

# Deep Active Inference for Pixel-Based Discrete Control: Evaluation on the Car Racing Problem\*

Niels van Hoeffelen and Pablo Lanillos

Department of Artificial intelligence  
 Donders Institute for Brain, Cognition, and Behaviour  
 Radboud University  
 Montessorilaan 3, 6525HR Nijmegen, the Netherlands  
 niels.vanhoeffelen@ru.nl  
 p.lanillos@donders.ru.nl

**Abstract.** Despite the potential of active inference for visual-based control, learning the model and the preferences (priors) while interacting with the environment is challenging. Here, we study the performance of a deep active inference (dAIF) agent on OpenAI’s car racing benchmark, where there is no access to the car’s state. The agent learns to encode the world’s state from high-dimensional input through unsupervised representation learning. State inference and control are learned end-to-end by optimizing the expected free energy. Results show that our model achieves comparable performance to deep Q-learning. However, vanilla dAIF does not reach state-of-the-art performance compared to other world model approaches. Hence, we discuss the current model implementation’s limitations and potential architectures to overcome them.

**Keywords:** Deep Active Inference · Deep Learning · POMDP · Visual-based Control

## 1 Introduction

Learning from scratch which actions are relevant to succeed in a task using only high-dimensional visual input is challenging and essential for artificial agents and robotics. Reinforcement learning (RL) [26] is currently leading the advances in pixel-based control, e.g., the agent learns an action policy that maximizes the accumulated discounted rewards. Despite its dopamine biological inspiration, RL is far from capturing the physical processes happening in the brain. We argue, that prediction in any form (e.g., visual input, muscle feedback or dopamine) may be the driven motif of the general learning process of the brain [14]. Active inference [8,4], a general framework for perception, action and learning, proposes that the brain uses hierarchical generative models to predict incoming sensory data [6] and tries to minimize the difference between the predicted and observed sensory signals. This difference, mathematically described as the variational free

---

\* 2nd International Workshop on Active Inference IWAI2021, European Conference on Machine Learning (ECML/PCKDD 2021)

energy (VFE), needs to be minimized to generate better predictions about the world that causes these sensory signals [8,17]. Through acting on its environment, an agent can affect sensory signals to be more in line with predicted signals, which in turn leads to a decrease of the error between observed and predicted sensory signals.

AIF models have shown great potential in low-dimensional and discrete state spaces. To work in higher-dimensional state spaces, deep active inference (dAIF) has been proposed, which uses deep neural networks for approximating probability density functions (e.g., amortised inference) [17,27,29,2,23]. Interestingly, dAIF can be classified as a generalization of the world models approach [10] and can incorporate reward-based learning, allowing for a direct comparison to RL methods. Since the first attempt of dAIF [29], developments have happened concurrently in adaptive control [23,16] and in planning, exploiting discrete-time optimization, e.g., using the expected free energy [21,28]. In [17], a dAIF agent was tested on several environments in which the state is observable (Cartpole, Acrobot, Lunar-lander). In [5], a dAIF agent using Monte-Carlo sampling was tested on the Animal-AI environment. In [3], dAIF tackled the mountain car problem and was also tested on OpenAI’s car racing environment. For the car racing environment, they trained the dAIF agent on a handful of demonstration rollouts and compared it to a DQN that interacted with the environment itself. Their results showed that DQN needs a lot more interaction with the environment to start obtaining rewards compared to dAIF trained on human demonstrations of the task. Relevant for this work, in [11], a dAIF agent solved the Cartpole environment as both MDP and as POMDP instances, training the agent on just visual input.

In this paper, we study a dAIF agent<sup>1</sup> based on the proposed architectures in [17,11] for a more complex pixel-based control POMDP task, namely the OpenAI’s Car Racing environment [13], and discuss its advantages and limitations compared to other state-of-the-art models. The performance of the dAIF agent was shown to be in line with previous works and on-par with deep Q-learning. However, it did not achieve the performance of other world model approaches [1]. Hence, we discuss the reasons for this, as well as architectures that may help to overcome these limitations.

## 2 Deep Active Inference Model

The dAIF architecture studied is based on [11] and described in Fig. 1. It makes use of five networks to approximate the densities of Eq. (2): observation (encoding and decoding), transition, policy, and value. The full parameter description of the networks can be found in the Appendix A.

