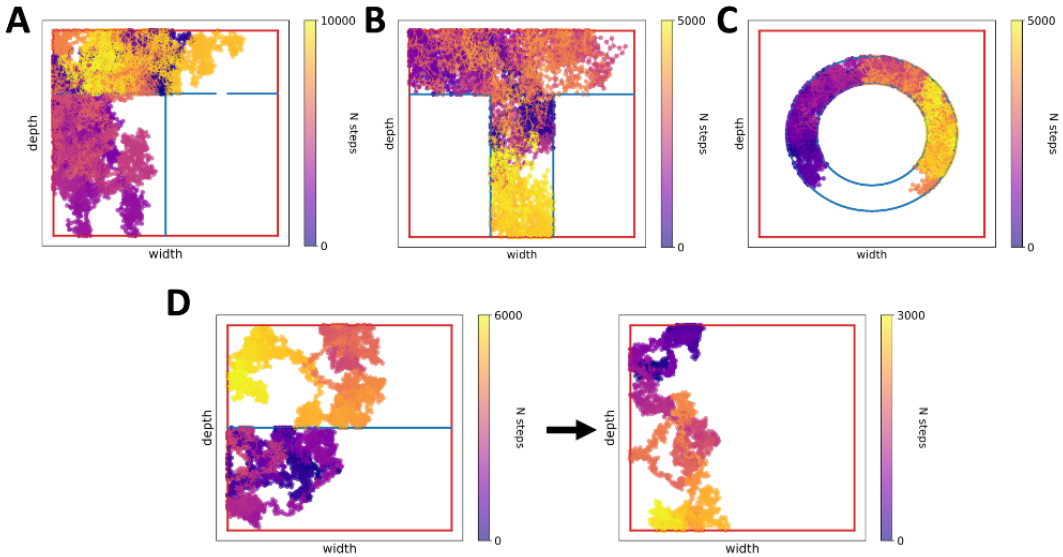


Figure 7: Agent navigating under random walk policy in common environment types found in HEC-experiments **A:** Connected rooms **B:** T-Maze **C:** Circular **D:** Merging Room: The room is originally separated by a wall, than is later removed.

In addition to the environments that are used in the original TEM paper, the NeuralPlayground framework includes some other virtual environments that have been instrumental in understanding spatial navigation. One example, which I was responsible for implementing, is the merging of environments [38]. In this case, the agent (or animal) is left to explore one subsection of a larger arena, that is initially hidden from view. After a set length of time, the barrier to the rest of the arena is removed, allowing the agent to explore the whole environment. See fig. 7 (D) for an example agent trajectory.

This type of arena has been used to observed integration of entorhinal grid cells in the rodent hippocampus [38], an example of the neural feature of remapping. Whilst the grid structure around the edges of the full environment are largely retained, those near the boundary undergo immediate relocation to bring about *"local spatial periodicity and continuity between the two original maps"*. For implementation, these dynamic boundaries inherit their basic nature from the simple 2D environment class but also include the option to place a custom wall, that is removed at a set time during the experiment.

The introduction of the Tollman-Eichenbaum machine to NeuralPlayground required several extensions to the capabilities of the environment class. To begin with, TEM was the first batch-based agent included in NPG and therefore the environment had to be able to produce positional information for multiple different environments simultaneously. The agent trajectory in TEM is also broken up into a series of shorter walks, each of these being the smallest unit of information handled by agent or environment. This required an extension to the original step-by-step process of previously-implemented models as the last state of the first walk must be saved in order to be used as the starting point for the second walk.

Implementation to our framework will hopefully be of benefit to those looking to investigate and improve TEM. It offers the opportunity to apply this model to novel environments and to unseen experimental data, whilst also enabling the user to dictate the agent's behavioural policy. We hope this flexibility will encourage further exploration and understanding of the methods of abstraction and representation that make TEM such an influential model.

## 4 Implementation

### 4.1 TEM & NeuralPlayground

The basis of our implementation is the interaction of two classes, one for the TEM agent and the other for our continuous environment; these are initialised in a main file and interact within nested training loops, one for the training iterations, and another for the sequence of walks within an environment. The parameters of both the model (learning rates and sizes of HPC/EC representations etc.) and environments (width and state density etc.) can be set prior to training, using a separate file that is passed as an argument to both classes during initialisation. All operations associated with the model are carried out within the agent class, as opposed to in the original implementation where compression and initialisation are done outside. Both the agent and environment classes inherit from core NPG classes and thus can be used in the context of any other experimental setup, accounted for by our test bed. The ability to train on batches was a novel contribution to the NPG framework, and will be useful in the future when running models on given experimental trajectories.

For each training iteration, the environment produces a trajectory of 25 steps, within the continuous environment; this trajectory is determined by the action policy, inherent to the agent class, which is used by the environment to determine the actions at each step. An example of these walks is given in fig. 8. The step size of the agent, as well as all parameters associated with the physical environment, are fully customisable. In the case of TEM, each batch consists of environments of varying sizes, encouraging the agent to learn the abstract structure, divorced from any notions of width and depth.

Figure 8 shows a random agent policy, as implemented in the original Tollman-Eichenbaum. In the case of a comparison to experimental data, one would simply match the layout of sensory objects and replace these positions with the known path of the animal. The agent then discretises these positions to produce a sequence of states, corresponding with each step in the path. Each of these states is then associated with a sensory observation, represented in the agent as a one-hot vector. It is this vector that is used by the model to generate internal representations and make predictions.

Here we note an interesting result that was encountered when introducing the TEM agent to NeuralPlayground. As mentioned in the Environment section, TEM uses a fully prescribed path, over approximately 250 walks, each of 25 steps, totalling an average trajectory of 6250 steps. This is passed into the agent in it's entirety at the beginning of each training iteration and then cut up into shorter walks. Our implementation approaches this differently, and generates one walk at a time, as the agent moves through it's continuous environment. We utilise an action policy that generates actions at each step, based on those available to it, whilst the original implementation uses a transition and adjacency matrix to select from available states.

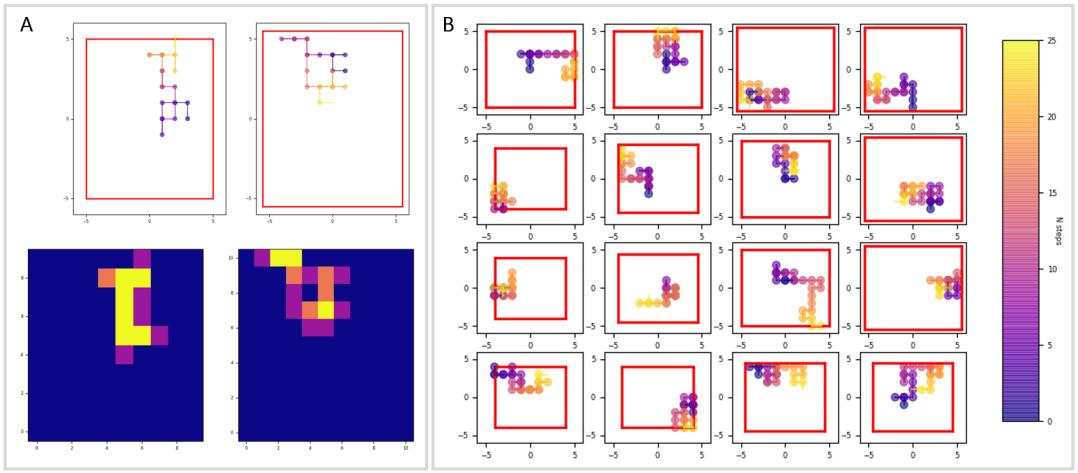


Figure 8: **NeuralPlayground Batch Environment: A:** A single walk of 25 steps, represented as agent trajectory (top) and the count of occupancy at each location (bottom). **B:** One walk taken by the TEM agent in a batch of 16 square environments. The size of environments varies, and is shown by the red border of each subplot.

As a result of this seemingly random agent policy and the symmetrical, bounded environments it operated in, patterns began to emerge in the agents position, when iterated over hundreds of steps. Figure 9 shows the emergence of these stationary state patterns, as one increases the number of steps within an environment. A key note here is that, under this action policy, when the agent attempts to step through a boundary, it instead remains in the same state.

