Deep Active Inference with Monte-Carlo Tree Search in a 2D Zebrafish Foraging Environment

Anonymous

Abstract

We present a deep active inference agent equipped with Monte-Carlo Tree Search (MCTS) and a habitual policy network, adapted from the Monte-Carlo active inference framework of Fountas et al. (NeurIPS 2020). Our agent operates in a continuous 2D zebrafish-inspired environment with partial observability through a 1D egocentric vision strip and an internal hunger state. The agent learns a latent-state world model (encoder, decoder, and transition dynamics) and selects actions by minimizing expected free energy (EFE) approximated using Monte-Carlo sampling and MC-dropout. We describe the environment, model architecture, EFE objective (including preference priors), MCTS planning procedure, and an off-policy replay-based training pipeline.

1 Introduction

Active inference frames perception and action as approximate Bayesian inference under a generative model, where agents act to minimize (expected) free energy. Recent deep active inference work introduced scalable Monte-Carlo (MC) estimators and planning via Monte-Carlo Tree Search (MCTS), along with a habitual network that amortizes the planner's policy distribution [1]. In this work we implement a zebrafish-inspired agent using the same high-level ingredients: (i) a latent generative model of observations and state transitions, (ii) EFE-based action valuation with epistemic terms, (iii) MCTS for policy selection, and (iv) a habitual policy network trained to match the planner.

Our implementation differs from the original dSprites/Animal-AI settings in its observation modality (1D RGB vision rays) and task structure (foraging while avoiding predators and obstacles), but retains the core theory and computational structure.

Codebase. The agent, networks, training loop, and environment are implemented in PyTorch and Gymnasium. The main agent and MCTS node logic are in `agent_vision.py`, the neural modules in `models_vision.py`, the training script in `train_deep_agent_vision.py`, and the environment in `zebrafish_env_vision.py`. These implement: discrete actions, a 1D vision strip and hunger observation, an explicit predator-distance ground truth for supervised belief learning, and an EFE decomposition with risk/ambiguity/epistemic terms.

2 Related Work

Fountas et al. introduced *Deep Active Inference agents using Monte-Carlo methods* (DAIMC) that leverage MCTS to search action policies minimizing expected free energy and use a habitual network to amortize the planner's distribution [1]. EFE is decomposed into preference (risk) and epistemic terms for state and parameter uncertainty (Eq. (8a–c) in [1]). Our implementation follows these principles and adapts them to a zebrafish-like 2D environment with visual rays and internal hunger.

3 Environment: 2D Zebrafish Foraging with Predator and Obstacles

3.1 State and Dynamics

The environment is a continuous 2D world of size 800×600 with a fish agent, multiple food items, optional rectangular obstacles, and an optional predator that pursues the fish. The fish has continuous position, heading, and speed, updated with turning, thrust, drag, boundary bounce, and obstacle collision resolution. The predator moves with a bounded turn rate toward the fish at a fixed speed.

3.2 Action Space

The agent uses a discrete action space of size $|\mathcal{A}| = 3$:

- **LEFT**: turn left (no thrust)
- **RIGHT**: turn right (no thrust)
- **FORWARD**: thrust forward (no turn)

This is implemented as a Gymnasium `Discrete(3)` action space.

3.3 Observations

The observation is a dictionary $\mathbf{o}_t = \{\texttt{vision}_t, \texttt{hunger}_t\}$:

- **Vision strip:** a 1D RGB array of width $L = 160$ generated by casting rays within a field of view (FOV) and shading by depth. Each pixel encodes the nearest hit among food, predator, obstacles, or boundary walls.
- **Hunger:** a scalar in $[0, 1]$ decaying over time and increasing when food is eaten.

Additionally, the environment provides ground-truth predator distance (normalized) as an information signal:

$$d_t^{\text{gt}} = \frac{\|\mathbf{x}_t^{\text{fish}} - \mathbf{x}_t^{\text{pred}}\|}{\sqrt{W^2 + H^2}} \in [0, 1], \tag{1}$$

and sets $d_t^{\text{gt}} = 1$ when the predator is disabled. This scalar is returned in `info["pred_dist_gt"]`.

3.4 Rewards and Termination

Although the agent is trained primarily by active inference losses rather than reward maximization, the environment supplies a scalar reward for monitoring: small step penalty, positive reward for eating, termination penalties for predation or starvation, and a completion reward when all food is collected. Episodes end upon predation, hunger depletion, food exhaustion, or time limit.

