

# Explicit Memory, MERLIN, FTW

Milan Straka

 January 5, 2021



EUROPEAN UNION  
European Structural and Investment Fund  
Operational Programme Research,  
Development and Education

Charles University in Prague  
Faculty of Mathematics and Physics  
Institute of Formal and Applied Linguistics



unless otherwise stated

Recall that a **Markov decision process** (MDP) is a quadruple  $(\mathcal{S}, \mathcal{A}, p, \gamma)$ , where:

- $\mathcal{S}$  is a set of states,
- $\mathcal{A}$  is a set of actions,
- $p(\mathcal{S}_{t+1} = s', R_{t+1} = r | \mathcal{S}_t = s, A_t = a)$  is a probability that action  $a \in \mathcal{A}$  will lead from state  $s \in \mathcal{S}$  to  $s' \in \mathcal{S}$ , producing a **reward**  $r \in \mathbb{R}$ ,
- $\gamma \in [0, 1]$  is a **discount factor**.

**Partially observable Markov decision process** extends the Markov decision process to a sextuple  $(\mathcal{S}, \mathcal{A}, p, \gamma, \mathcal{O}, o)$ , where in addition to an MDP

- $\mathcal{O}$  is a set of observations,
- $o(\mathcal{O}_t | \mathcal{S}_t, A_{t-1})$  is an observation model, which is used as agent input instead of  $\mathcal{S}_t$ .

# Partially Observable MDPs

In Deep RL, partially observable MDPs are usually handled using recurrent networks. After suitable encoding of input observation  $O_t$  and previous action  $A_{t-1}$ , a RNN (usually LSTM) unit is used to model the current  $S_t$  (or its suitable latent representation), which is in turn utilized to produce  $A_t$ .

## a. RL-LSTM

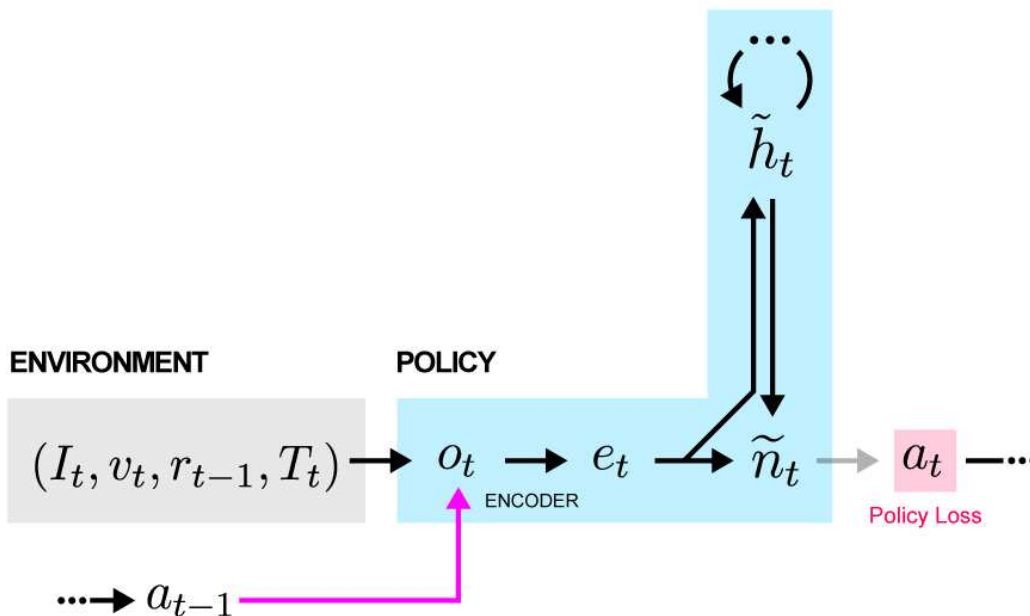


Figure 1a of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

However, keeping all information in the RNN state is substantially limiting. Therefore, *memory-augmented* networks can be used to store suitable information in external memory (in the lines of NTM, DNC or MANN models).

We now describe an approach used by Merlin architecture (*Unsupervised Predictive Memory in a Goal-Directed Agent* DeepMind Mar 2018 paper).

