GRADUATE SCHOOL OF COMPUTATIONAL NEUROSCIENCE
UNIVERSITY OF TÜBINGEN

Stress-testing rollout-based planning and benchmarking continuous-time foraging with recurrent neural network agents

Laboratory Report

Presented by

Mohammed Abbas Ansari

The study was supervised by

Prof. Dr. Peter Dayan

Dr. Roxana Zeraati

Max Planck Institute for Biological Cybernetics
Department of Computational Neuroscience

Duration of the lab rotation: 2.5 months

Deadline for submission: 03.01.2026

ABSTRACT

Foraging and planning share a core challenge: making sequential choices when time is the limiting resource, spent acting, travelling, or thinking ahead. We asked when recurrent neural-network (RNN) agents should benefit from explicit internal planning and whether they can learn to optimise the trade-off between the time costs of actions. We stress-tested planning in a controlled maze task, in which an agent can spend time running internal simulations ("rollouts") using a learned model of the environment. On familiar mazes, planning did not substantially outperform a non-planning RNN, implying recurrent agents can amortise effective strategies. Under larger changes to maze structure, planning helped only when the new mazes were similar to training and became harmful under strong novelty. This motivates a staged testbed to study planning for foraging tasks. We established a continuous-time, multi-patch foraging task with depletion and recovery, and computed optimal reward-rate ceilings for benchmarking. In a fully observable setting, a model-free recurrent agent learned efficient reward-rate control, approaching the computed optimum in a restricted regime and outperforming simple threshold heuristics. These benchmarks provide a starting point for an investigation of the role of planning during foraging as observability is progressively reduced to more ecological settings.

1 INTRODUCTION

Foraging involves sequential decisions about whether to continue exploiting a local option or transition to search for alternatives, subject to constraints imposed by action durations, travel times, and diminishing returns. Classical optimal foraging theory formalises these decisions as rate maximisation problems in which the relevant currency is long-run energetic gain per unit time, with policy structure determined by environmental statistics rather than local cues alone (Stephens, 1988). In patchy environments with diminishing within-patch returns and non-rewarding travel, the Marginal Value Theorem (MVT) specifies an optimal leaving rule: depart when the instantaneous reward rate of the patch falls below the global average reward rate (Charnov, 1976).

Decision neuroscience has used foraging as a framework for studying control when choice is sequential and time-costly rather than simultaneous and costless (Mobbs et al., 2018). In this view, the central computational variable is the opportunity cost of time, operationalised by a background reward rate against which foreground engagement is evaluated, yielding "stay/leave" and "accept/reject" policies that differ structurally from standard economic choice models. This perspective motivates tasks and models that make search costs explicit and treat state transitions as part of the decision problem, with proposed dissociations between neural systems supporting background monitoring and those supporting evaluation of current options (Hall-McMaster and Luyckx, 2019).

Structured foraging settings introduce additional requirements beyond the MVT's classical assumptions, including revisitation, heterogeneous patch dynamics, and controllability over the next encounter. When agents can choose where to go next, leaving can depend on the value of specific alternatives rather than on a single global background rate, changing normative benchmarks and behavioural signatures (Hall-McMaster et al., 2021). In environments with slow replenishment, optimal policies can depend on tracking the joint state of multiple patches and adapting leave thresholds to the current configuration of alternatives, leading to deviations from fixed-threshold heuristics (Zeraati, 2025). Related work links foraging decisions to structure learning and planning, suggesting that deviations from simple thresholds can reflect inference over environmental structure and uncertainty (Harhen and Bornstein, 2023).

A mechanistic account of these computations requires models that can be trained on sequential interaction and analyzed at the level of internal dynamics. Meta-reinforcement learning (meta-RL) provides one route by training recurrent neural networks (RNN) across a distribution of tasks such that within-episode adaptation is implemented in activity dynamics, while parameters are updated across episodes by a slower learning process (Duan et al., 2016). This "learning-to-learn" template has been proposed as a computational account of fast learning implemented in prefrontal recurrent dynamics, trained by slower dopaminergic reinforcement signals (Wang et al., 2017; 2018). Meta-trained networks have also been argued to support rational process models that connect normative objectives to constrained computation, with empirical evidence that meta-learned agents can ap-

proximate Bayes-optimal behaviour under some task distributions (Binz et al., 2023; Mikulik et al., 2020).