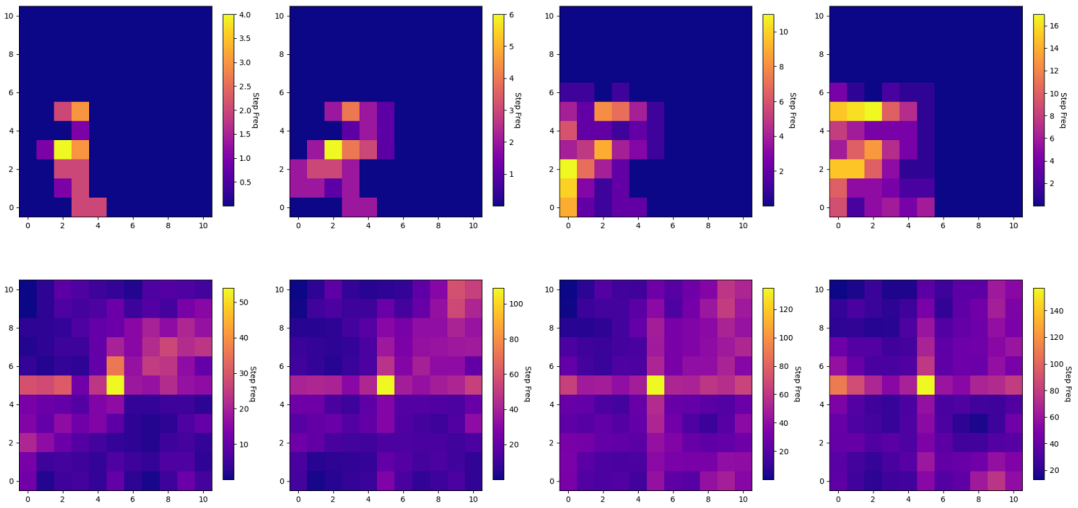


Figure 9: **Stationary State in the Action Policy:** An emerging stationary state in the 'random' walk of the agent, shown using the frequency of visits to each state at 1, 2, 5, 10, 50, 100, 150 and 200 walks (top left to bottom right).

A stationary state emerges in the centre of the arena, due to the symmetrical reflections off of boundaries on all sides. In order to solve this problem, the action policy was changed to select from an alternative available action, when an attempt is made to step through a boundary. The results are show in fig. 10.

It should be noted that these patterns in position within an environment had significant effects on the resulting grid cells and prediction accuracy. This is important as it suggests that TEM may be susceptible to even slight alterations in agent behaviour. This may become a problem when the model is inevitably tested against animal data, which is seldom random. It isn't surprising that positional regularities impact TEM's performance, as it's primary mode of understanding environmental structure is abstract spatial representation, $g$. However, animal paths often have regularities, not unlike those outlined in fig. 9, and these may cause TEM to struggle when generalising across environments.

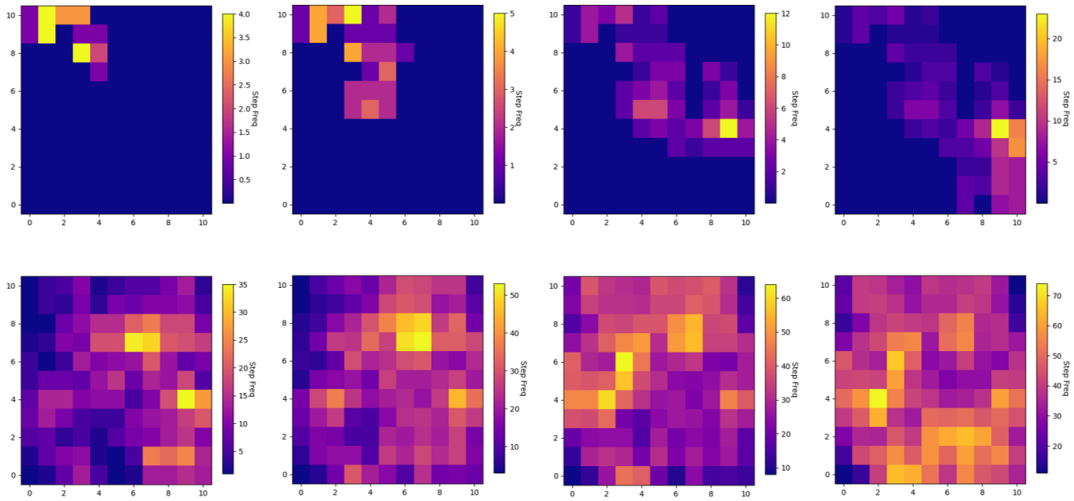

Figure 10: **Random Action Policy:** The random walk of an agent in the environment, shown using the frequency of visits to each state at 1, 2, 5, 10, 50, 100, 150 and 200 walks (top left to bottom right).

### 4.2 Model Implementation

Similarly to the original paper, and for computational efficiency, the calculations of TEM are done in batches. A single training iteration consists of a batch with 16 different environments, each of varying size and consisting of randomly distributed sensory objects. These batches are handled in parallel by the TensorFlow, improving speed of learning. This TensorFlow implementation generates a graph, prior to training that includes all the operations relevant to the desired output. Once this graph is created, values from the environment and the models internal states are simply passed in as inputs and resulting outputs calculated. This is much faster than re-initialising the graph structure at every training iteration and results in a 65% reduction in training time. Full training consisted of 300 environments, approximately 2000-5000 steps, and took on average 70 hours (with CPU). Training time could be shortened with the use of a GPU, as TensorFlow utilises the parallelisation of operations.

In order to further aid computation, the sensory data is compressed within the agent before being used in calculations. Upon observation, the sensorium handles each object as a one-hot encoded vector, containing 45 elements (allowing 45 discrete objects). The agent allows for two possible methods of compression, the first uses a look-up table to directly convert from a one-hot to a two-hot encoding, the latter containing only 10 elements. The alternative is to use a fully-connected layer to both compress and then later decompress $x$.

As discussed above, both sensory and EC representations are split into 5 frequency modules which are handled separately and interact with one another within the attractor network. These interactions are described using a $5 \times 5$ matrix of 0s and 1s, where each row denotes the interaction of that stream with the four other streams; 1 means an interaction, 0 means no interaction at all. For EC input to the hippocampus, a hierarchical matrix $\mathbf{H}_h$ is used...

$$\mathbf{H}_h = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

... such that higher frequency modules (lower rows) interact with lower frequency columns (higher rows) but not the other way round. These matrices are also used in the masks that are applied to $g$ during path integration, and hippocampal memories within the attractor. For sensory input to the hippocampus, these interactions are equal across all modules and so the matrix used is simply a $5 \times 5$ of 1s. There is evidence that the lateral entorhinal cortex receives information at hierarchical temporal scales [39] and it is therefore unclear why hierarchies are not implemented at the sensory input to the hippocampal attractor.

#### 4.2.1 Hebbian Updates

Whilst the theory of how the Hebbian weights of TEM are updated has been well described, the way in which these updates are implemented requires clarification. Hebbian updates are considered within a fast inner loop of weight updates, and are thus described separately from other network weights. It is important to note that, when implemented, the Hebbian weights are actually updated at the end of every walk, just like the network weights. This allows you to do computations without computing the full fast weight matrix at every step and is done by simply

appending memories to a list and then performing the outer product on the list, at the end of each walk. This is mathematically identical to the step-by-step update, described in eq. (7).

### 4.3 Variables & Naming Scheme

As mentioned in the description of the Tollman-Eichenbaum machine, much of the mathematical detail is described from the perspective of a probabilistic model. This makes naming of variables in the implementation, which is deterministic, at times opaque. Table 2 details the key variables used by TEM, along with their shape and function. We replace what were originally probabilistic terms, such as $\mu$ and $\sigma$, with terms that are more appropriate.

| | Shape | Name | Function |
|---|---|---|---|
| **Environment Variables** | $(16, 4*, 26)$ | $d_t$ | Direction of agent |
| | $(16, 2*, 26)$ | $a_t$ | Action of agent |
| **Sensory Observation (LEC)** | $(16, 45, 25)$ | $x_t^g$ | Sensory prediction generated from path-integrated $g^g$ |
| | | $x_t^p$ | Sensory prediction generated from inferred memory $p$ |
| | | $x_t^i$ | Sensory prediction generated from inferred $g^i$ |
| | | $x_t^{obs}$ | Observed sensory object |
| | $(16, 10, 25)$ | $x^c$ | Compressed observation |
| **Place Cells (HPC)** | $(16, 400)$ | $\mathbf{p}^g$ | Hippocampal memory generated from path-integrated $\mathbf{g}^g$ |
| | | $\mathbf{p}^i$ | Hippocampal memory generated from inferred $\mathbf{g}^i$ |
| | | $\mathbf{p}^x$ | Hippocampal memory inferred from observed sensory object $\mathbf{x}^{obs}$ |
| | | $\mathbf{p}$ | Hippocampal memory updated by $\mathbf{x}^{obs}$ and inferred $\mathbf{g}^i$ |
| **Grid Cells (MEC)** | $(16, 120)$ | $\mathbf{g}^g$ | Path-integrated MEC representation |
| | | $\mathbf{g}^i$ | MEC representation inferred from HPC memory $\mathbf{p}^x$ |
| | | $\mathbf{g}$ | Combination of $\mathbf{g}^g$ and $\mathbf{g}^i$, used to path-integrate |
| **Hebbian Networks** | $(16, 400, 400)$ | $\mathbf{A}$ | Path-integrated MEC representation |
| | | $\mathbf{A}_{inv}$ | MEC representation inferred from HPC memory $p^x$ |
| **Important Parameters** | | $\mathbf{n}_{walk}$ | The number of walks/chunks in a training iteration |
| | | $n_{step}$ | The number of steps within a chunk/walk |
| | | $n_{iters}$ | The number of training iterations |
| **Other** | $(16, 25)$ | $\mathbf{s_{vis}}$ | Whether a state in the trajectory has been visited (binary) |

Table 2: Important variables used in the Tollman-Eichenbaum Machine. $*$: action and direction size change depending on shape of environment.