## b. RL-MEM

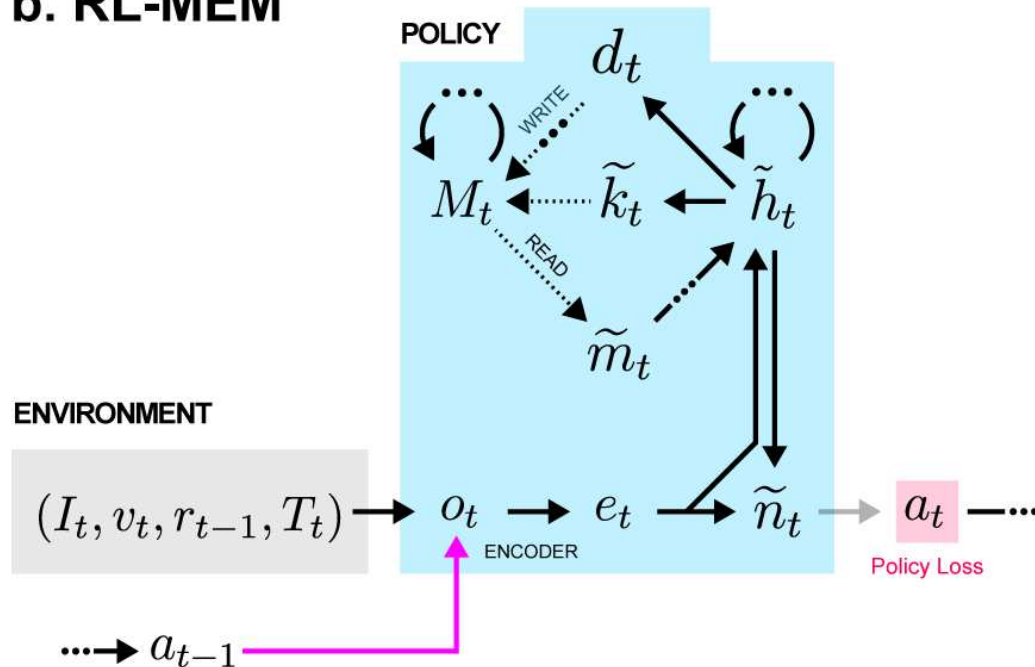


Figure 1b of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

# MERLIN – Memory Module

Let  $\mathbf{M}$  be a memory matrix of size  $N_{mem} \times 2|\mathbf{e}|$ .

Assume we have already encoded observations as  $\mathbf{e}_t$  and previous action  $a_{t-1}$ . We concatenate them with  $K$  previously read vectors and process them by a deep LSTM (two layers are used in the paper) to compute  $\mathbf{h}_t$ .

Then, we apply a linear layer to  $\mathbf{h}_t$ , computing  $K$  key vectors  $\mathbf{k}_1, \dots, \mathbf{k}_K$  of length  $2|\mathbf{e}|$  and  $K$  positive scalars  $\beta_1, \dots, \beta_K$ .

**Reading:** For each  $i$ , we compute cosine similarity of  $\mathbf{k}_i$  and all memory rows  $M_j$ , multiply the similarities by  $\beta_i$  and pass them through a softmax to obtain weights  $\omega_i$ . The read vector is then computed as  $\mathbf{M}\omega_i$ .

**Writing:** We find one-hot write index  $\mathbf{v}_{wr}$  to be the least used memory row (we keep usage indicators and add read weights to them). We then compute  $\mathbf{v}_{ret} \leftarrow \gamma \mathbf{v}_{ret} + (1 - \gamma) \mathbf{v}_{wr}$ , and update the memory matrix using  $\mathbf{M} \leftarrow \mathbf{M} + \mathbf{v}_{wr}[\mathbf{e}_t, \mathbf{0}] + \mathbf{v}_{ret}[\mathbf{0}, \mathbf{e}_t]$ .

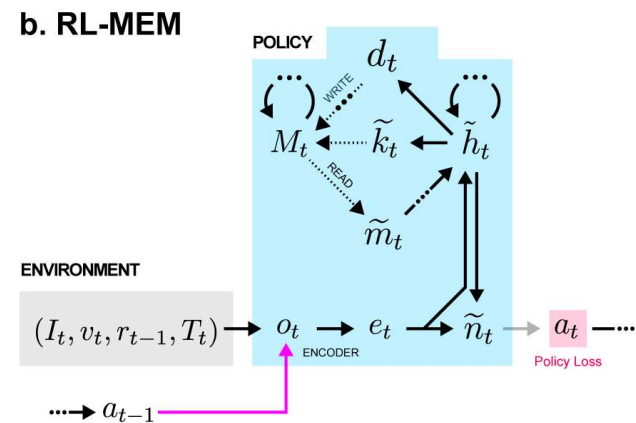


Figure 1b of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

However, updating the encoder and memory content purely using RL is inefficient. Therefore, MERLIN includes a *memory-based predictor (MBP)* in addition to policy. The goal of MBP is to compress observations into low-dimensional state representations  $\mathbf{z}$  and storing them in memory.

According to the paper, the idea of unsupervised and predictive modeling has been entertained for decades, and recent discussions have proposed such modeling to be connected to hippocampal memory.

We want the state variables not only to faithfully represent the data, but also emphasise rewarding elements of the environment above irrelevant ones. To accomplish this, the authors follow the hippocampal representation theory of Gluck and Myers, who proposed that hippocampal representations pass through a compressive bottleneck and then reconstruct input stimuli together with task reward.