---

<sup>1</sup> The code can be found at [https://github.com/NTAvanHoeffelen/DAIF\\_CarRacing](https://github.com/NTAvanHoeffelen/DAIF_CarRacing)

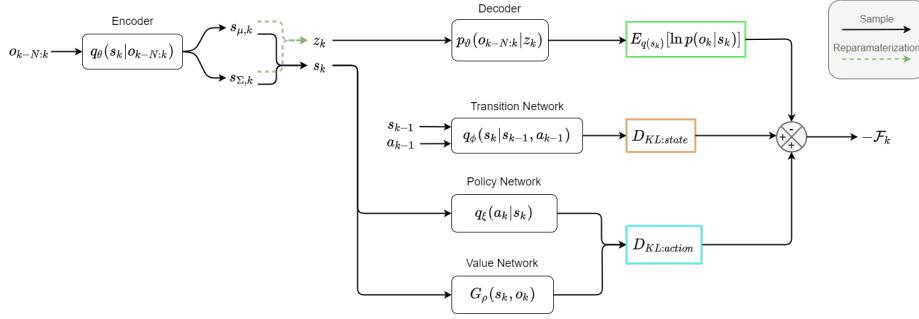


Fig. 1: Deep Active Inference architecture. Five deep artificial neural networks are used to model the observation encoding and decoding, the state transition, the policy and the EFE values. The architecture is trained end-to-end by optimizing the expected variational free energy.

**Variational Free Energy (VFE).** AIF agents infer their actions by minimizing the VFE expressed at instant  $k$  (with explicit action 1-step ahead) as:

$$\mathcal{F}_k = -\text{KL}[q(s_k, a_k) \parallel p(o_k, s_{0:k}, a_{0:k})] \quad (1)$$

Where  $s_k, o_k, a_k$  are the state, the observation and action respectively,  $q(s_k, a_k)$  is the recognition density, and  $p(o_k, s_{0:k}, a_{0:k})$  the generative model. Under the Markov assumption and factorizing [17,11], Eq. (1) can be rewritten as:

$$\mathcal{F}_k = \underbrace{E_{q(s_k)}[\ln p(o_k|s_k)]}_{\text{sensory prediction}} - \underbrace{\text{KL}[q(s_k) \parallel p(s_k|s_{k-1}, a_{k-1})]}_{\text{state prediction}} - \underbrace{\text{KL}[q(a_k|s_k) \parallel p(a_k|s_k)]}_{\text{action prediction}} \quad (2)$$

**Sensory prediction.** The observation network predicts the observations—first term in Eq. (2)—and encodes the high-dimensional input to states  $q_\theta(s_k|o_{k-N:k})$  and decodes latent spaces to observations  $p_\theta(o_{k-N:k}|z_k)$ . It can be implemented with a variational autoencoder, where latent representation is described as a multivariate Gaussian distribution with mean  $s_\mu$  and variance  $s_\Sigma$ . The latent space  $z_k$  is obtained using the reparametrisation trick. To train the network, we use the binary cross-entropy and the KL regularizer to force the latent space to be Gaussian:

$$\begin{aligned} L_{o,k} &= -E_{q_\theta(s_k|o_{k-N:k})}[\ln p_\theta(o_{k-N:k}|z_k)] \\ &= BCE(\hat{o}_{k-N:k}, o_{k-N:k}) - \frac{1}{2} \sum (1 + \ln s_{\Sigma,k} - s_{\mu,k}^2 - s_{\Sigma,k}) \end{aligned} \quad (3)$$

**State prediction.** The transition network models a distribution that allows the agent to predict the state at time  $k$  given the state and action at time

$k - 1$ , where the input state consists of both the mean  $s_\mu$  and variance  $s_\Sigma$ . Under the AIF approach this is the difference between the state distribution (generated by the transition network) and the actual observed state (from the encoder):  $\mathbf{KL}[q(s_k) \parallel p(s_k|s_{k-1}, a_{k-1})]$ . For the sake of simplicity, we define a feed-forward network that computes the maximum-a-posteriori estimate of the predicted state  $\hat{s}_k = q_\phi(s_{k-1}, a_{k-1})$  and train it using the mean squared error:

$$MSE(s_{\mu,k}, q_\phi(s_{k-1}, a_{k-1})) \quad (4)$$

**Action prediction.** We use two networks to evaluate action: the policy and value network. The action prediction part of Eq. (2) is the difference between the model's action distribution and the “optimal” true distribution. It is a KL divergence, which can be split into an energy and an entropy term:

$$\begin{aligned} \mathbf{KL}[q(a_k|s_k) \parallel p(a_k|s_k)] &= - \sum_a q(a_k|s_k) \ln \frac{p(a_k|s_k)}{q(a_k|s_k)} \\ &= \underbrace{- \sum_a q(a_k|s_k) \ln p(a_k|s_k)}_{\text{energy}} - \underbrace{\sum_a q(a_k|s_k) \ln q(a_k|s_k)}_{\text{entropy}} \end{aligned} \quad (5)$$

The policy network models the distribution over actions at time  $k$  given the state at time  $k$   $q_\xi(a_k|s_k)$ . It is implemented as a feed-forward neural network that returns a distribution over actions given a state.