There are also a host of constants that impact both the agent's behaviour (such as it's learning rate) as well as the environment it occupies (such as it's shape). It would be futile to attempt to describe all of these in this paper, however we list several important values here: the rate of Hebbian learning and forgetting both start low and rise to $\lambda = 0.9999$ and $\eta = 0.5$ respectively, $n_s = 45$ is the number of possible sensory objects and $n_{s*} = 10$ are their compressed size. The number of TEM entorhinal cells in each module are [30, 30, 24, 18, 18], and the number that project to the hippocampus, $n_f$ are [10, 10, 8, 6, 6] (hence the down-sampling of $\frac{1}{3}$).

## 5 Results

The aim throughout this thesis has been to implement the Tollman-Eichenbaum machine into our standardised environment toolkit, so that it could be further investigated with ease as part of a collaborative effort to better understand the HPC and EC. Above, we have provided a detailed description of TEM theory, in parallel with it's implementation to said framework. We hope this will encourage and ennoble scientists interested in the hippocampus and entorhinal cortex to use and extend this model, and to contribute novel results and understanding to this area of the brain and beyond.

### 5.1 TEM Implementation

The core aim of this paper revolves around successfully implementing the Tollman-Eichenbaum machine into our collaborative framework, so that researchers might use it to explore and expand upon the model in the future. Here we outline the major results of our implementation, and compare them to results generated using the original model.

The results of foremost interest here are the gird and place cells generated from abstract and memory representations $g$ and $p$, respectively. These are prime examples of TEM successfully reproducing important neural phenomena and are compared to results from the paper in fig. 11. We also include a comparison of the correlative plots of $g$, which includes the periodic repetition characteristic of grid cells.

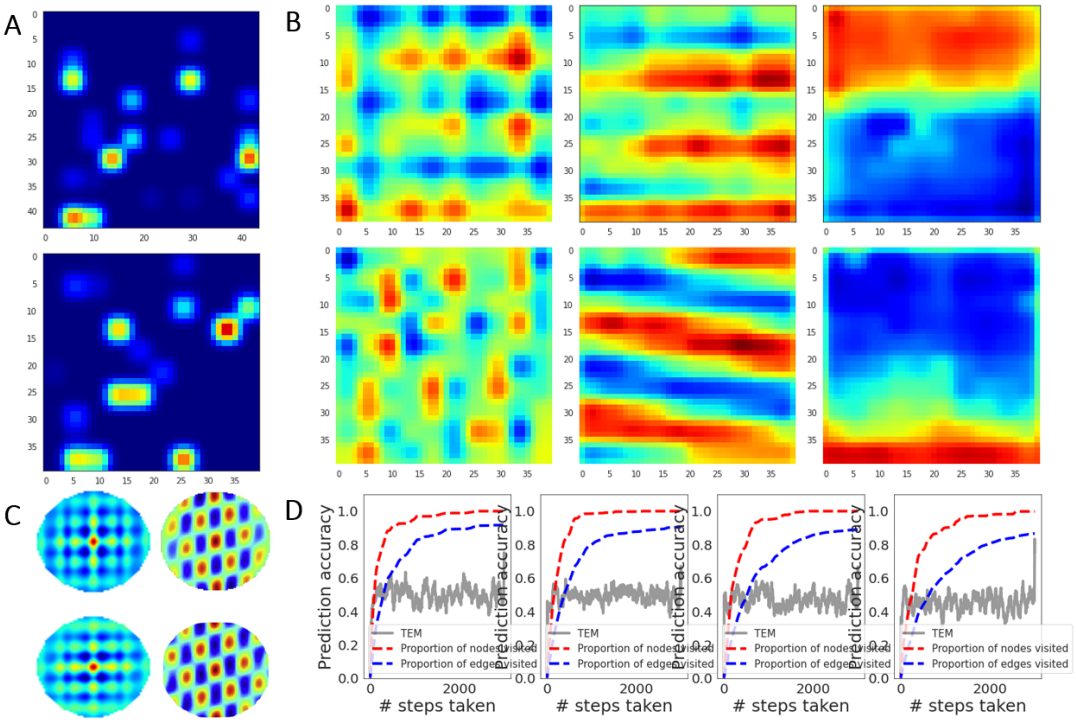


Figure 11: **Implementation Results: A:** place cells shown for two different environments. **B:** grid cells at high, medium and low frequencies (left to right), compared to those generated by the original code. **C:** autocorrelation plot comparison between implemented (left) and original (right) models **D:** Zero-shot prediction accuracy for 4 environments (grey) plotted against the proportion of nodes (red) and edges (blue) visited by the agent.

Another important result of TEM is it's ability to accurately perform zero-shot prediction, whereby the sensory objects at previously visited states are correctly predicted upon return. Figure 12 shows the prediction accuracy of our implementation compared to that of the original model. We note that, whilst accuracy is lower, a successful prediction is still made $\sim 60\%$ of the time. This indicates a still-impressive ability to generalise, considering the 45 possible sensory objects at each state.

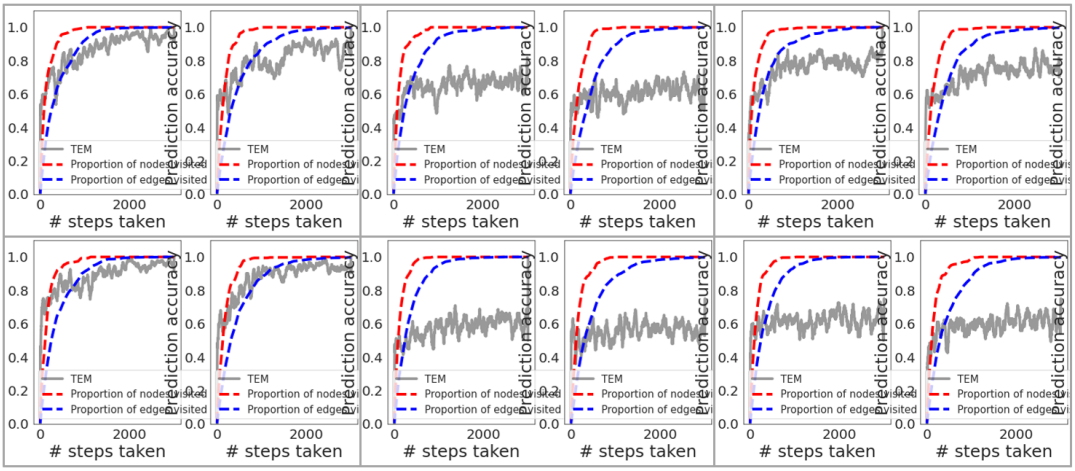


Figure 12: **Implementation Accuracy:** prediction accuracy from 6 simultaneous tests of the original TEM code. The results of two environments for each run are shown (hence 12 plots).

It is worth appending here that, whilst the accuracy of our implementation falls short of that outlined in the original TEM paper, the code used to generated these results has issues with reproducibility that are not shown in the original plots. The plot of prediction accuracy in fig. 12 shows this uncertainty by reproducing accuracy plots using the original TEM code, where prediction accuracy varies between $\sim 60$ and $\sim 95\%$. It should also be noted that the sample size used to generate this data is small, due to the length of time it takes to train the model to completion.

To conclude, it is clear that our implementation of the Tollman-Eichenbaum machine reproduces all of the neural features outlined in the original paper, relevant to our simplified version of it. We hope this implementation, and the results provided above, are used by those interested in probing the function of the hippocampus and surrounding systems. This implementation will be available as a part of our open-source framework once it is published (in October of this year). The GitHub for this implementation is available at https://github.com/LukeHollingsworth/Tollman-Eichenbaum-Implementation.

### 5.2 Ablation Studies

Due to TEM's powerful ability to both generalise across varying environments as well as to reproduce neural phenomena of interest, any progress in understanding the mechanisms that underlie certain results is of great value in uncovering mechanisms behind the HPC and EC. Here we provide some analysis that may be of use in answering key questions; these include whether structured representations, like grid cells, are necessary for successful generalisation and prediction, and what the minimal structure of TEM is that is still able to produce compelling results. We hope that these will aid future progress in understanding the function and properties of TEM and similar models.

The Tollman-Eichenbaum machine is an extensive and complex model, containing a host of trained and fixed weights, responsible for the transmission of representations from one level of the model to another. It is therefore unsurprisingly difficult to capture the full complexity of the learning process in a description of the model itself. As a result, description of specific weights that are used, and how they affect the resulting performance, are missing from the original paper. Below are a collection of ablation studies that are designed to both highlight the aforementioned discrepancies, and to expand upon the results of TEM already presented.