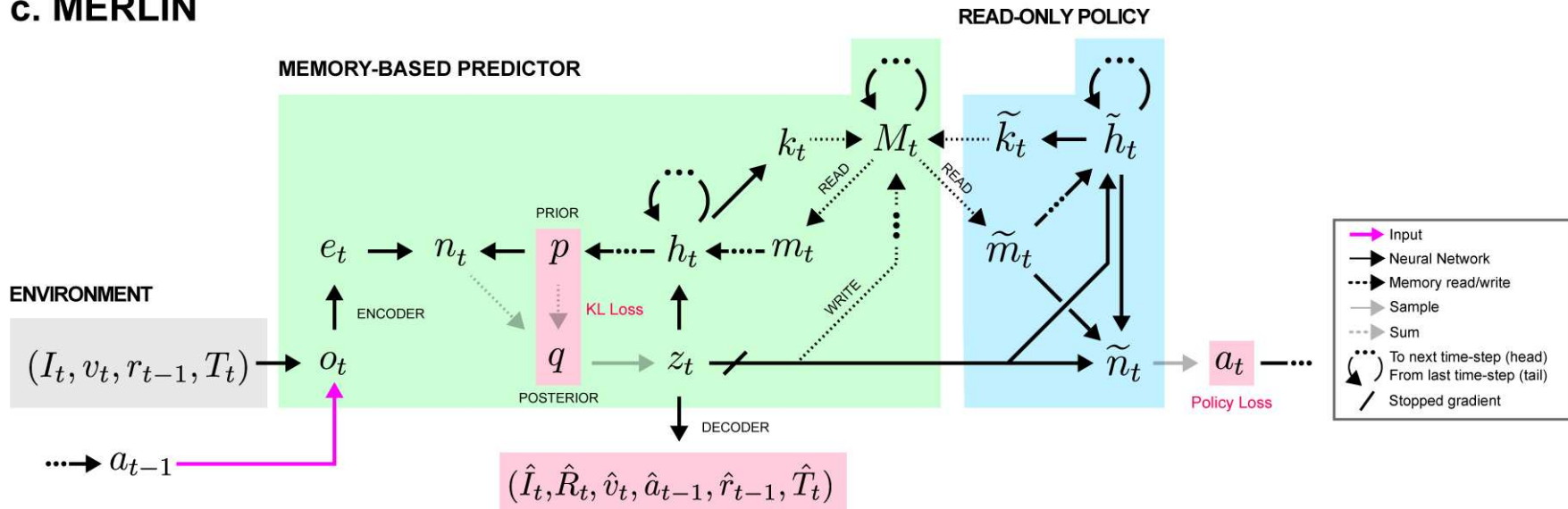
In MERLIN, a *prior* distribution over  $\mathbf{z}_t$  predicts next state variable conditioned on history of state variables and actions  $p(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{a}_{t-1}, \dots, \mathbf{z}_1, \mathbf{a}_1)$ , and *posterior* corrects the prior using the new observation  $\mathbf{o}_t$ , forming a better estimate  $q(\mathbf{z}_t | \mathbf{o}_t, \mathbf{z}_{t-1}, \mathbf{a}_{t-1}, \dots, \mathbf{z}_1, \mathbf{a}_1)$ .

# MERLIN — Prior and Posterior

To achieve the mentioned goals, we add two terms to the loss.

- We try reconstructing input stimuli, action, reward and return using a sample from the state variable posterior, and add the difference of the reconstruction and ground truth to the loss.
- We also add KL divergence of the prior and posterior to the loss, to ensure consistency between the prior and posterior.

## c. MERLIN



Reconstruction Loss  
 Figure 1c of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

---

**Algorithm 1** MERLIN Worker Pseudocode
 

---

```

// Assume global shared parameter vectors  $\theta$  for the policy network and  $\chi$  for the memory-
// based predictor; global shared counter  $T := 0$ 
// Assume thread-specific parameter vectors  $\theta', \chi'$ 
// Assume discount factor  $\gamma \in (0, 1]$  and bootstrapping parameter  $\lambda \in [0, 1]$ 
Initialize thread step counter  $t := 1$ 
repeat
  Synchronize thread-specific parameters  $\theta' := \theta; \chi' := \chi$ 
  Zero model's memory & recurrent state if new episode begins
   $t_{\text{start}} := t$ 
  repeat
    Prior  $\mathcal{N}(\mu_t^p, \log \Sigma_t^p) = p(h_{t-1}, m_{t-1})$ 
     $e_t = \text{enc}(o_t)$ 
    Posterior  $\mathcal{N}(\mu_t^q, \log \Sigma_t^q) = q(e_t, h_{t-1}, m_{t-1}, \mu_t^p, \log \Sigma_t^p)$ 
    Sample  $z_t \sim \mathcal{N}(\mu_t^q, \log \Sigma_t^q)$ 
    Policy network update  $\tilde{h}_t = \text{rec}(\tilde{h}_{t-1}, \tilde{m}_t, \text{StopGradient}(z_t))$ 
    Policy distribution  $\pi_t = \pi(\tilde{h}_t, \text{StopGradient}(z_t))$ 
    Sample  $a_t \sim \pi_t$ 
     $h_t = \text{rec}(h_{t-1}, m_t, z_t)$ 
    Update memory with  $z_t$  by Methods Eq. 2
     $R_t, o_t^r = \text{dec}(z_t, \pi_t, a_t)$ 
    Apply  $a_t$  to environment and receive reward  $r_t$  and observation  $o_{t+1}$ 
     $t := t + 1; T := T + 1$ 
  until environment termination or  $t - t_{\text{start}} == \tau_{\text{window}}$ 
  If not terminated, run additional step to compute  $V_t^\pi(z_{t+1}, \log \pi_{t+1})$ 
  and set  $R_{t+1} := V^\pi(z_{t+1}, \log \pi_{t+1})$  // (but don't increment counters)
  Reset performance accumulators  $\mathcal{A} := 0; \mathcal{L} := 0; \mathcal{H} := 0$ 
  for  $k$  from  $t$  down to  $t_{\text{start}}$  do
     $\gamma_t := \begin{cases} 0, & \text{if } k \text{ is environment termination} \\ \gamma, & \text{otherwise} \end{cases}$ 
     $R_k := r_k + \gamma_t R_{k+1}$ 
     $\delta_k := r_k + \gamma_t V^\pi(z_{k+1}, \log \pi_{k+1}) - V^\pi(z_k, \log \pi_k)$ 
     $A_k := \delta_k + (\gamma \lambda) A_{k+1}$ 
     $\mathcal{L} := \mathcal{L} + \mathcal{L}_k$  (Eq. 7)
     $\mathcal{A} := \mathcal{A} + A_k \log \pi_k[a_k]$ 
     $\mathcal{H} := \mathcal{H} - \alpha_{\text{entropy}} \sum_i \pi_k[i] \log \pi_k[i]$  (Entropy loss)
  end for
   $d\chi' := \nabla_{\chi'} \mathcal{L}$ 
   $d\theta' := \nabla_{\theta'} (\mathcal{A} + \mathcal{H})$ 
  Asynchronously update via gradient ascent  $\theta$  using  $d\theta'$  and  $\chi$  using  $d\chi'$ 
until  $T > T_{\text{max}}$ 

```

Algorithm 1 of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.