Planning adds a distinct computational element: prospective evaluation of action sequences using internal generative structure. Neural recordings and human neuroimaging indicate fast sequential state representations in the absence of external stimuli, consistent with internally generated trajectories that could support planning or credit assignment (Kurth-Nelson et al., 2016). behavioural and computational analyses distinguish online replay, which can guide imminent choices, from offline replay, which can propagate value information and support amortisation into cached policies, providing a framework for when deliberative computation should be deployed (Eldar et al., 2020). Jensen et al. (2024) propose a recurrent meta-reinforcement learning architecture for maze navigation tasks in which an explicit "planning action" triggers internal rollouts using a learned world model at a time cost, linking prospective simulation to decision-time behaviour and replay signatures (Jensen et al., 2024).

Both foraging and explicit planning with computation costs can be cast as time-aware decision problems in which actions consume variable amounts of physical time, making semi-Markov decision processes (SMDPs) a unifying formalism (Bradtke and Duff, 1994). SMDPs generalise Markov decision processes by associating each transition with a duration, supporting learning and planning when temporally extended actions or variable execution times are integral to the task (Sutton et al., 1999). Continuous-time reinforcement learning formulations and average-reward criteria provide normative and algorithmic foundations for problems where continuing performance is evaluated by long-run reward rate rather than discounted episodic return (Bradtke and Duff, 1994; Mahadevan, 1996). In practice, actor–critic deep-RL methods (Mnih et al., 2016) provide scalable optimisation tools for training recurrent policies in high-dimensional settings, and we show that they can be adapted to time-aware return definitions when durations vary across actions.

This study uses the SMDP framing to connect two distinct task domains under a shared time-aware interface and training protocol. First, we use a discrete spatial maze navigation benchmark to replicate and analyze a rollout-based planning agent, treating the maze task as a controlled setting in which internal rollouts have an explicit time cost and can be compared against recurrent baselines trained without access to a planning action (Jensen et al., 2024). Second, we introduce a continuous-time free-choice multi-patch foraging task motivated by structured foraging settings with control over patch encounters, with explicit travel times and replenishment dynamics that support normative benchmarking under an average-reward criterion (Hall-McMaster et al., 2021; Zeraati, 2025). For the foraging task, we focus on specifying the environment dynamics, constructing normative baselines via average-reward policy iteration, and evaluating learning in a fully observed validation regime over a restricted subset of task parameters, without treating observed performance as representative of the full parameter space

In the maze benchmark, recurrent agents trained without a planning action attain performance comparable to planning-enabled agents on the training distribution, indicating that recurrent dynamics can amortise effective policies over the sampled task family. Under systematic distribution shift across maze families, planning yields an advantage in a near-out-of-distribution regime. Still, it degrades on more distant families, accompanied by reduced world-model prediction accuracy and reduced stability of goal-location representations over post-discovery trajectories. In the foraging task, the continuous-time SMDP formulation enables instance-specific optimal reward-rate ceilings via average-reward policy iteration, allowing empirical policies to be evaluated by efficiency relative to an optimal benchmark rather than episodic return. In a fully observed validation regime and a restricted parameter setting, task-trained recurrent agents achieved approximately 90% efficiency relative to the computed optimum, which we interpret as partial evidence that the time-aware training setup can recover high reward-rate control on the tested structured regime. Together, these results delimit conditions under which rollout-based planning is beneficial versus brittle in the maze benchmark and establish a continuous-time structured foraging testbed with normative baselines for future testing of how recurrent agents can allocate time between exploitation, travel, and internal planning computation as observability and controllability of tasks are varied.

2 MATERIAL & METHODS

2.1 OVERVIEW

In this project, we studied when recurrent meta-reinforcement learning (meta-RL) agents benefit from explicit prospective planning in maze navigation tasks and whether the same architectural template can support strategic decision-making in continuous-time foraging tasks. Our methodology follows a common protocol across two environments: each training episode samples a new task instance ω from a task distribution $p(\omega)$, the agent's recurrent state is reset, and a gated recurrent unit (GRU)-based policy/value network is trained on-policy to maximise task-dependent returns from its within-episode interaction trajectory.

We first replicated the planning-enabled maze navigation agent of Jensen et al. (2024) in Python and used this benchmark to probe generalisation systematically: we parameterised maze families by a shortcut count n_{sc} controlling navigation difficulty, trained planning and non-planning agents on a fixed $n_{\mathrm{sc}}^{\mathrm{train}}$, and evaluated transfer across held-out families $n_{\mathrm{sc}} \in \{0, \ldots, 12\}$, quantifying both behavioural performance and internal model stability.