#### 5.2.1 Weight Ablation

Due to the large number of trained weights used by TEM, as well as the divide between slow and fast weight updates, we believe it crucial to highlight exactly what is trained and how. We have given a detail description of these weights above and here we demonstrate that without them, the model fails to generalise and produce grid cells. The weights explicitly outlined in the supplementary material of the original paper are given below, in table 3.

| | Name | Function |
|---|---|---|
| **Gradient descent weights** | $\mathbf{W}_a$ | Action-dependent transition weights |
| | $w_x$ | Scales sensory prediction |
| | $w_p$ | Scales sensory input to HC |
| | $\alpha^f$ | Temporal filtering parameter |
| **Fixed weights** | $\mathbf{W}_{repeat}$ | Repeat each element of vector in place a specified number of times |
| | $\mathbf{W}_{tile}$ | Make copies of vector and concatenate |
| **Hebbian weights** | $\mathbf{M}$ | Store memories, used in attractor network |

Table 3: The weights outlined in the extension to the supplementary material of Whittington et al. 2020 [4].

To test these weights, we edited the original code to ensure that only those weights given in the table above were trained. The resulting grid and place cells are given in fig. 13 and are compared to results produced using the original code.

It is clear that even after a full training period. Ablation of the additional weights eliminates TEM's ability to produce both grid and place cells. This inability to generalise is also captured by the correlative plots in fig. 14, where the periodic nature of grid cell correlation is completely removed. The plots show the correlation of the two abstract representations $g$, where correlation $z$ is defined as...

$$z[k] = (x * y)(k - N + 1) = \sum_{l=0}^{\|x\|-1} x_l y^*_{l-k+N-1} \tag{12}$$

... for $k = 0, 1, ..., \|x\| + \|y\| - 2$, where $\|x\|$ is the length of $x \in \mathbb{R}^{45}$, $N = \max\left(\|x\|, \|y\|\right)$, and $y_m$ is 0 when $m$ is outside the range of $y$.

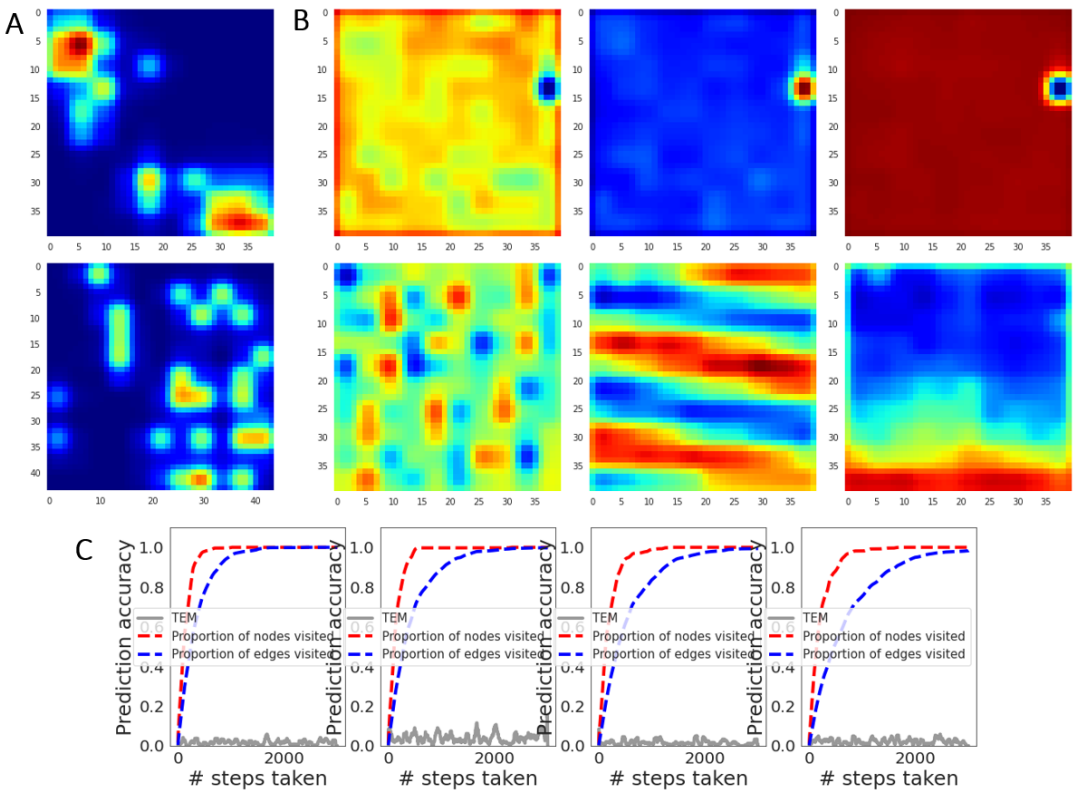


Figure 13: **Weight Ablation:** Effects of only training the weights and biases explicitly outlined in Whittington et al. 2020 [4]. **A:** place cells shown for two different environments. **B:** grid cells at high, medium and low frequencies (left to right), compared to those generated by the original code. **C:** Zero-shot prediction accuracy for 4 environments (grey) plotted against the proportion of nodes (red) and edges (blue) visited by the agent.

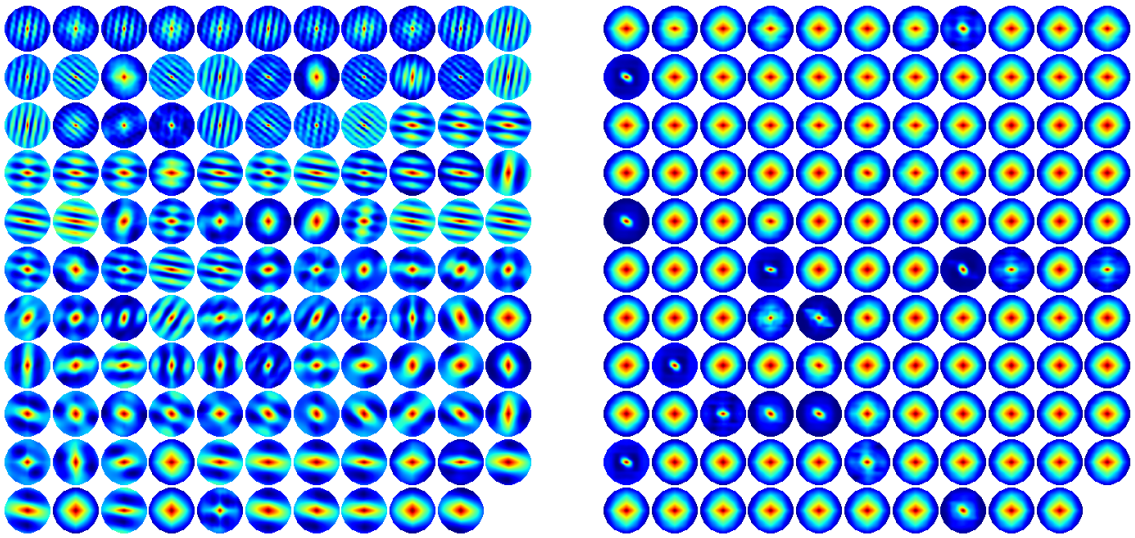

Figure 14: **Weight Ablation:** Correlation between grid cells across environments, produced when training on the full set of weights (left) and only those weights mentioned in the original paper (right). Red represents a high level of correlation, blue a low level.

Here we also investigate a specific weight that is trained in TEM yet not discussed, that is the *prior* on $g$. As mentioned in the description of the step update process, a prior on $g$ is used as an initialisation at the start of each walk. This prior has learned weights and biases that are not discussed in the original paper at all. Figure 15 shows the resulting grid and place cells, as well as prediction accuracy, of the original Tollman-Eichenbaum model when $g_{prior}$ is not trained.

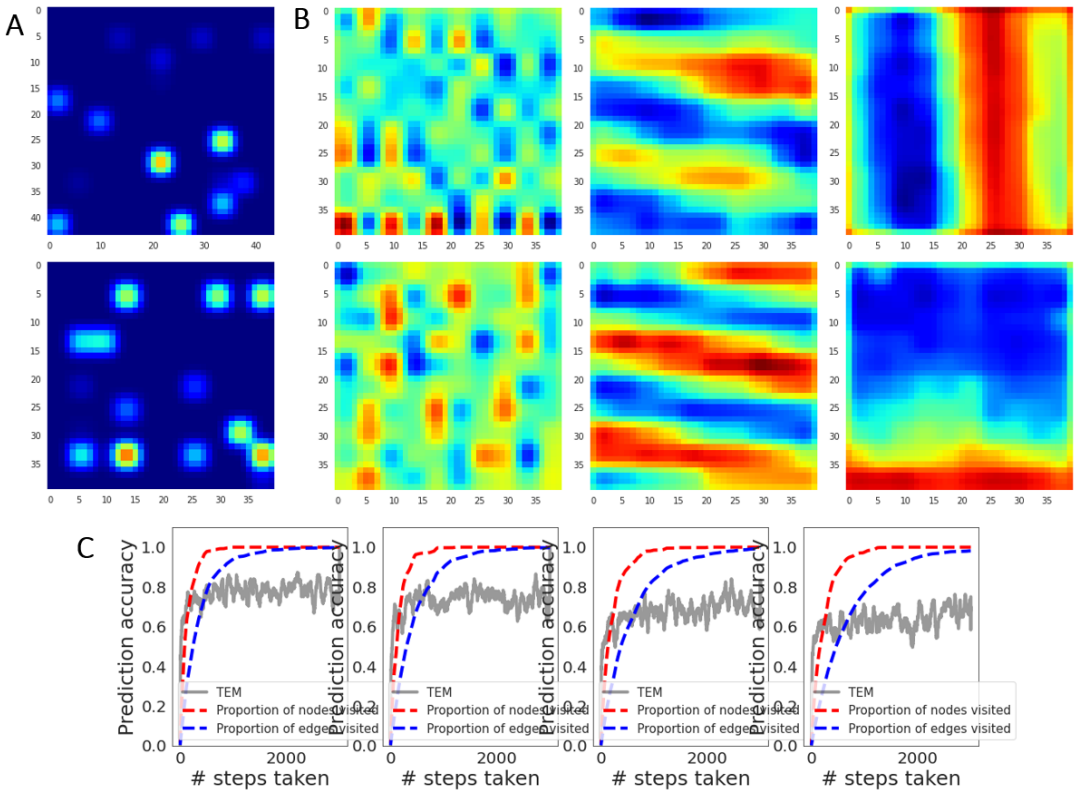


Figure 15: **Ablation of Prior on $g$:** Effects of not training the weights and biases on $g_{prior}$. **A:** place cells shown for two different environments. **B:** grid cells at high, medium and low frequencies (left to right), compared to those generated by the original code. **C:** Zero-shot prediction accuracy for 4 environments (grey) plotted against the proportion of nodes (red) and edges (blue) visited by the agent.