The value network computes the Expected Free Energy (EFE) [21,17,11] which is used to model the true action posterior  $p(a_k|s_k)$ . As the true action posterior is not exactly known, we assume that prior belief makes the agent select policies that minimize the EFE. We model the distribution over actions as a precision-weighted Boltzmann distribution over the EFE [21,17,5,24]:

$$p(a_k|s_k) = \sigma(-\gamma G(s_{k:N}, o_{k:N})) \quad (6)$$

where  $G(s_{k:N}, o_{k:N})$  is the EFE for a set of states and observations up to some future time  $N$ . As we are dealing with discrete time steps, it can be written as a sum over these time steps:

$$G(s_{k:N}, o_{k:N}) = \sum_k^N G(s_k, o_k) \quad (7)$$

The EFE is evaluated for every action, because of this we implicitly conditioned on every action [17]. We then define the EFE of a single time step as<sup>2</sup>:

$$\begin{aligned} G(s_k, o_k) &= \mathbf{KL}[q(s_k) \parallel p(s_k, o_k)] \\ &\approx -\ln p(o_k) + \mathbf{KL}[q(s_k) \parallel q(s_k|o_k)] \\ &\approx -r(o_k) + \mathbf{KL}[q(s_k) \parallel q(s_k|o_k)] \end{aligned} \quad (8)$$

---

<sup>2</sup> The full derivation can be found in Appendix D

The negative log-likelihood (or surprise) of an observation  $-\ln p(o_t)$ , is replaced by the reward  $-r(o_k)$  [17,7,11]. As AIF agents act to minimize their surprise, by replacing the surprise with the negative reward, we encode the agent with the prior that its goal is to maximize reward. This formulation needs the EFE computation for all of the possible states and observations up to some time  $N$ , making it computationally intractable. Tractability has been achieved through bootstrapping [17,11] and combining Monte-Carlo tree search and amortized inference [5]. Here we learn a bootstrapped estimate of the EFE. The value network is used to get an estimate of the EFE for all of the possible actions. It is modelled as a feed-forward neural network  $G_\rho(s_k, o_k) = f_\rho(s_k)$ .

To train the value network, we use another bootstrapped EFE estimate which uses the EFE of the current time step and a  $\beta \in (0, 1]$  discounted value-net estimate of the EFE under  $q(a_{k+1}|s_{k+1})$  for the next time step:

$$\hat{G}(s_k, o_k) = -r(o_k) + \mathbf{KL}[q(s_k) \parallel q(s_k|o_k)] + \beta E_{q(a_{k+1}|s_{k+1})}[G_\rho(s_{k+1}, o_{k+1})] \quad (9)$$

Using gradient descent, we can optimize the parameters of the value network by computing the MSE between  $G_\rho(s_k, o_k)$  and  $\hat{G}(s_k, o_k)$ :

$$L_{f_\rho, k} = \text{MSE}(G_\rho(s_k, o_k), \hat{G}(s_k, o_k)) \quad (10)$$

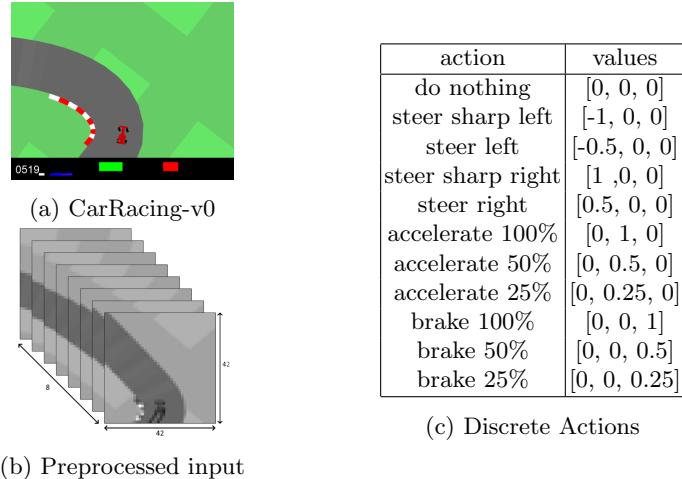
In summary, with our implementation, the VFE loss function becomes:

$$\begin{aligned} -\mathcal{F}_k = & \text{BCE}(\hat{o}_{k-N:k}, o_{k-N:k}) - \frac{1}{2} \sum (1 + \ln s_{\Sigma, k} - s_{\mu, k}^2 - s_{\Sigma, k}) \\ & + \text{MSE}(s_{\mu, k}, q_\phi(s_{k-1}, a_{k-1})) \\ & + \mathbf{KL}[q_\xi(a_k|s_k) \parallel \sigma(-\gamma G_\rho(s_k, o_k))] \end{aligned} \quad (11)$$