We then implemented a continuous-time free-choice P-patch foraging environment (instantiated with $P = 3$) with stochastic replenishment dynamics and action-dependent durations, and trained a non-planning recurrent agent in a fully observed validation regime to assess whether meta-RL can recover near-optimal reward-rate control when the full Markov state and task parameters are provided as inputs. For foraging, we benchmarked learned policies against normative average-reward optima computed via policy iteration and against threshold-based heuristics, and assessed robustness via controlled parameter sweeps.

2.2 PROBLEM FORMULATION

Time-aware interaction formalism Both the maze navigation with planning and foraging tasks can be unified and expressed as *time-aware* decision processes in which actions consume variable amounts of physical time. We model each environment as a semi-Markov decision process (SMDP): at decision step t, the agent observes an environment-specific observation vector o_t (defined in Section 2.4), updates an internal recurrent state h_t, and selects an action a_t from the task-dependent action set. Executing a_t induces a transition to a new state s_{t+1}, yields an immediate reward r_t, and advances physical time by an action-dependent duration $\Delta\tau_t > 0$. We write this generically as

$$(s_{t+1}, r_t, \Delta\tau_t) \sim P_\omega(\cdot \mid s_t, a_t), \tag{1}$$

where ω denotes the sampled task instance and P_ω its transition kernel. The accumulated time after t decisions is $\tau_t = \sum_{u=0}^{t-1} \Delta\tau_u$. This formulation makes explicit a key distinction between the two environments: in the maze task, where planning and physical actions take different amounts of time, performance is constrained by a fixed time budget T_{time} (50s), so the relevant objective is the total number of goal hits within this budget; in the foraging task, decisions unfold in continuous time with heterogeneous action durations, so performance is evaluated in terms of reward *rate* (rewards per unit time). For training stability in foraging, we use a continuous-time discounted surrogate in which stepwise discounting depends on elapsed time, with per-step discount factor $\gamma_t = \exp(-\beta \cdot \Delta\tau_t)$ for a discount rate $\beta > 0$; the resulting return targets are defined explicitly in Section 2.8. Throughout, we reserve the term *planning rollout* exclusively for prospective internal look-ahead computations triggered by the maze agent's planning action, and use *interaction trajectory* to refer to the externally generated agent–environment sequence used for learning and evaluation.

Meta-RL protocol over a task distribution In both domains we train agents in a *meta-RL* regime, where the objective is not to solve a single fixed environment but to learn a recurrent policy that adapts online to new task instances sampled from a distribution. Concretely, each training episode begins by sampling a task instance $\omega \sim p(\omega)$ (a maze wall layout configuration and goal location in navigation; a set of patch parameters in foraging), resetting the environment to its episode-specific initial state, and resetting the agent's recurrent state h_0 (and any auxiliary inputs) to a fixed initialisation. The agent then generates an interaction trajectory $\{(o_t, a_t, r_t, \Delta\tau_t)\}_{t=0}^{T-1}$ by selecting actions on-policy from $\pi_\theta(\cdot \mid h_t)$ and updating its recurrent state (of a gated recurrent unit (GRU)) as $h_t = \mathrm{GRU}_\theta(h_{t-1}, x_t)$ with inputs x_t that include the observation and recent interaction history

(defined in Section 2.4). Importantly, within-episode adaptation is implemented purely through the evolution of the recurrent state: the network parameters θ are held fixed during an episode, and learning across episodes occurs only via gradient updates using data collected from many independently sampled ω. The meta-training objective therefore optimises expected performance under the task distribution,

$$\max_{\theta} \mathbb{E}_{\omega \sim p(\omega)} \mathbb{E}_{\pi_\theta(\cdot|\omega)} \big[G(\omega) \big], \tag{2}$$

where $G(\omega)$ denotes a task-dependent return functional of the interaction trajectory. In the maze task $G(\omega)$ corresponds to the undiscounted cumulative reward accrued within a fixed time budget, whereas in the foraging task we train using a continuous-time discounted surrogate while evaluating performance primarily in terms of reward rate relative to the optimal average-reward policy (defined in Section 2.7). For evaluation, we follow the same episode protocol but with greedy action selection, and assess generalisation by sampling held-out task instances (and, in the maze experiments, held-out *families* indexed by the shortcut parameter n_{sc}) that differ systematically from the training distribution.