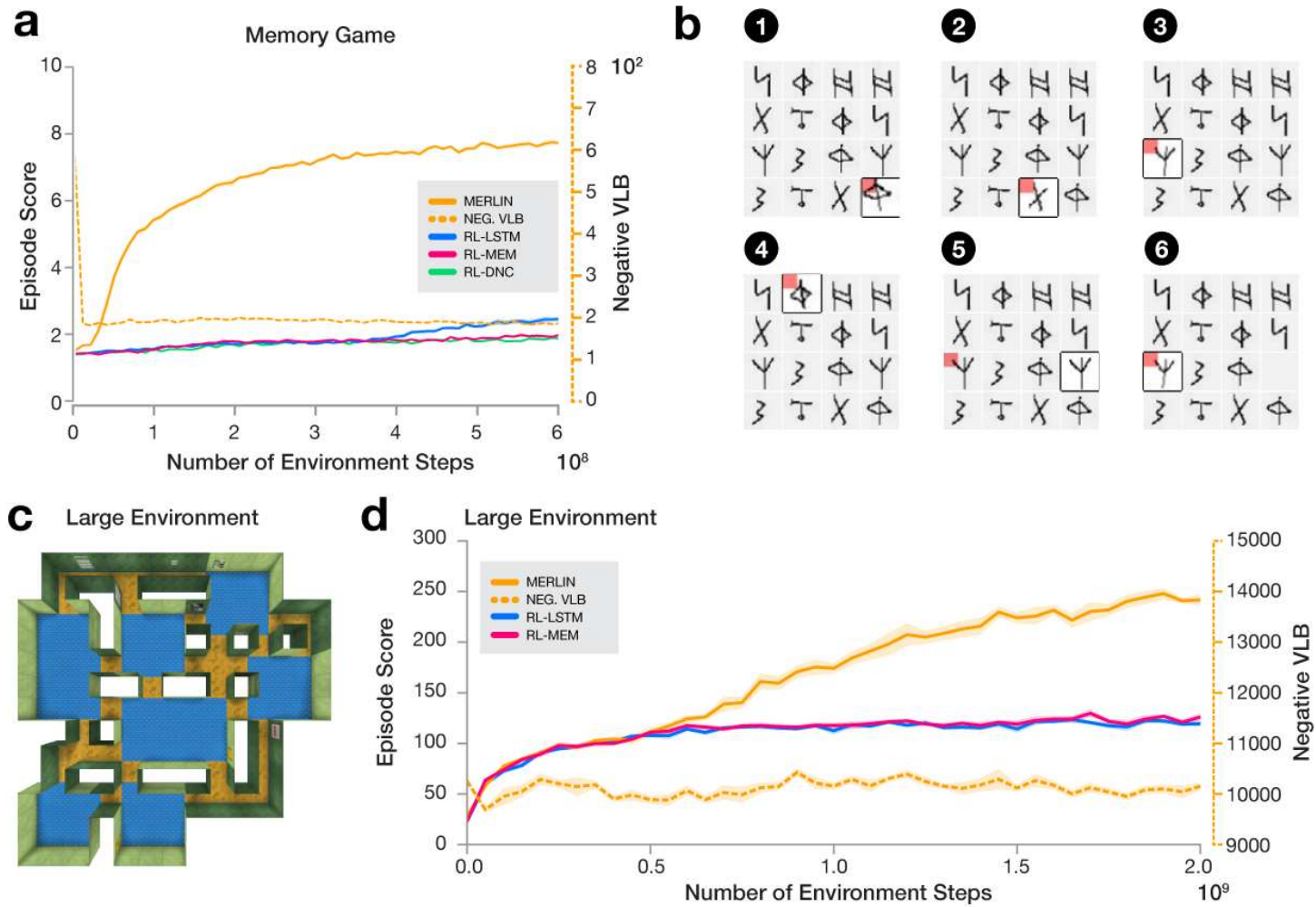


Figure 2 of