### 3 Experimental Setup

We evaluated the algorithm on OpenAI’s CarRacing-v0 environment [13] (Fig. 2a). It is considered a Partial Observable Markov Decision Process (POMDP) problem as there is no access to the state of the agent/environment. The input is the top-view image ( $96 \times 96$  RGB) of part of the racing track centred on the car. The goal of this 2D game is to maximize the obtained reward by driving as fast and precise as possible. A reward of  $+1000/N$  is received for every track tile that is visited, where  $N$  is the total number of tiles (placed on the road), and a reward of  $-0.1$  is received for every frame that passes. Solving the game entails that an agent scores an average of more than 900 points over 100 consecutive episodes. By definition, an episode is terminated after the agent visited all track tiles, the agent strayed out of bounds of the environment, or when 1000 time steps have elapsed. Every episode, a new race track is randomly generated.

The agent/car has three continuous control variables, namely steering  $[-1, 1]$  (left and right), accelerating  $[0, 1]$ , and braking  $[0, 1]$ . The action space was discretized into 11 actions, similarly to [31,25,30], described in Table 2c.



## 4 Results

We compared our dAIF implementation with other state-of-the-art algorithms. First, Fig. 3 shows the average reward evolution while training for our dAIF architecture, Deep-Q learning (DQN) [19] (our implementation) and a random agent. Second, Table 1 shows the average reward over 100 consecutive episodes for the top methods in the literature. The average reward performance test and reward per episode for DQN and dAIF are provided in Appendix E.

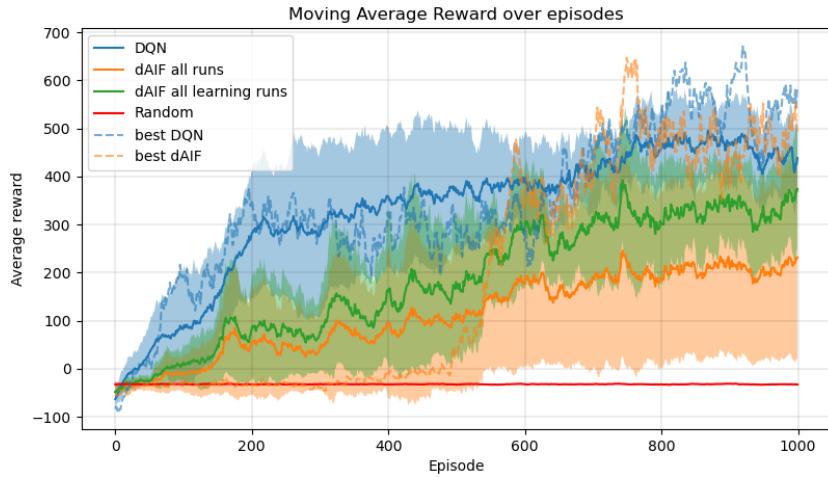


Fig. 3: Moving average reward (MAR) comparison for OpenAI’s CarRacing-v0.  $MAR_e = 0.1CR_e + 0.9MAR_{e-1}$ , where  $CR_e$  is the cumulative reward of the current episode. In green (solid line), the mean of the dAIF runs that were able to learn a policy, and in orange (dashed line), the best of all training runs.

For the dAIF and the DQN implementations, observations are first preprocessed by removing the bottom part of the image. This part contains information about the accumulated rewards of the current episode, the car’s true speed, four ABS sensors, steering wheel position, and gyroscope. Afterwards, the image is grey-scaled and reshaped to a size of  $42 \times 42$ . The input is defined as a stack of  $k$  to  $k - N$  observations to provide temporal information by allowing the encoding of velocity and steering. We use experience replay [15] with a batch size of 250 and memory capacity of 300000 and 100000 transitions for DQN and dAIF respectively, and make use of target networks [19] (copy of the policy network for DQN and value network for dAIF) with a freeze period of 50 time steps.

The dAIF agent makes use of a pre-trained VAE which was frozen during training. Following a similar procedure as [3], the VAE was pre-trained on observations collected by having a human play the environment for 10000 time steps.