As a result of the ablation of $g_{prior}$, the high-frequency grid cells are less clear than in the original, yet the medium and lower frequency spatial representations appear unaffected. This suggests the learning of a prior on $g$ influences TEM's ability to reproduce grid cells. The model's prediction accuracy is also negatively affected, decreasing by $\sim 20\%$, however the reliability of this result requires clarification due to uncertainty associated with the original models accuracy. This is discussed in more detail later.

### 5.2.2 Hierarchical Ablation

As mentioned above, the Tollman-Eichenbaum machine utilises spatial hierarchies when inputs are given to the attractor from the MEC representation, $g$. Here we test the effect of these interactions by replacing the hierarchy matrix with both an all-to-all matrix, representing interactions between all modules, and an identity matrix, which represents no interactions of any kind. The results are shown in fig. 16.

It can be seen that replacing the hierarchy with all-to-all interactions has the effect of slowing the learning of grid cells, as place and grid cells are still visible yet the clarity is diminished. This is likely because the hierarchies enable course-level understanding of the structure to be passed down to the finer-leveled understanding, thus speeding up learning, as previously discussed. The zero-shot prediction accuracy is also severely affected, dropping by $\sim 50\%$.

Removing modular interactions entirely has a similar effect to replacement with all-to-all interactions. By replacing the interaction matrices with an identity matrix, frequency modules only interact with themselves, causing both grid and place cell representations to become less visible. The prediction accuracy still remains relatively high, at $\sim 75 - 80\%$, suggesting TEM's ability to generalise may not be integrally linked with it's ability to reproduce grid cells.

We also ran tests to determine whether removing the hierarchical interactions altogether had any impact upon the production of grid cells. To do this, we substituted the hierarchical matrix for the 'separate' identity matrix, the results are shown in fig. 17 above. It is clear that the removal of hierarchical interactions impedes grid cell learning, however structure is still present within the MEC representations. This supports the authors claim that hierarchies aid learning but are not necessary for generalisation.

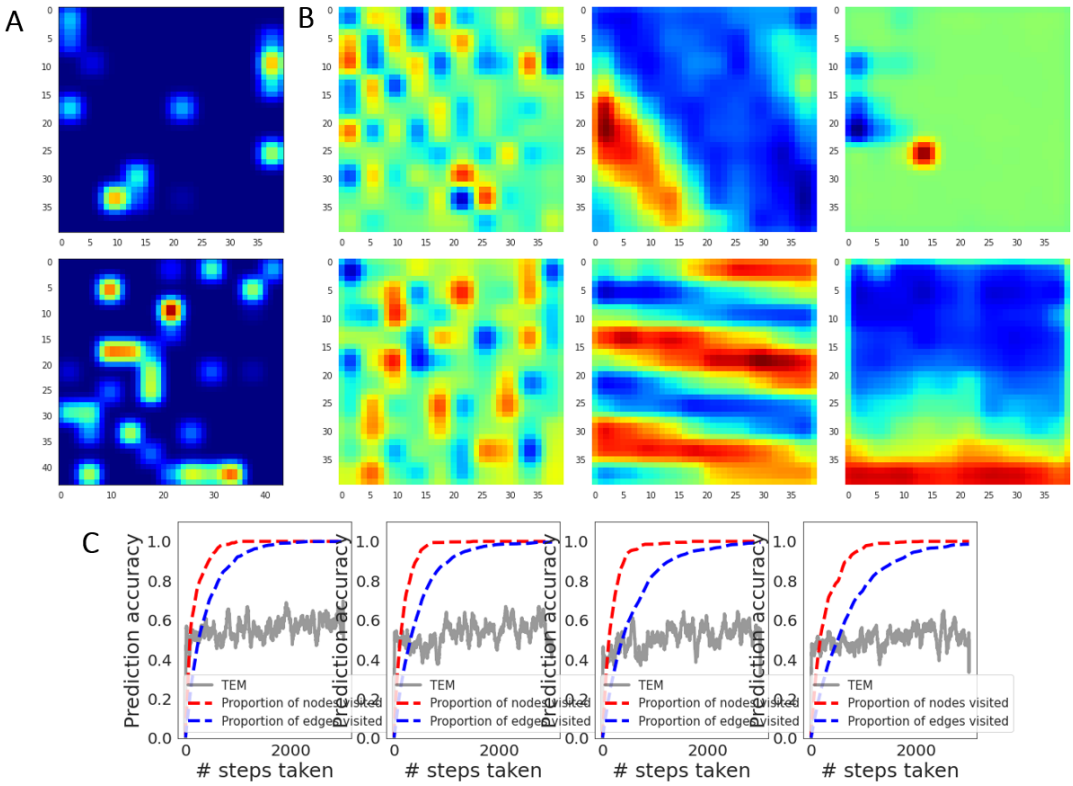


Figure 16: **All-to-All Hierarchy Test:** Effects of replacing spatial hierarchy with all-to-all interactions of frequency modules. **A:** place cells shown for two different environments. **B:** grid cells at high, medium and low frequencies (left to right), compared to those generated by the original code. **C:** Zero-shot prediction accuracy for 4 environments (grey) plotted against the proportion of nodes (red) and edges (blue) visited by the agent.

### 5.3 Mask Studies

In addition to the hierarchies discussed in the previous section, which are responsible for dictating how different frequency modules interact as they enter the attractor network, TEM also utilises hierarchical matrices via 'masks' on $p$ and $g$. These masks are applied to both MEC and HPC representations, and have a significant impact on TEM's ability to generalise, yet are not mentioned by the authors. Initially, we replace the hierarchical structure of both $p$ and $g$ masks, with separate and all-to-all matrices. The results of the former case, where frequency modules don't interact at all in either $p$ or $g$ masks are shown in the left panel of fig. 18, whilst the latter results are shown on the right.

The place mask has the same hierarchical structure, $\mathbf{H}_h$, as that used in the attractor network when taking input from the MEC, and is applied to the hippocampal memory $\mathbf{M}_t$ during the final Hebbian update at the end of each walk. The ablation of this hierarchical structure, by replacing it with an identity matrix, has the effect of destroying TEM's ability to reproduce place cells whilst reducing prediction accuracy, see fig. 19. The right panel shows the results when this hierarchy is replaced by all-to-all interactions, whereby every module interacts with every other module.

Whilst the grid mask also had a hierarchical structure, it was the transpose, $\mathbf{M}_t^T$ of that used in the p-mask and attractor network. This mask is applied during path integration of $g^g$, to the transition matrix learned by the MLP $W_a$. The effect of the ablation of this hierarchy can be seen in fig. 20. It can be seen that hippocampal place cells remain intact (those shown are of low frequency) but the grid cells are severely affected. This is especially noticeable at higher frequencies, where a strong banded structure appears along the diagonal. It is thus clear that the entorhinal mask has a profound affect on TEM's ability to reproduce grid cells.

These results seem to show decreases in prediction accuracy that aren't correlated with a similar reduction in the ability to generate grid cells. Similar to the results from hierarchy ablation, this therefore suggests that the ability of TEM to generalise might not be integrally linked to its capacity to reproduce neural phenomena. It should be noted, and is detailed below, that the accuracy of TEM has an associated uncertainty, and therefore more analyses should be done to clarify this point.