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

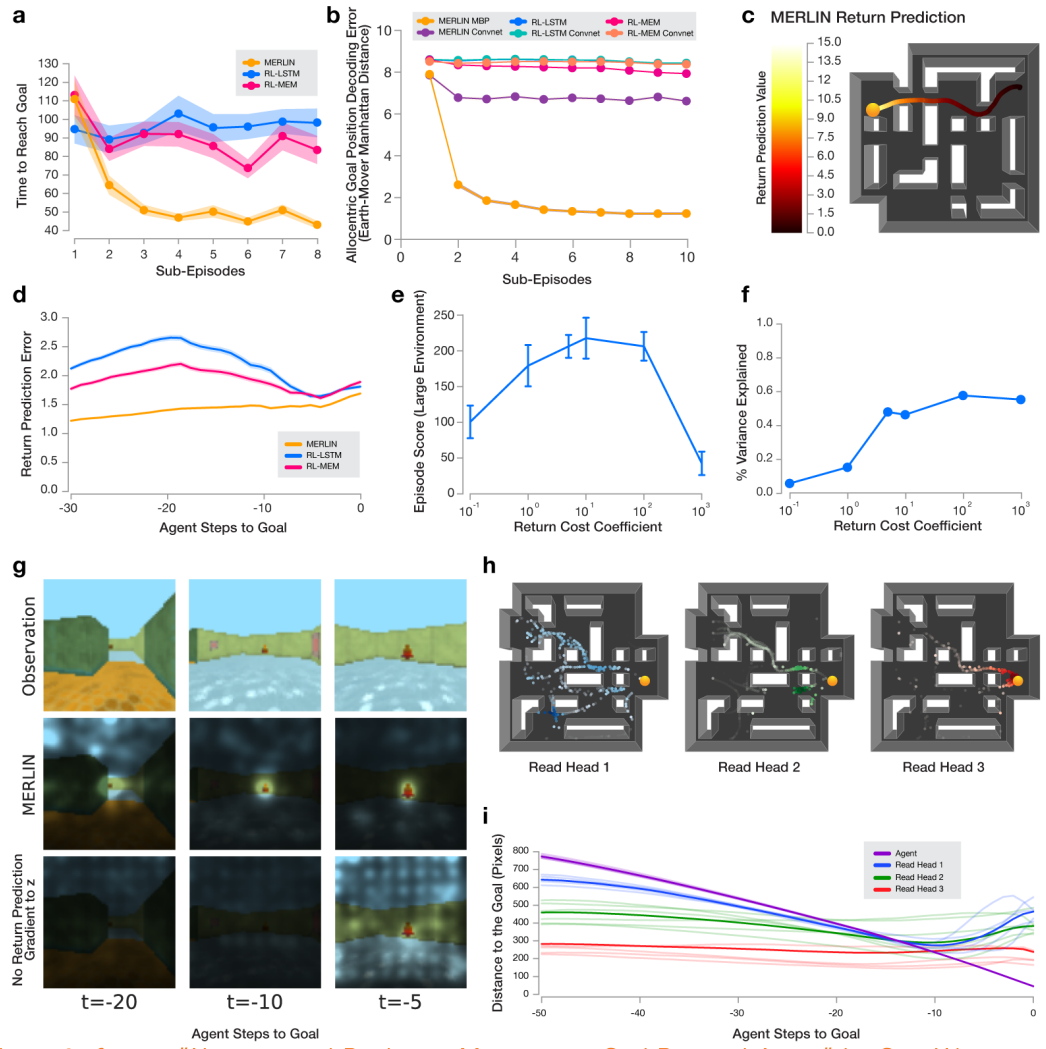