Table 1: Average rewards for CarRacing-v0

| Method                              | Average Reward |
|-------------------------------------|----------------|
| DQN (our implementation)            | $515 \pm 162$  |
| dAIF (our implementation)           | $494 \pm 241$  |
| A3C (Continuous) [18]               | $591 \pm 45$   |
| A3C (Discrete) [12]                 | $652 \pm 10$   |
| Weight Agnostic Neural Networks [9] | $893 \pm 74$   |
| GA [22]                             | $903 \pm 72$   |
| World models [10]                   | $906 \pm 21$   |

## 5 Discussion

The dAIF implementation described in this paper has shown to reach performance on par with Deep Q-learning. However, there are some remarks. First, it showed a slower learning curve as described in previous works [11], due to the need to learn the world model. Second, we identified some runs where the system was not able to learn—See Fig. 3 orange solid line. These runs drag down the average performance. Finally, it has failed to reach state-of-the-art performance when comparing to other world model approaches—See Table 1. Here we discuss the limitations of the current implementation and alternative architectures to overcome the challenge of learning the preferences in dAIF approaches.

**Observation and transition model.** Our implementation does not fully exploit temporal representation learning, such as other models that use recurrent neural networks (RNN). Figure 4 describes two architectures to learn the encoding and the transitioning of the environment. Figure 4 left shows the autoencoding and transition model implemented in this work. This architecture is similar

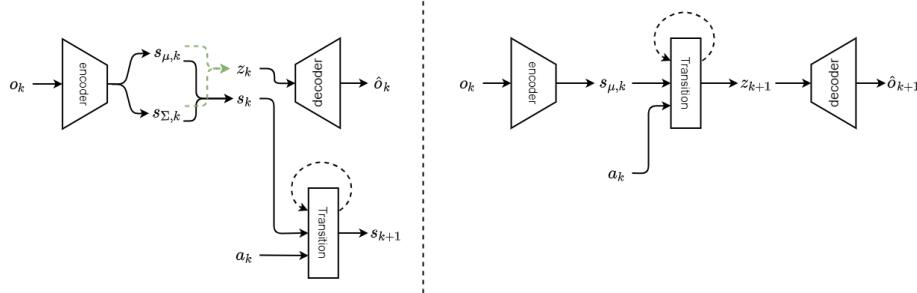


Fig. 4: Transition network outside of the observation network (left) and the transition network in between the encoder and decoder (right).

to the successful world models [10], but while we used a simple feed-forward network, they modelled the state-transition with a mixed density network RNN. Interestingly, dAIF also permits the alternative architecture described in Fig. 4 right (also proposed in [20]), in which the network learns to predict future observations. Here the transition network is part of the autoencoding. By incorporating the transition network in the structure of the observation network, we avoid the need for the dual objectives: perceptual reconstruction and dynamics learning. Preliminary testing did not show any improvements. Future work could involve more extensive testing to uncover possible performance improvements.

**Dependency of input space** The performance of dAIF has shown a strong dependency on the learning of the observation model. Different image pre-processing methods would lead to improvements of more than 50% in the agent performance, as shown in other DQN implementations in the literature. Testing showed that without a pre-trained observation network, the model was unable to learn consistently and rarely showed performance that would suggest an indication of task comprehension. By using a pre-trained observation network learning occurred in 6 out of the 10 runs. To produce a proper action-centric representation, likelihood, transition and control should be learnt concurrently. However, parameters uncertainty may be tackled as the models are being learnt. The current implementation uses static values for the networks learning rates, future testing could investigate different variable learning rates for each network or decaying dropout temperature.

**Bootstrapping of the policy and the value.** Estimating both the policy and the value from state encoding has shown end-to-end issues when we do not pre-train the observation model. In particular, 1-step ahead action formulation in conjunction with bootstrapping might not capture a proper structure of the world, which is needed to complete the task, even if we use several consecutive input images to compute the state. N-step ahead observation optimization EFE formulations, as proposed in [5,28,20], may aid learning. Particularly, when sub-

stituting the negative log surprise by the rewards, the agent might loose the exploratory AIF characteristic, thus focusing only on goal-oriented behaviour. Furthermore, and very relevant, the reward implementation in CarRacing-v0 might be not the best way to provide dAIF with rewards for proper preference learning.