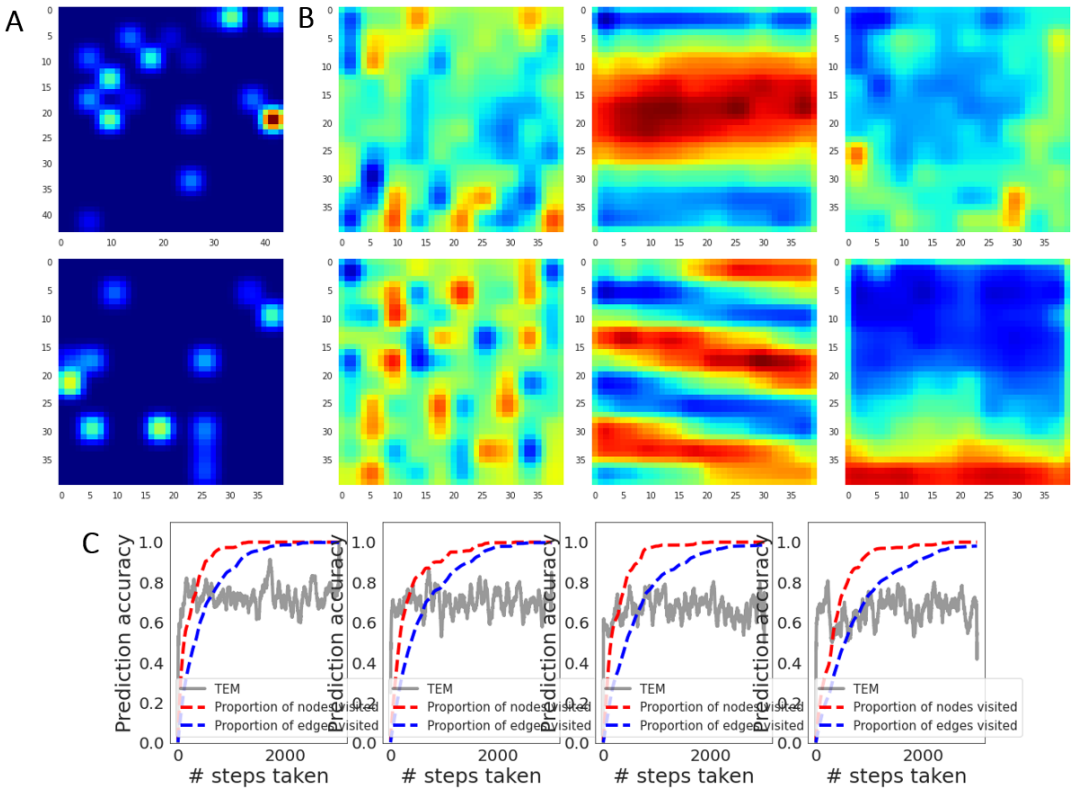


Figure 17: **Separate Hierarchy Test:** Effects of replacing spatial hierarchy with no interactions of frequency modules. **A:** place cells shown for two different environments. **B:** grid cells at high, medium and low frequencies (left to right), compared to those generated by the original code. **C:** Zero-shot prediction accuracy for 4 environments (grey) plotted against the proportion of nodes (red) and edges (blue) visited by the agent..

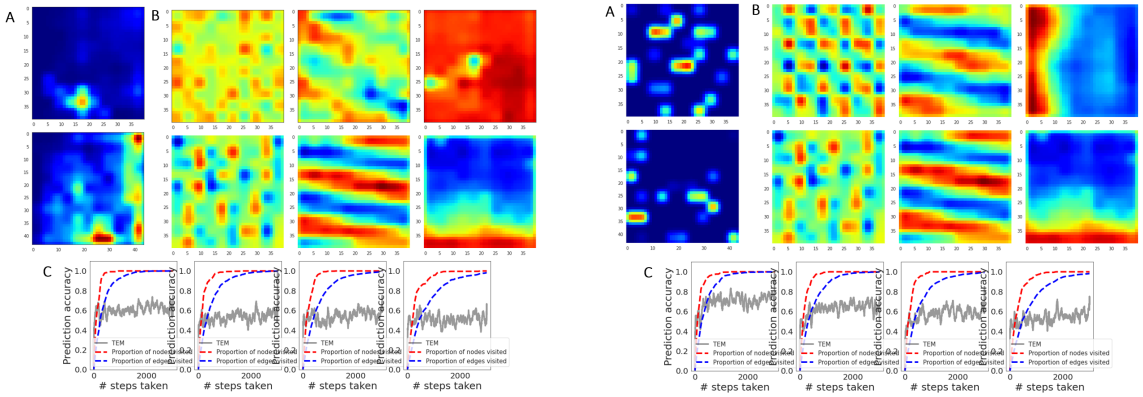


Figure 18: **Full Mask Replacement:** Effects of replacing both $p$ and $g$ mask structure with no interaction (left) and all-to-all interaction (right). **A:** place cells shown for two different environments. **B:** grid cells at high, medium and low frequencies (left to right), compared to those generated by the original code. **C:** Zero-shot prediction accuracy for 4 environments (grey) plotted against the proportion of nodes (red) and edges (blue) visited by the agent.

A similar test is done on both masks, however this time these are instead replaced by all-to-all interaction matrices. This has the effect of ensuring all frequency modules interact with one another whenever the $p$ and $g$ masks are applied, within the Hebbian update and path integration, respectively. The results for replacement of the mask on $p$ and $g$ can be seen in fig. 21 and fig. 22, respectively.

A similar, banded, periodic structure appears both in the hippocampal memories and the lowest frequency MEC representations. High and medium frequency grid cells remain relatively intact whilst the accuracy is only slightly reduced, decreasing by $\sim 25\%$. The result of similar replacement of the mask on $g$ yields less extreme changes, with place cells remaining intact and general structure of grid cell representations clearly visible. The highest frequency grid cells appear even clearer than in the baseline, however this may be due to inconsistencies in model training that have been discussed above.

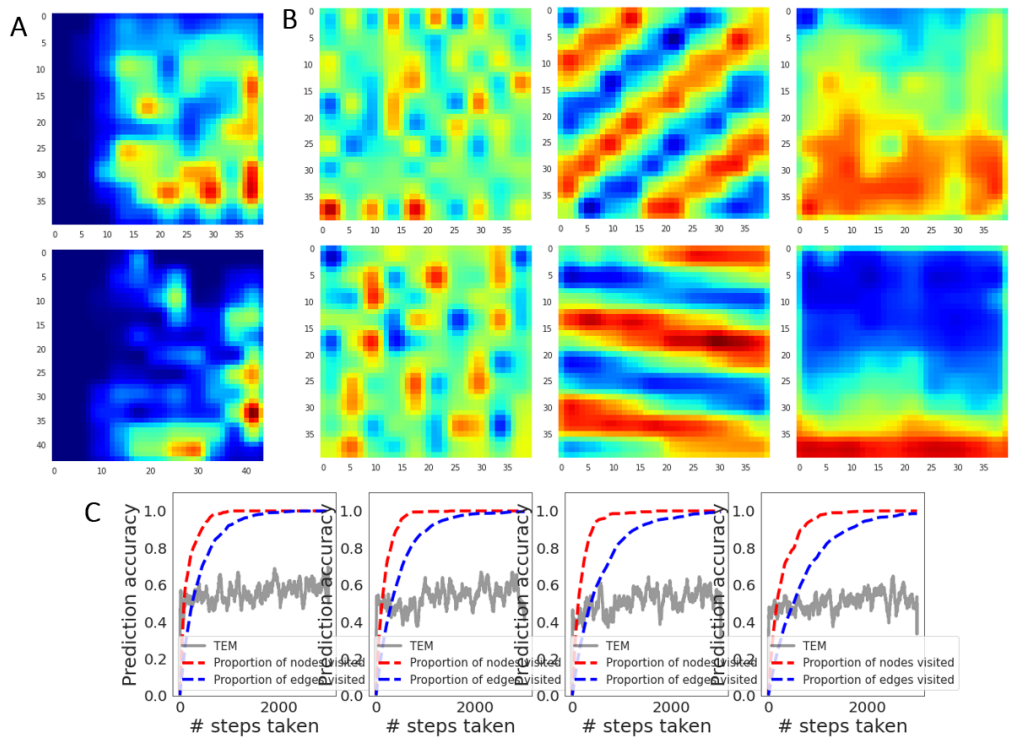


Figure 19: **Hippocampal Mask Test 1:** Effects of removing the hierarchical mask on $p$. **A:** place cells shown for two different environments. **B:** grid cells at high, medium and low frequencies (left to right), compared to those generated by the original code. **C:** Zero-shot prediction accuracy for 4 environments (grey) plotted against the proportion of nodes (red) and edges (blue) visited by the agent.