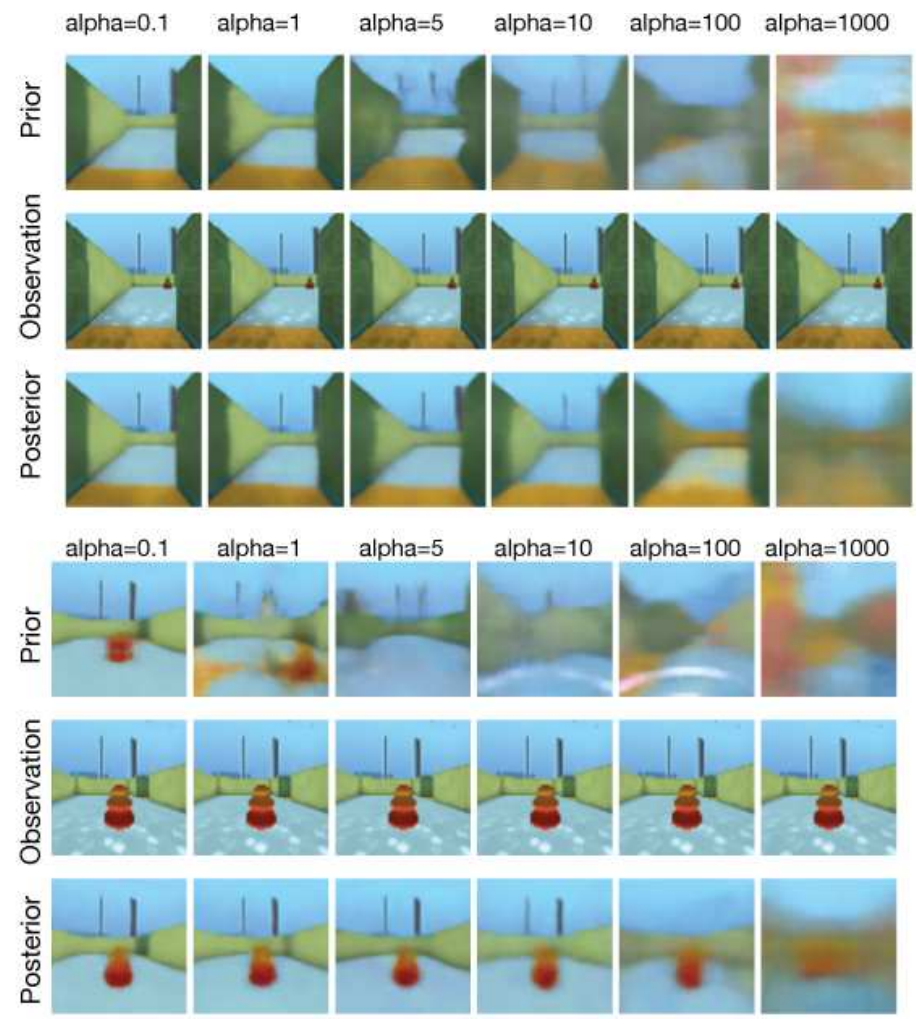
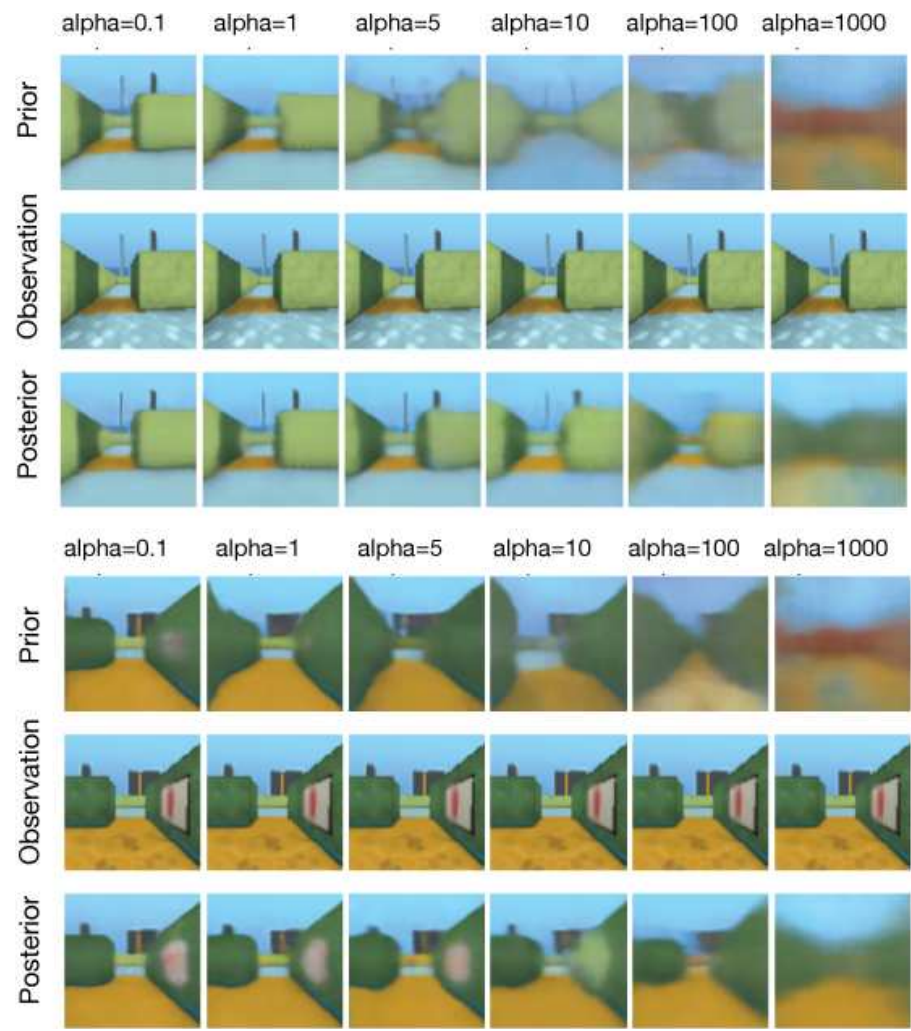