## References

1. Openai's carracing-v0 leaderboard. <https://github.com/openai/gym/wiki/Leaderboard#carracing-v0>
2. Çatal, O., Nauta, J., Verbelen, T., Simoens, P., Dhoedt, B.: Bayesian policy selection using active inference. arXiv preprint arXiv:1904.08149 (2019)
3. Çatal, O., Wauthier, S., De Boom, C., Verbelen, T., Dhoedt, B.: Learning generative state space models for active inference. *Frontiers in Computational Neuroscience* **14**, 103 (2020)
4. Da Costa, L., Parr, T., Sajid, N., Veselic, S., Neacsu, V., Friston, K.: Active inference on discrete state-spaces: a synthesis. *Journal of Mathematical Psychology* **99**, 102447 (2020)
5. Fountas, Z., Sajid, N., Mediano, P.A., Friston, K.: Deep active inference agents using monte-carlo methods. arXiv preprint arXiv:2006.04176 (2020)
6. Friston, K.: A theory of cortical responses. *Philosophical transactions of the Royal Society B: Biological sciences* **360**(1456), 815–836 (2005)
7. Friston, K., Samothrakis, S., Montague, R.: Active inference and agency: optimal control without cost functions. *Biological cybernetics* **106**(8), 523–541 (2012)
8. Friston, K.J., Daunizeau, J., Kilner, J., Kiebel, S.J.: Action and behavior: a free-energy formulation. *Biological cybernetics* **102**(3), 227–260 (2010)
9. Gaier, A., Ha, D.: Weight agnostic neural networks. arXiv preprint arXiv:1906.04358 (2019)
10. Ha, D., Schmidhuber, J.: World models. arXiv preprint arXiv:1803.10122 (2018)
11. van der Himst, O., Lanillos, P.: Deep active inference for partially observable mdps. In: International Workshop on Active Inference. pp. 61–71. Springer (2020)
12. Khan, M., Elibol., O.: Car racing using reinforcement learning (2018), <https://web.stanford.edu/class/cs221/2017/restricted/p-final/elibol/final.pdf>
13. Klimov, O.: Carracing-v0. <https://gym.openai.com/envs/CarRacing-v0/>
14. Lanillos, P., van Gerven, M.: Neuroscience-inspired perception-action in robotics: applying active inference for state estimation, control and self-perception. arXiv preprint arXiv:2105.04261 (2021)
15. Lin, L.: Reinforcement learning for robots using neural networks (1992)
16. Meo, C., Lanillos, P.: Multimodal vae active inference controller. arXiv preprint arXiv:2103.04412 (2021)
17. Millidge, B.: Deep active inference as variational policy gradients. *Journal of Mathematical Psychology* **96**, 102348 (2020)
18. Min J. Jang, S., Lee, C.: Reinforcement car racing with a3c (2017), <https://www.scribd.com/document/358019044/Reinforcement-Car-Racing-with-A3C>
19. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. *nature* **518**(7540), 529–533 (2015)
20. Noel, A.D., van Hoof, C., Millidge, B.: Online reinforcement learning with sparse rewards through an active inference capsule. arXiv preprint arXiv:2106.02390 (2021)

21. Parr, T., Friston, K.J.: Generalised free energy and active inference. *Biological cybernetics* **113**(5), 495–513 (2019)
22. Risi, S., Stanley, K.O.: Deep neuroevolution of recurrent and discrete world models. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 456–462 (2019)
23. Sancaktar, C., van Gerven, M.A., Lanillos, P.: End-to-end pixel-based deep active inference for body perception and action. In: 2020 Joint IEEE 10th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob). pp. 1–8. IEEE (2020)
24. Schwartenbeck, P., Paszke, J., Hauser, T.U., FitzGerald, T.H., Kronbichler, M., Friston, K.J.: Computational mechanisms of curiosity and goal-directed exploration. *Elife* **8**, e41703 (2019)
25. Slik, J.: Deep reinforcement learning for end-to-end autonomous driving (2019)
26. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
27. Tschantz, A., Baltieri, M., Seth, A.K., Buckley, C.L.: Scaling active inference. In: 2020 International Joint Conference on Neural Networks (IJCNN). pp. 1–8. IEEE (2020)
28. Tschantz, A., Millidge, B., Seth, A.K., Buckley, C.L.: Reinforcement learning through active inference. arXiv preprint arXiv:2002.12636 (2020)
29. Ueltzhöffer, K.: Deep active inference. *Biological cybernetics* **112**(6), 547–573 (2018)
30. van der Wal, D., Intelligentie, B.O.K., Shang, W.: Advantage actor-critic methods for car racing (2018)
31. Zhang, Y.: Deep reinforcement learning with mixed convolutional network. arXiv preprint arXiv:2010.00717 (2020)

## A Model Parameters

Table 2: General parameters

| Parameter             | Value | Description   |
|-----------------------|-------|---|
| $N_{screens}$         | 8     | Size of the observation stack   |
| $N_{colour}$          | 1     | Colour channels of the input image  |
| $N_{height}$          | 42    | Height in pixels of the input image   |
| $N_{width}$           | 42    | Width in pixels of the input image  |
| $N_{actions}$         | 11    | Number of actions the agent can select from   |
| $N_{episodes}$        | 1000  | Number of episodes the model is trained for   |
| $N_{length\_episode}$ | 1000  | The maximum amount of time steps in an episode  |
| Freeze period         | 50    | The amount of time steps the target network is frozen before copying the parameters of the policy/value network |
| Batch size            | 250   | Number of items in a mini-batch   |