2.3 Task Definitions

2.3.1 Maze Navigation Task

The maze navigation environment is a discrete spatial task defined on a $L \times L$ toroidal grid (here $L = 4$, $S = L^2 = 16$ cells). A single task instance is denoted $\omega = (W, g)$, where W specifies the wall layout (i.e., which edges between adjacent cells are blocked) and $g \in \{0, \dots, S-1\}$ is the goal cell. Within an episode, the wall layout W and goal g are fixed, but g is initially unknown to the agent and must be inferred from reward feedback. At decision step t, the agent occupies a cell $y_t \in \{0, \dots, S-1\}$ and selects an action from $\mathcal{A}_{\mathrm{maze}} = \{a_\uparrow, a_\downarrow, a_\leftarrow, a_\rightarrow, a_{\mathrm{plan}}\}$; the planning action a_{plan} is only available for planning-enabled agents (Section 2.6), and non-planning agents act only in $\mathcal{A}_{\mathrm{phys}} = \{a_\uparrow, a_\downarrow, a_\leftarrow, a_\rightarrow\}$. Physical actions deterministically attempt to move the agent to the adjacent cell in the chosen direction; if the corresponding edge is blocked by a wall in W, the transition is null and the agent remains in place ($y_{t+1} = y_t$). A unit reward is delivered only upon *entering* the goal cell via a physical transition, i.e. $r_t = 1$ if $a_t \in \mathcal{A}_{\mathrm{phys}}$ and $y_{t+1} = g$, and $r_t = 0$ otherwise. Following reward delivery, the environment teleports the agent in the next decision step to a uniformly sampled non-goal cell, independently of the action at that step, inducing a repeated explore–exploit structure within each episode: the agent must discover the goal location at least once and then repeatedly navigate back to it efficiently from arbitrary start locations. Episodes are constrained by a fixed physical time budget $T_{\mathrm{time}} = 50\,\mathrm{s}$; physical actions consume $\Delta\tau_t = 1\,\mathrm{s}$, while planning actions (when enabled) consume $\Delta\tau_t = 0.3\,\mathrm{s}$ (independent of rollout depth). Task *families* are defined by the distribution over wall layouts W, parameterised by the shortcut count n_{sc} (see below): for each episode, $\omega \sim p_{\mathrm{maze}}(\omega \mid n_{\mathrm{sc}})$ is sampled i.i.d., the agent is initialised at a uniformly random non-goal cell, and performance is measured by the number of goal hits (total reward) accrued within the 50s budget.

Maze procedural generation and shortcut parameter n_{sc}. We generate maze wall layouts W procedurally to obtain a controllable family of task distributions with varying navigation difficulty. Each layout is defined on the $L \times L$ toroidal grid graph whose vertices correspond to cells and whose edges correspond to possible moves between horizontally/vertically adjacent cells. We begin from the fully walled configuration (all edges blocked), then construct a dense maze (a spanning tree over the grid graph) using a randomised depth-first search (DFS). Specifically, we sample a starting cell uniformly at random and perform a DFS traversal over unvisited neighbouring cells; whenever the traversal moves along an edge to a new cell, the corresponding wall is removed, and the neighbour is marked visited. This procedure terminates when all S cells have been visited, yielding a connected acyclic corridor structure in which there exists a unique path between any pair of cells. We denote the resulting spanning-tree layout by W_{tree}.

To modulate maze difficulty and introduce path degeneracy, we then add *shortcuts* by removing additional walls that remain after DFS. Let $\mathcal{E}_{\mathrm{rem}}(W_{\mathrm{tree}})$ denote the set of blocked edges after spanning-tree generation. For a specified shortcut count n_{sc}, we sample n_{sc} distinct edges uniformly without replacement from $\mathcal{E}_{\mathrm{rem}}(W_{\mathrm{tree}})$ and remove the corresponding walls to obtain the final layout W. Thus $n_{\mathrm{sc}} = 0$ yields a perfect maze (tree-structured, maximally constrained), while larger n_{sc} yields