Figure 3 of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.



Extended Figure 3 of paper "Unsupervised Predictive Memory in a Goal-Directed Agent" by Greg Wayne et al.

# For the Win agent for Capture The Flag

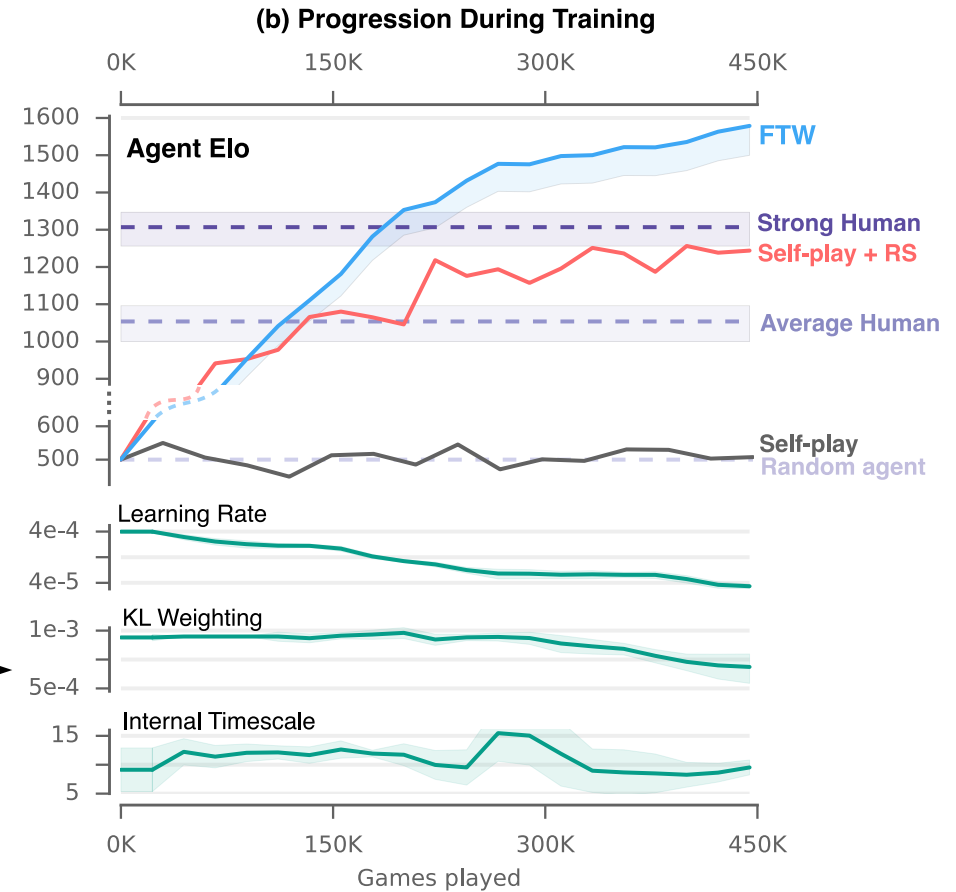
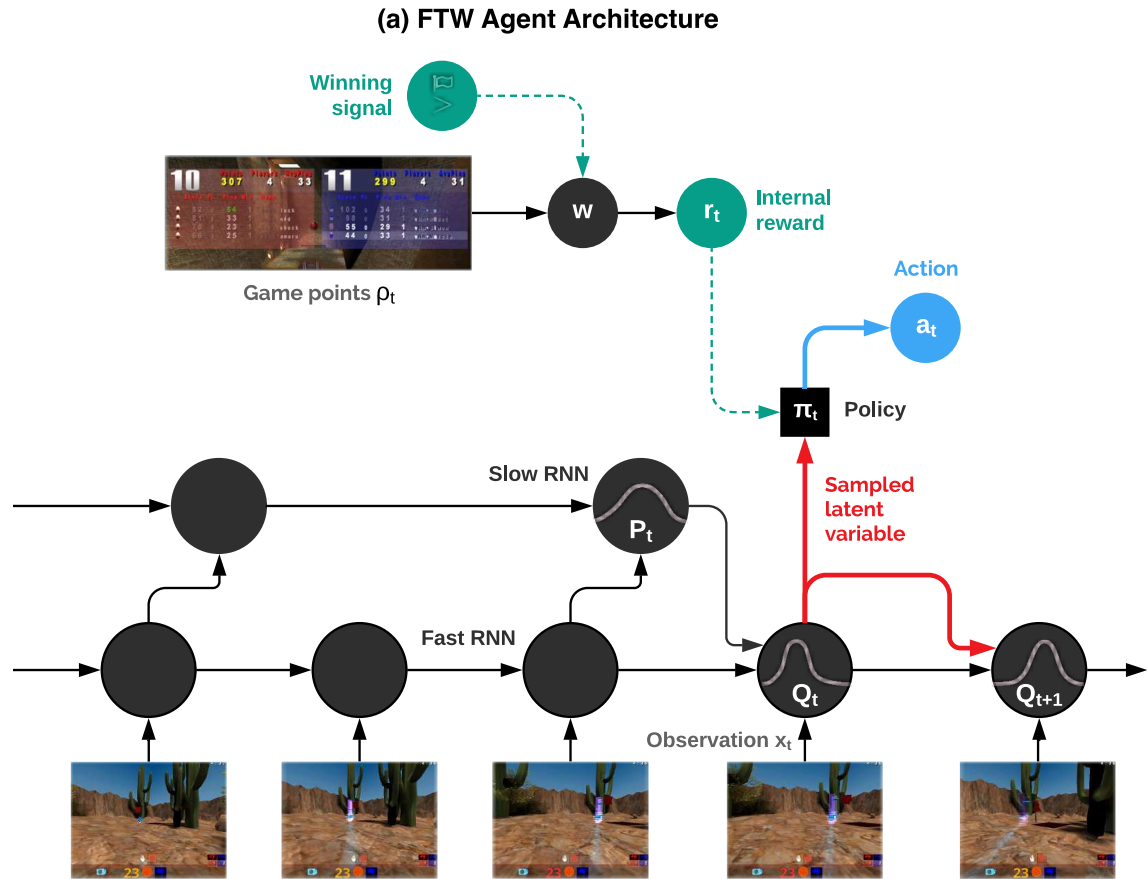


Figure 2 of paper "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning" by Max Jaderber et al.

- Extension of the MERLIN architecture.
- Hierarchical RNN with two timescales.
- Population based training controlling KL divergence penalty weights, slow ticking RNN speed and gradient flow factor from fast to slow RNN.