Table 3: DQN parameters

| Parameter       | Value       | Description   |
|-----------------|-------------|---|
| Policy network  |             | Convolutional neural network which estimates Q-values given a state see Appendix B.   |
| Target network  |             | Copy of the Policy network which is updated after each freeze period see Appendix B.  |
| $N_{hidden}$    | 512         | Number of hidden units in the policy and target network   |
| $\lambda$       | 1e-5        | Learning rate   |
| $\gamma$        | 0.99        | Discount factor   |
| $\epsilon$      | 0.15 → 0.05 | Probability of selecting a random action. (Starts as 0.15, decreases linearly per episode with 0.00015 until a minimum of 0.05) |
| Memory capacity | 300000      | Number of transitions the replay memory can store   |

Table 4: dAIF parameters

| Parameter              | Value  | Description   |
|------------------------|--------|---|
| Observation network    |        | VAE; see Appendix C.  |
| Transition network     |        | Feed-forward neural network of shape:<br>$(2N_{latent} + 1) \times N_{hidden} \times N_{actions}$                                   |
| Policy network         |        | Feed-forward neural network of shape:<br>$2N_{latent} \times N_{hidden} \times N_{actions}$ ; with a softmax function on the output |
| Value network          |        | Feed-forward neural network of shape:<br>$2N_{latent} \times N_{hidden} \times N_{actions}$   |
| Target network         |        | Copy of the Value network which is updated after each freeze period   |
| $N_{hidden}$           | 512    | Number of hidden units in the transition, policy, and value network.  |
| $N_{latent}$           | 128    | Size of the latent state  |
| $\lambda_{transition}$ | 1e-3   | Learning rate of the transition network   |
| $\lambda_{policy}$     | 1e-4   | Learning rate of the policy network   |
| $\lambda_{value}$      | 1e-5   | Learning rate of the value network  |
| $\lambda_{VAE}$        | 5e-6   | Learning rate of the VAE  |
| $\gamma$               | 12     | Precision parameter   |
| $\beta$                | 0.99   | Discount factor   |
| $\alpha$               | 18000  | $\frac{1}{\alpha}$ is multiplied with the VAE loss to scale its size to that of the other term in the VFE                           |
| Memory capacity        | 100000 | Number of transitions the replay memory can store   |

## B DQN: Policy network

Table 5: Layers DQN policy network

| Type      | out channels | kernel | stride | input           | output          |
|-----------|--------------|--------|--------|-----------------|-----------------|
| conv      | 64           | 4      | 2      | (1, 8, 42, 42)  | (1, 64, 20, 20) |
| batchnorm | -            | -      | -      | -               | -               |
| maxpool   | -            | 2      | 2      | (1, 64, 20, 20) | (1, 64, 10, 10) |
| relu      | -            | -      | -      | -               | -               |
| conv      | 128          | 4      | 2      | (1, 64, 10, 10) | (1, 128, 4, 4)  |
| batchnorm | -            | -      | -      | -               | -               |
| maxpool   | -            | 2      | 2      | (1, 128, 4, 4)  | (1, 128, 2, 2)  |
| relu      | -            | -      | -      | -               | -               |
| conv      | 256          | 2      | 2      | (1, 128, 2, 2)  | (1, 256, 1, 1)  |
| relu      | -            | -      | -      | -               | -               |
| dense     | -            | -      | -      | 256             | 512             |
| dense     | -            | -      | -      | 512             | 11              |