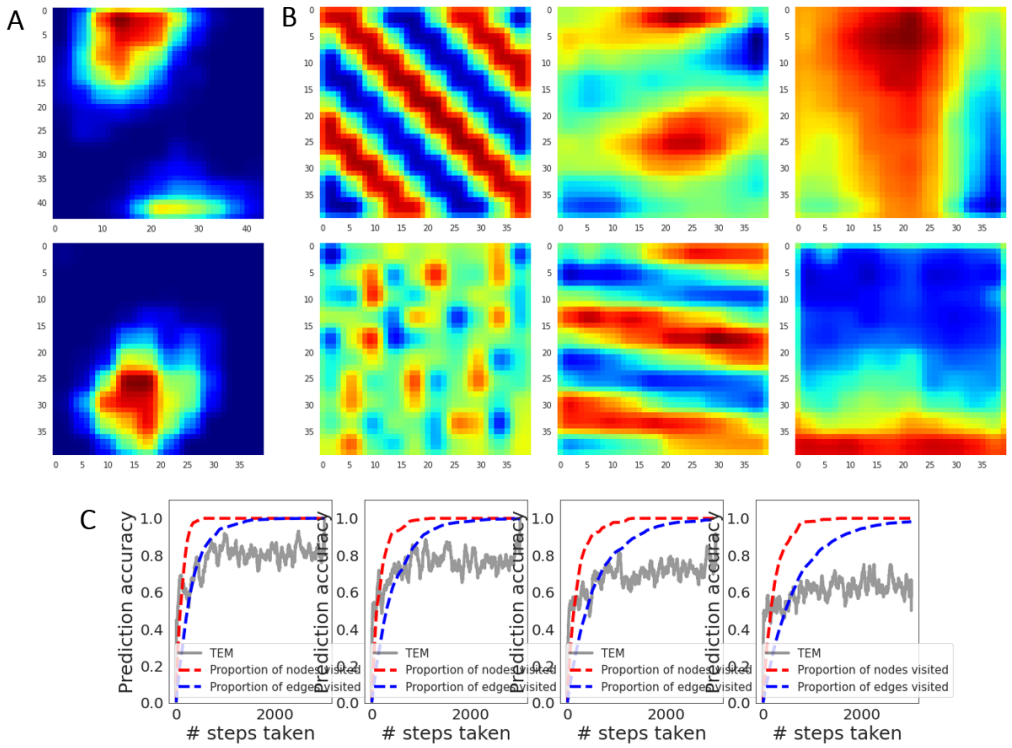


Figure 20: **Entorhinal Mask Test 1:** Effects of removing the hierarchical mask on $g$. **A:** place cells shown for two different environments. **B:** grid cells at high, medium and low frequencies (left to right), compared to those generated by the original code. **C:** Zero-shot prediction accuracy for 4 environments (grey) plotted against the proportion of nodes (red) and edges (blue) visited by the agent.

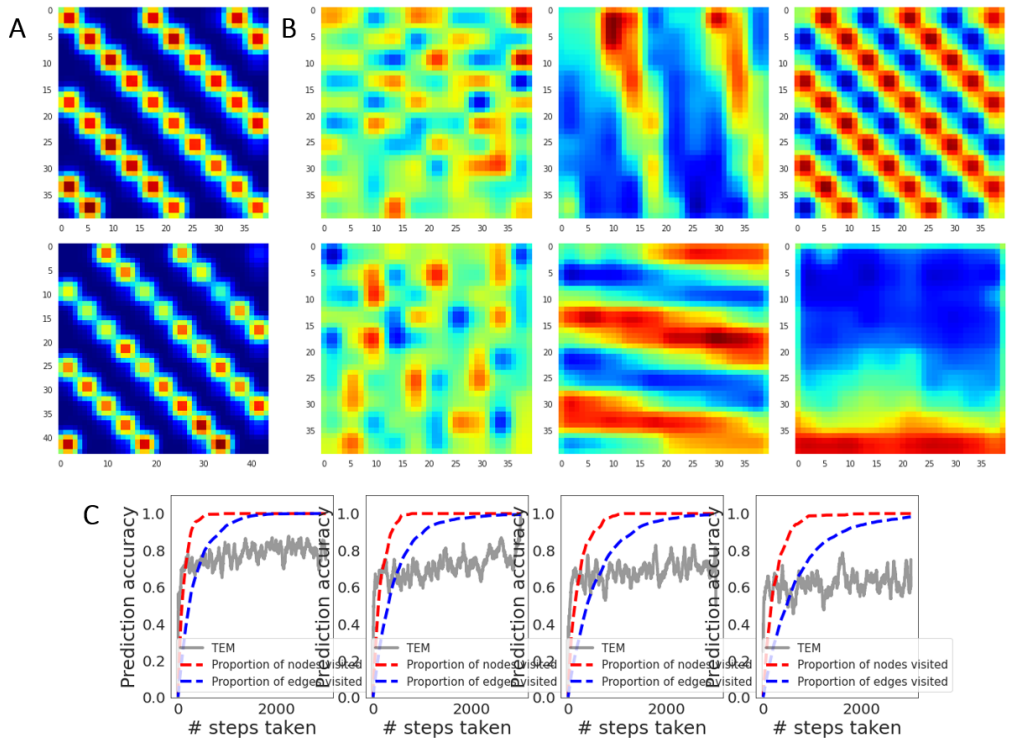


Figure 21: **Hippocampal Mask Test 2:** Effects of replacing the hierarchical mask on $p$ with an all-to-all interaction matrix. **A:** place cells shown for two different environments. **B:** grid cells at high, medium and low frequencies (left to right), compared to those generated by the original code. **C:** Zero-shot prediction accuracy for 4 environments (grey) plotted against the proportion of nodes (red) and edges (blue) visited by the agent.

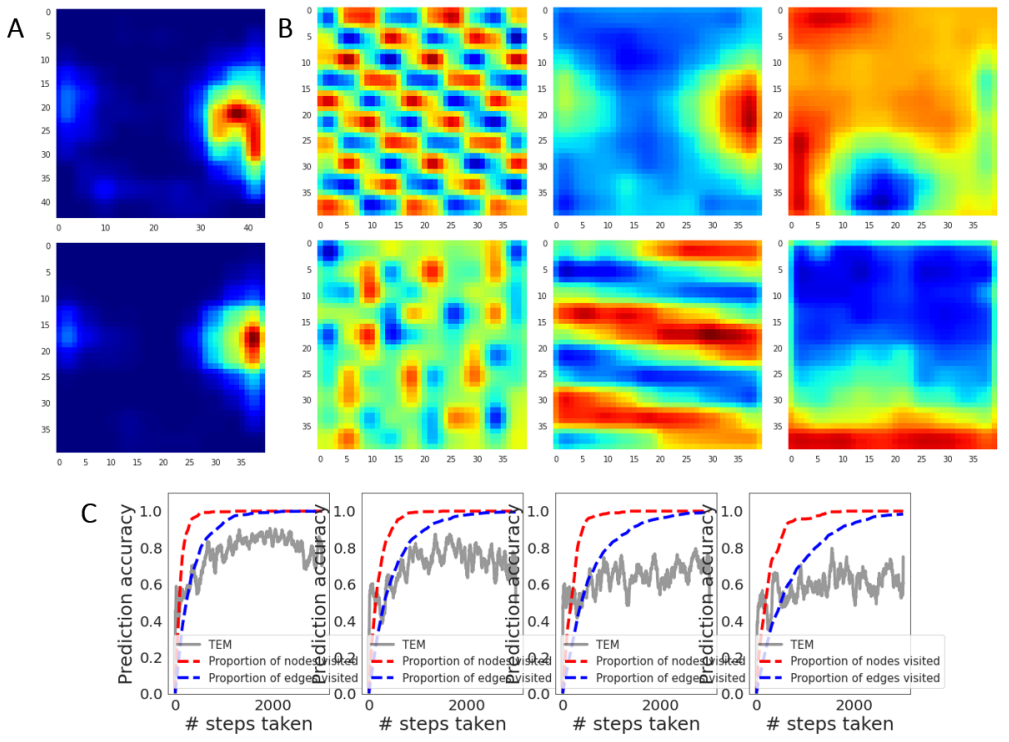


Figure 22: **Entorhinal Mask Test 2:** Effects of replacing the hierarchical mask on $g$ with an all-to-all interaction matrix. **A:** place cells shown for two different environments. **B:** grid cells at high, medium and low frequencies (left to right), compared to those generated by the original code. **C:** Zero-shot prediction accuracy for 4 environments (grey) plotted against the proportion of nodes (red) and edges (blue) visited by the agent.

## 6 Discussion

The Tollman-Eichenbaum machine is a hugely influential model in the field of neuroscience, and has been pivotal in aiding our understanding of spatial abstraction and representation in the hippocampus and surrounding areas. It provides a powerful framework of abstraction, which successfully learns both in spatial and non-spatial tasks, whilst also reproducing neural features that characterise HPC and EC function. It is for these reasons that TEM has been the focus of much attention, with attempts made to both explain and extent the model [21]. There is yet much scope for analysis and expansion and we hope that the description and implementation laid out here will aid those who are motivated to push the understanding of TEM forward.

The foundation of this work is rooted in a desire to improve the ease and reliability with which neuroscientists in disparate domains can cooperate. We aimed to implement the Tollman-Eichenbaum machine into our open-source framework, NeuralPlayground, so that experimentalists and computational neuroscientists alike can explore and engage with this compelling model of the hippocampus. As the implementation progressed, it became clear that there were aspects of TEM that should be clarified and expanded upon; we thus made a concerted effort to provide a consistent and robust description of the model theory, that paralleled it's implementation in our framework.

As a result of this detailed description of the model's function, we identified several key areas of interest, which we further pursued in testing. We produced results that highlight the importance of previously discussed components of TEM, such as it's hierarchical memory-retrieval, as well as features that weren't given due credit in the original paper, like hierarchical masks. We hope these results are informative to those who are eager to understand the results of this impressive model and form the first of many similar future analyses.