4 Agent Architecture

4.1 Latent State Model

We maintain a latent state $\mathbf{z}_t \in \mathbb{R}^D$ with $D = 10$ (default in our training configuration). The agent includes:

- **Encoder** $q_\phi(\mathbf{z}_t|\mathbf{o}_t)$: a Conv1D encoder mapping the flattened RGB strip (and auxiliary hunger) to mean/log-variance of a diagonal Gaussian.
- **Decoder** $p_\theta(\mathbf{o}_t|\mathbf{z}_t)$: a deconvolutional decoder producing a continuous vision mean and log-variance, plus an auxiliary hunger head.
- **Transition** $p_\theta(\mathbf{z}_{t+1}|\mathbf{z}_t, \mathbf{a}_t)$: an MLP transition model outputting mean/log-variance of a diagonal Gaussian. Dropout is used to enable MC-dropout sampling for parameter uncertainty.
- **Habitual policy** $q_\psi(\mathbf{a}_t|\mathbf{z}_t)$: a softmax policy network mapping latent states to action probabilities, trained to match the planner distribution.

Supervised predator-distance head. In addition to the decoder outputs, the agent includes a separate head $h(\mathbf{z}_t) \in [0, 1]$ predicting predator distance from latent state. This is trained with a supervised loss against `pred_dist_gt` for improved state belief calibration in partially observable settings.

4.2 Monte-Carlo Reparameterization

We sample latent states via:

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\epsilon} \odot \exp\left(\tfrac{1}{2} \log \boldsymbol{\sigma}^2\right), \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I). \tag{2}$$

5 Expected Free Energy Objective

Following [1], we approximate EFE as a sum of three conceptual components:

$$G(\mathbf{a}_t) \approx \underbrace{\text{Risk}}_{\text{preferences}} + \underbrace{\text{Ambiguity}}_{\text{observation entropy}} - \underbrace{\text{State info gain}}_{\text{epistemic}} - \underbrace{\text{Parameter info gain}}_{\text{epistemic}}. \tag{3}$$

We compute these one-step quantities for each candidate action and use them inside MCTS rollouts.

5.1 Preference Term (Risk)

We encode preferences over predicted observations using soft features extracted from the decoded vision mean (e.g., food presence near the center, predator presence, obstacle proximity), as well as hunger/satiation and predator-distance preference. Concretely, we define Bernoulli preference priors for each feature f and compute a negative log-likelihood:

$$\text{Risk}(\mathbf{z}) = \sum_f w_f \left(- \left[\hat{p}_f \log p_f^\star + (1 - \hat{p}_f) \log(1 - p_f^\star)\right]\right), \tag{4}$$

where $\hat{p}_f$ is the feature probability derived from decoded observations and $p_f^\star$ is the preferred probability (e.g., high for food-visible, low for predator-visible).

5.2 Ambiguity Term

Ambiguity is approximated as the entropy of the predicted observation distribution. Since the decoder outputs per-dimension log-variance for the vision strip, we use Gaussian diagonal entropy:

$$H(\mathbf{o}|\mathbf{z}) = \sum_i \tfrac{1}{2}\Big(1 + \log(2\pi)\Big) + \tfrac{1}{2}\log\sigma_i^2, \tag{5}$$

scaled to account for observation dimensionality.

5.3 State Epistemic Value

We approximate the reduction in state uncertainty after a hypothetical observation $\mathbf{o}'$ is sampled from the decoder:

$$\mathrm{IG}_{\mathrm{state}} \approx H\big[p(\mathbf{z}'|\mathbf{z},\mathbf{a})\big] - \mathbb{E}_{\mathbf{o}'\sim p(\mathbf{o}'|\mathbf{z}')}\left[H\big[q(\mathbf{z}'|\mathbf{o}')\big]\right]. \tag{6}$$

Operationally, we sample $\mathbf{o}'$ from the decoder (using its predicted mean and variance) and re-encode it to estimate the posterior entropy. This matches the idea of Eq. (8b) in [1].

5.4 Parameter Epistemic Value via MC-Dropout

Parameter uncertainty is approximated using MC-dropout sampling of transition and decoder parameters. Following the spirit of Eq. (8c) in [1], we estimate a mutual-information-like difference between the entropy of the predictive mixture and the expected entropy under sampled parameters:

$$\mathrm{IG}_\theta \approx H\big[p(\mathbf{o}'|\mathbf{z},\mathbf{a})\big] - \mathbb{E}_\theta\left[H\big[p_\theta(\mathbf{o}'|\mathbf{z},\mathbf{a})\big]\right]. \tag{7}$$

6 Planning with Monte-Carlo Tree Search

6.1 Node Statistics and Selection

We represent a search node by a latent state $\mathbf{z}$ and store per-action counts $N(a)$ and values $W(a)$, where W accumulates $-G$ (so larger is better). The action prior $Q_\psi(a|\mathbf{z})$ (habitual network) is also stored at each node.

Selection uses a PUCT-like score with exploration constant C:

$$\mathrm{score}(a) = Q(a) + C \cdot \frac{\sqrt{\sum_{a'} N(a') + 1}}{1 + N(a)} \cdot \tilde{Q}_\psi(a|\mathbf{z}), \tag{8}$$

where $\tilde{Q}_\psi$ optionally scales exploration using the habitual prior.

6.2 Expansion

On expansion, we generate next-state predictions for all actions using the transition model, sample (or take means) for $\mathbf{z}_{t+1}$, compute $G(a)$ for each action, initialize W, N (optionally), and create child nodes.