## C VAE

Table 6: VAE layers

| Type                | out channels | kernel | stride | input           | output          |         |  |
|---------------------|--------------|--------|--------|-----------------|-----------------|---------|--|
| conv                | 32           | 4      | 2      | (1, 8, 42, 42)  | (1, 32, 20, 20) | Encoder |  |
| batchnorm           |              |        |        |                 |                 |         |  |
| relu                |              |        |        |                 |                 |         |  |
| conv                | 32           | 4      | 2      | (1, 32, 20, 20) | (1, 64, 9, 9)   |         |  |
| batchnorm           |              |        |        |                 |                 |         |  |
| relu                |              |        |        |                 |                 |         |  |
| conv                | 128          | 5      | 2      | (1, 64, 9, 9)   | (1, 128, 3, 3)  |         |  |
| batchnorm           |              |        |        |                 |                 |         |  |
| relu                |              |        |        |                 |                 |         |  |
| conv                | 256          | 3      | 2      | (1, 128, 3, 3)  | (1, 256, 1, 1)  |         |  |
| relu                |              |        |        |                 |                 |         |  |
| dense               | -            | -      | -      | 256             | 128             | Decoder |  |
| dense $\mu$         | -            | -      | -      | 128             | 128             |         |  |
| dense $\log \Sigma$ | -            | -      | -      | 128             | 128             |         |  |
| dense               | -            | -      | -      | 128             | 128             |         |  |
| dense               | -            | -      | -      | 128             | 256             |         |  |
| deconv              | 128          | 3      | 2      | (1, 256, 1, 1)  | (1, 128, 3, 3)  |         |  |
| batchnorm           |              |        |        |                 |                 |         |  |
| relu                |              |        |        |                 |                 |         |  |
| deconv              | 64           | 5      | 2      | (1, 128, 3, 3)  | (1, 64, 9, 9)   |         |  |
| batchnorm           |              |        |        |                 |                 |         |  |
| relu                |              |        |        |                 |                 |         |  |
| deconv              | 32           | 4      | 2      | (1, 64, 9, 9)   | (1, 32, 20, 20) |         |  |
| batchnorm           |              |        |        |                 |                 |         |  |
| relu                |              |        |        |                 |                 |         |  |
| deconv              | 8            | 4      | 2      | (1, 32, 20, 20) | (1, 8, 42, 42)  |         |  |
| batchnorm           |              |        |        |                 |                 |         |  |
| relu                |              |        |        |                 |                 |         |  |
| sigmoid             |              |        |        |                 |                 |         |  |

## D Derivations

Derivation for the EFE for a single time step:

$$\begin{aligned}
G(s_k, o_k) &= \mathbf{KL}[q(s_k) \parallel p(s_k, o_k)] \\
&= \int q(s_k) \ln \frac{q(s_k)}{p(s_k, o_k)} \\
&= \int q(s_k) \ln q(s_k) - \ln p(s_k, o_k) \\
&= \int q(s_k) \ln q(s_k) - \ln p(s_k|o_k) - \ln p(o_k) \\
&\approx \int q(s_k) \ln q(s_k) - \ln q(s_k|o_k) - \ln p(o_k) \\
&\approx -\ln p(o_k) + \int q(s_k) \ln q(s_k) - \ln q(s_k|o_k) \\
&\approx -\ln p(o_k) + \int q(s_k) \ln \frac{q(s_k)}{q(s_k|o_k)} \\
&\approx -\ln p(o_k) + \mathbf{KL}[q(s_k) \parallel q(s_k|o_k)] \\
&\approx -r(o_k) + \mathbf{KL}[q(s_k) \parallel q(s_k|o_k)]
\end{aligned}$$

## E Average Reward over 100 episodes

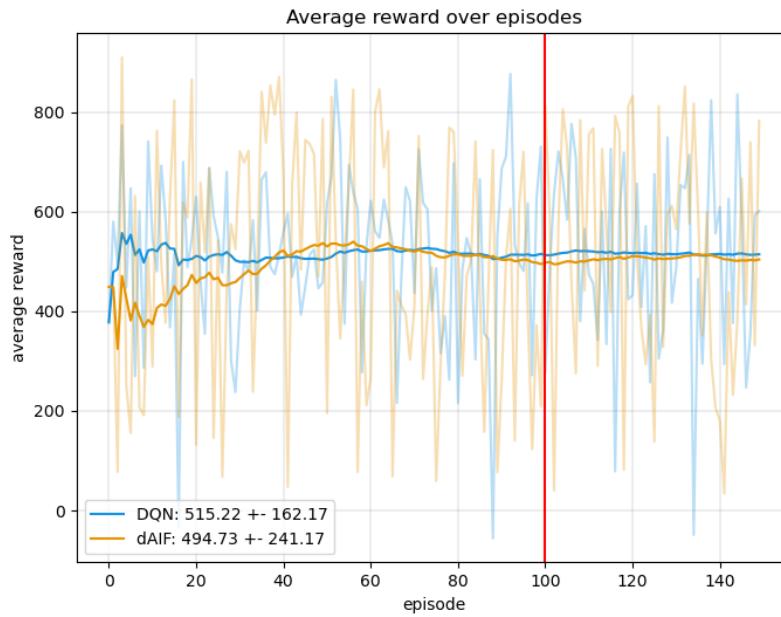


Fig. 5: Average reward test over 100 episodes for DQN and dAIF. The bright lines show the mean over episodes. The transparent lines show the reward that was obtained in a particular episode.