The result of this implementation is an instantiation of the Tollman-Eichenbaum machine that reproduces the important results whilst also offering a test bed to expand and explore the model further. We introduce TEM to a new environment, allowing it to be tested in novel and challenging arenas and implement it in such a way that the action-policy can be altered as the model trains, allowing for exciting future extensions. We hope this work will be of use to those interested in the Tollman-Eichenbaum machine, and related models, and will constitute a key feature of our new collaborative framework.

## References


[1] A. Martín-Araguz, C. Bustamante-Martínez, M. Emam-Mansour, and J. Moreno-Martínez, "Neuroscience in ancient egypt and in the school of alexandria," *Revista de neurologia*, vol. 34, pp. 1183–94, 06 2002.

[2] S.-M. Pulst, "Neurogenetics: single gene disorders," *Journal of Neurology, Neurosurgery &amp Psychiatry*, vol. 74, pp. 1608–1614, Dec. 2003.

[3] N. Schaefer, C. Rotermund, E.-M. Blumrich, M. V. Lourenco, P. Joshi, R. U. Hegemann, S. Jamwal, N. Ali, E. M. G. Romero, S. Sharma, S. Ghosh, J. K. Sinha, H. Loke, V. Jain, K. Lepeta, A. Salamian, M. Sharma, M. Golpich, K. Nawrotek, R. K. Paidi, S. M. Shahidzadeh, T. Piermartiri, E. Amini, V. Pastor, Y. Wilson, P. A. Adeniyi, A. K. Datusalia, B. Vafadari, V. Saini, E. Suárez-Pozos, N. Kushwah, P. Fontanet, and A. J. Turner, "The malleable brain: plasticity of neural circuits and behavior - a review from students to students," *Journal of Neurochemistry*, vol. 142, pp. 790–811, Aug. 2017.

[4] J. C. Whittington, T. H. Muller, S. Mark, G. Chen, C. Barry, N. Burgess, and T. E. Behrens, "The tolman-eichenbaum machine: Unifying space and relational memory through generalization in the hippocampal formation," *Cell*, vol. 183, pp. 1249–1263.e23, Nov. 2020.

[5] N. Burgess, C. Barry, and J. O'Keefe, "An oscillatory interference model of grid cell firing," *Hippocampus*, vol. 17, no. 9, pp. 801–812, 2007.

[6] H. T. Blair, K. Gupta, and K. Zhang, "Conversion of a phase- to a rate-coded position signal by a three-stage model of theta cells, grid cells, and place cells," *Hippocampus*, vol. 18, pp. 1239–1255, Dec. 2008.

[7] N. Burgess, "Grid cells and theta as oscillatory interference: Theory and predictions," *Hippocampus*, vol. 18, pp. 1157–1174, Dec. 2008.

[8] C. Barry, C. Lever, R. Hayman, T. Hartley, S. Burton, J. O'Keefe, K. Jeffery, and . Burgess, "The boundary vector cell model of place cell firing and spatial memory," *Reviews in the Neurosciences*, vol. 17, Jan. 2006.

[9] A. Arleo and W. Gerstner, "Spatial cognition and neuro-mimetic navigation: a model of hippocampal place cell activity," *Biological Cybernetics*, vol. 83, pp. 287–299, Aug. 2000.

[10] T. Hafting, M. Fyhn, S. Molden, M.-B. Moser, and E. I. Moser, "Microstructure of a spatial map in the entorhinal cortex," *Nature*, vol. 436, pp. 801–806, June 2005.

[11] J. Krupic, N. Burgess, and J. O'Keefe, "Neural representations of location composed of spatially periodic bands," *Science*, vol. 337, pp. 853–857, Aug. 2012.

[12] J.-R. Duhamel, C. L. Colby, and M. E. Goldberg, "The updating of the representation of visual space in parietal cortex by intended eye movements," *Science*, vol. 255, pp. 90–92, Jan. 1992.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[14] M. Botvinick, J. X. Wang, W. Dabney, K. J. Miller, and Z. Kurth-Nelson, "Deep reinforcement learning and its neuroscientific implications," *Neuron*, vol. 107, pp. 603–616, Aug. 2020.

[15] D. Lee, H. Seo, and M. W. Jung, "Neural basis of reinforcement learning and decision making," *Annual Review of Neuroscience*, vol. 35, pp. 287–308, July 2012.

[16] K. L. Stachenfeld, M. M. Botvinick, and S. J. Gershman, "The hippocampus as a predictive map," Dec. 2016.

[17] Ø. A. Høydal, E. R. Skytøen, S. O. Andersson, M.-B. Moser, and E. I. Moser, "Object-vector coding in the medial entorhinal cortex," *Nature*, vol. 568, pp. 400–404, Apr. 2019.

[18] J. A. Dusek and H. Eichenbaum, "The hippocampus and memory for orderly stimulus relations," *Proceedings of the National Academy of Sciences*, vol. 94, pp. 7109–7114, June 1997.

[19] M. Rmus, H. Ritz, L. E. Hunter, A. M. Bornstein, and A. Shenhav, "Humans can navigate complex graph structures acquired during latent learning," *Cognition*, vol. 225, p. 105103, Aug. 2022.

[20] P. Piray and N. D. Daw, "Linear reinforcement learning in planning, grid fields, and cognitive control," *Nature Communications*, vol. 12, Aug. 2021.

[21] J. C. R. Whittington, J. Warren, and T. E. J. Behrens, "Relating transformers to models and neural representations of the hippocampal formation," 2021.

[22] S. Ciranka, J. Linde-Domingo, I. Padezhki, C. Wicharz, C. M. Wu, and B. Spitzer, "Asymmetric reinforcement learning facilitates human inference of transitive relations," *Nature Human Behaviour*, vol. 6, pp. 555–564, Jan. 2022.

[23] A. B. Baram, T. H. Muller, H. Nili, M. M. Garvert, and T. E. J. Behrens, "Entorhinal and ventromedial prefrontal cortices abstract and generalize the structure of reinforcement learning problems," *Neuron*, vol. 109, pp. 713–723.e7, Feb. 2021.

[24] J. P. Geerts, F. Chersi, K. L. Stachenfeld, and N. Burgess, "A general model of hippocampal and dorsal striatal learning and decision making," *Proceedings of the National Academy of Sciences*, vol. 117, pp. 31427–31437, Nov. 2020.

[25] E. C. Tolman, "Cognitive maps in rats and men.," *Psychological Review*, vol. 55, no. 4, pp. 189–208, 1948.

[26] H. Eichenbaum and N. J. Cohen, "Can we reconcile the declarative memory and spatial navigation views on hippocampal function?," *Neuron*, vol. 83, pp. 764–770, Aug. 2014.

[27] S. S. Deshmukh and J. J. Knierim, "Representation of non-spatial and spatial information in the lateral entorhinal cortex," *Frontiers in Behavioral Neuroscience*, vol. 5, 2011.

[28] F. Savelli, D. Yoganarasimha, and J. J. Knierim, "Influence of boundary removal on the spatial representations of the medial entorhinal cortex," *Hippocampus*, vol. 18, pp. 1270–1282, Dec. 2008.

[29] J. R. Manns and H. Eichenbaum, "Evolution of declarative memory," *Hippocampus*, vol. 16, no. 9, pp. 795–808, 2006.

[30] D. Hebb, *The Organization of Behavior*. Psychology Press, Apr. 2005.

[31] M. L. Shapiro, H. Tanila, and H. Eichenbaum, "Cues that hippocampal place cells encode: Dynamic and hierarchical representation of local and distal stimuli," *Hippocampus*, vol. 7, no. 6, pp. 624–642, 1997.

[32] H. Stensola, T. Stensola, T. Solstad, K. Frøland, M.-B. Moser, and E. I. Moser, "The entorhinal grid map is discretized," *Nature*, vol. 492, pp. 72–78, Dec. 2012.

[33] K. B. Kjelstrup, T. Solstad, V. H. Brun, T. Hafting, S. Leutgeb, M. P. Witter, E. I. Moser, and M.-B. Moser, "Finite scale of spatial representation in the hippocampus," *Science*, vol. 321, pp. 140–143, July 2008.

[34] C. Sun, W. Yang, J. Martin, and S. Tonegawa, "Hippocampal neurons represent events as transferable units of experience," *Nature Neuroscience*, vol. 23, pp. 651–663, Apr. 2020.

[35] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities.," *Proceedings of the National Academy of Sciences*, vol. 79, pp. 2554–2558, Apr. 1982.

[36] J. Ba, G. Hinton, V. Mnih, J. Z. Leibo, and C. Ionescu, "Using fast weights to attend to the recent past," 2016.

[37] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013.

[38] T. Wernle, T. Waaga, M. Mørreaunet, A. Treves, M.-B. Moser, and E. I. Moser, "Integration of grid maps in merged environments," *Nature Neuroscience*, vol. 21, pp. 92–101, Dec. 2017.

[39] A. Tsao, J. Sugar, L. Lu, C. Wang, J. J. Knierim, M.-B. Moser, and E. I. Moser, "Integrating time from experience in the lateral entorhinal cortex," *Nature*, vol. 561, pp. 57–62, Aug. 2018.