6.3 Rollout Policy

During simulated rollouts from a leaf, actions are sampled from the habitual policy (with a small uniform mixture for exploration). The rollout returns an average EFE over depth.

Algorithm 1 DAIMC-style Planning in the Zebrafish Environment

Require: Observation $\mathbf{o}_t$, models $(q_\phi, p_\theta, p_\theta^{\text{trans}}, q_\psi)$, repeats R, rollout depth D

Infer latent mean $\boldsymbol{\mu}_t$ via encoder; set root state $\mathbf{z}_t \leftarrow \boldsymbol{\mu}_t$
Compute habitual prior $Q_\psi(a|\mathbf{z}_t)$
if habitual shortcut condition satisfied **then**
    Sample/argmax $a_t \sim Q_\psi(a|\mathbf{z}_t)$ and **return**
end if
Expand root: compute children and one-step $G(a)$
for $r = 1$ to R **do**
    Select a path by maximizing PUCT-like scores
    Expand leaf if needed
    Roll out D steps using habitual policy; compute average G_{roll}
    Backpropagate G_{roll} along the path
end for
Form planner distribution $P_{\text{plan}}(a|\mathbf{z}_t) \propto N(a)$
Sample/argmax $a_t \sim P_{\text{plan}}(a|\mathbf{z}_t)$ and **return**

6.4 Backpropagation and Action Choice

Backpropagation updates each traversed edge (s, a) by:

$$W(a) \leftarrow W(a) - G, \quad N(a) \leftarrow N(a) + 1. \tag{9}$$

After planning, the action distribution is taken as normalized visit counts:

$$P_{\text{plan}}(a|\mathbf{z}) = \frac{N(a)}{\sum_{a'} N(a')}. \tag{10}$$

The final executed action is sampled (or chosen greedily) from P_{plan}.

Habitual shortcut. To reduce compute, planning may be skipped when the habitual policy is already confident (low entropy or high max-probability). In that case, the agent directly samples from $Q_\psi(a|\mathbf{z})$.

7 Learning and Optimization

7.1 Replay-Based Off-Policy Training

Training proceeds by interacting with the environment, running MCTS (or habitual shortcut) to obtain planner probabilities, and storing transitions in a replay buffer:

$$(\mathbf{o}_t, a_t, \mathbf{o}_{t+1}, P_{\text{plan}}(\cdot|\mathbf{z}_t), r_t, \texttt{done}, d_t^{\text{gt}}).$$

Mini-batches are sampled from the buffer for gradient updates.

7.2 Losses

Our optimization includes:

- **Reconstruction and auxiliary prediction:** decoder predicts vision mean/log-variance and hunger; additionally, a predator-distance head predicts d^{gt} from $\mathbf{z}$.
- **Transition learning:** match posterior latents inferred from next observation to the transition prior (Gaussian KL).
- **Habitual policy learning:** minimize $\text{KL}(Q_\psi(a|\mathbf{z}) \,\|\, P_{\text{plan}}(a|\mathbf{z}))$ to distill the planner into the habit network.

Precision/attention scheduling. Following [1], we maintain a scalar precision ω as a logistic function of the policy KL divergence. This modulates KL terms between posterior and prior latents (a top-down attention analogue).

8 Implementation Details

8.1 Neural Network Design

The encoder is a Conv1D stack over the RGB strip reshaped to $(B, 3, 160)$, followed by MLP heads producing $(\boldsymbol{\mu}, \log \boldsymbol{\sigma}^2)$. The decoder is a transposed-convolution stack producing vision mean/log-variance and an auxiliary hunger head. The transition model is an MLP with dropout; the habitual network is a lightweight MLP producing softmax logits over 3 actions.

8.2 Hyperparameters

We use a latent dimension $D = 10$, MCTS repeats R (e.g., 15), rollout depth (e.g., 3), and MC-dropout samples for parameter epistemic value. A replay buffer and multiple update steps per episode provide stable training.

9 Discussion

Our zebrafish implementation demonstrates how deep active inference with MC sampling and MCTS can be adapted to a continuous 2D embodied setting with partial observability. The explicit decomposition of EFE into preference, ambiguity, and epistemic terms supports both goal-directed and exploratory behavior, while the habitual network amortizes planning and can short-circuit MCTS when confident. A distinctive addition in this environment is the supervised predator-distance belief head, which provides an interpretable latent belief aligned with ecological risk.

10 Conclusion

We described a DAIMC-style deep active inference agent for zebrafish-inspired foraging. The system combines a learned latent world model, EFE-based valuation with epistemic terms, and MCTS planning distilled into a habitual policy network. This setup provides a biologically motivated framework for studying active perception and action in continuous environments.

References

[1] Z. Fountas, N. Sajid, P. A. M. Mediano, and K. Friston. Deep active inference agents using Monte-Carlo methods. *NeurIPS*, 2020.