# For the Win agent for Capture The Flag

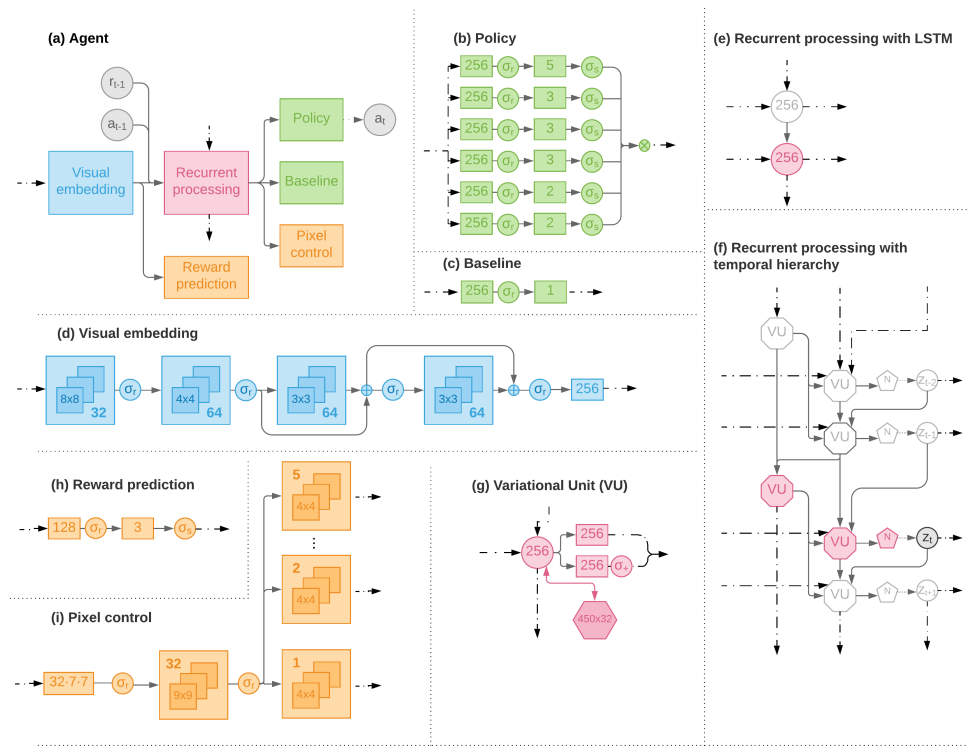


Figure S10 of paper "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning" by Max Jaderber et al.

# For the Win agent for Capture The Flag

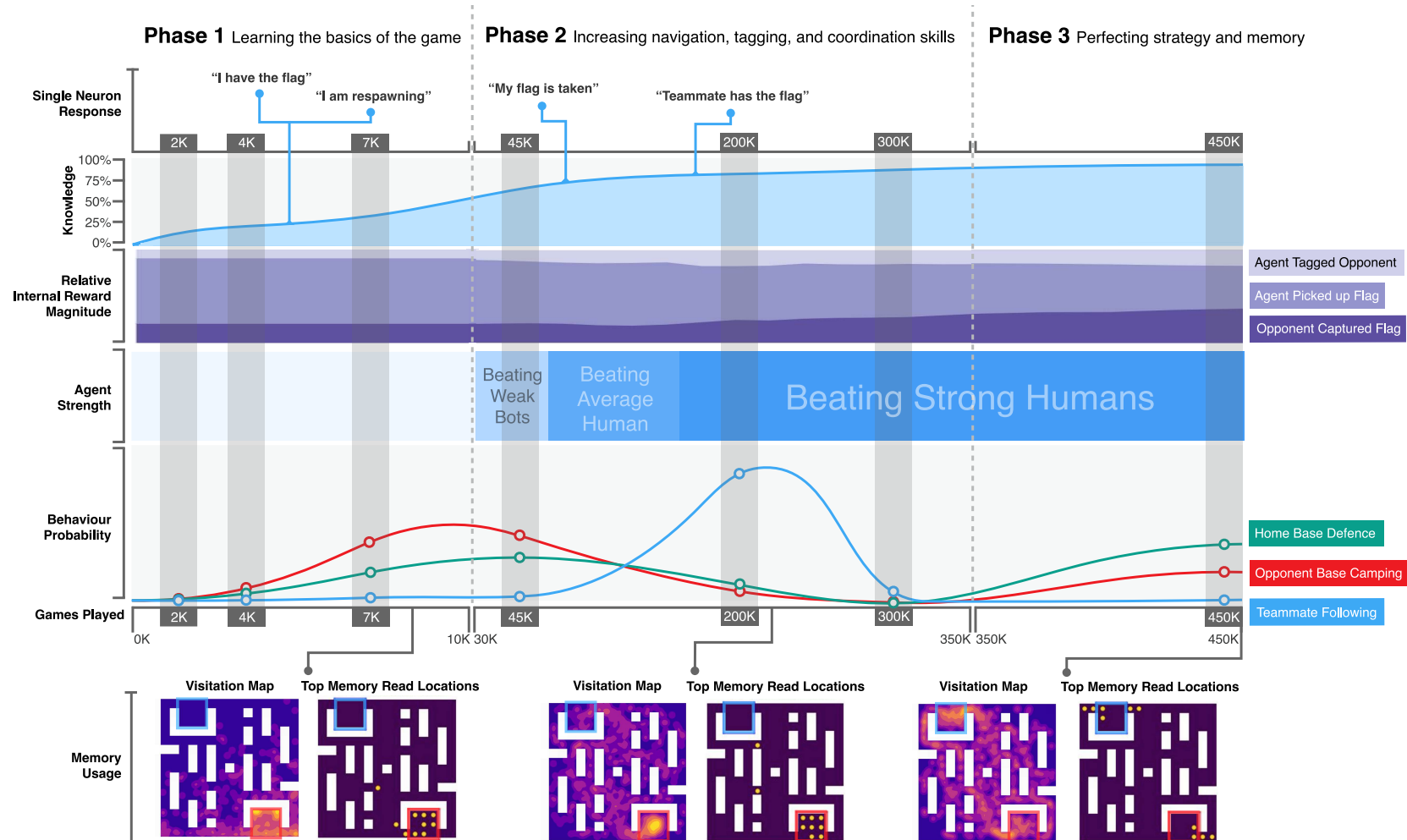


Figure 4 of paper "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning" by Max Jaderber et al.