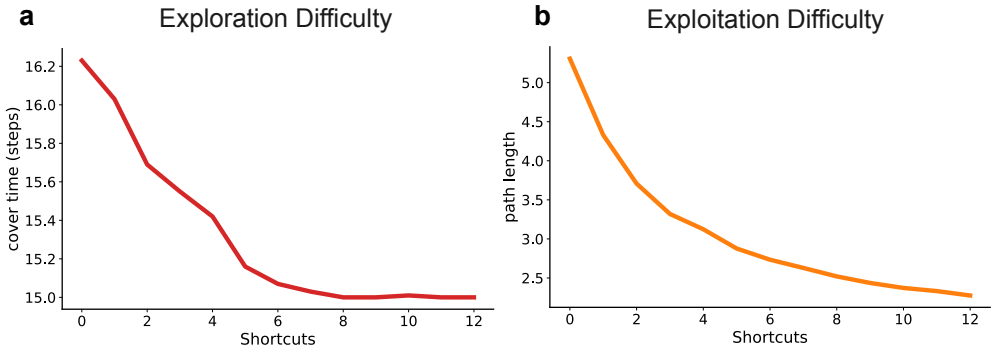

Figure 1: **Varying maze task difficulty across shortcuts (a)** Maze cover time (in terms of physical steps) i.e. the length of shortest walk that visits all locations at least once averaged over hundred sampled mazes for each shortcut configuration, indicating that the optimal time to find the reward in maze (exploration) decreases with shortcuts, thus decreasing difficulty. **(b)** Average minimum path length between any two locations in a maze averaged over hundred sampled mazes for each shortcut configuration, indicating that its quicker to go between locations, thus easier for an agent to navigate back to the reward (exploitation) after each teleportation step.

increasingly loopy mazes with multiple paths between locations and typically shorter geodesic distances, thus easier to navigate in (Figure 1). In our experiments, we consider $n_{\mathrm{sc}} \in \{0, \ldots, 12\}$, which spans the range from dense, corridor-like mazes to comparatively open layouts on the 4×4 grid.

This construction induces a family of maze task distributions $p_{\mathrm{maze}}(\omega \mid n_{\mathrm{sc}})$ over task instances $\omega = (W, g)$. Conditional on n_{sc}, we sample W via the DFS and shortcut procedure described above, and sample the goal location g uniformly over the S cells. For training and generalisation analyses, we define a *training family* by fixing $n_{\mathrm{sc}}^{\mathrm{train}}$ and sampling ω i.i.d. from $p_{\mathrm{maze}}(\cdot \mid n_{\mathrm{sc}}^{\mathrm{train}})$, and we evaluate transfer by sweeping test families ($n_{\mathrm{sc}} \in \{0, \ldots, 12\}$) while holding all other aspects of the task constant.

2.3.2 Continuous-Time Foraging Task

Unlike the maze navigation task, which primarily probes *spatial* control and prospective look-ahead under a fixed time budget, the foraging task is a *temporal* control problem in which the central challenge is deciding **when to exploit a patch and when to leave**, given that both exploitation and travel consume time and patch quality changes while the agent is away. We consider a continuous-time, free-choice P-patch environment (instantiated with $P = 3$) in which, at any moment, the agent occupies one patch and can either (i) **exploit** the current patch to sample a stochastic reward or (ii) **travel** to another patch of its choosing, incurring an action-dependent travel time. Patches **deplete** when rewards are harvested and **replenish** stochastically over time when left alone (potentially after a patch-specific delay). Because time is explicit and action durations are heterogeneous, performance is characterised by **reward rate** (reward per unit time), and optimal behaviour must trade off immediate exploitation against the opportunity cost of travelling to a richer patch elsewhere.

State and location. At decision step t, the agent occupies a patch indexed by $\ell_t \in \{0, \ldots, P-1\}$. Each patch i has an internal discrete depletion index $z_{i,t} \in \{0, \ldots, N_i - 1\}$, where $z_{i,t} = 0$ denotes a fully replenished patch and larger values denote increasingly depleted states. The full environment state is

$$s_t = (\ell_t, z_{0,t}, \ldots, z_{P-1,t}). \tag{3}$$

We initialise depletion states at episode start by sampling $z_{i,0}$ uniformly over $\{0, \ldots, N_i - 1\}$, rather than starting all patches fully replenished, to reduce initial transient rich "harvesting" phase and to better approximate a steady-state environment under agent's interaction.

Patch reward probabilities (reward ladder). Patch i is parameterised by a maximum reward probability $\lambda_{\max,i} \in (0, 1]$ and a depletion ratio $\rho_i \in (0, 1)$. The probability of receiving reward

upon exploitation in depletion state z follows a geometric ladder:

$$\lambda_i(z) = \lambda_{\max,i}\rho_i^z. \tag{4}$$

Thus $\lambda_i(0) = \lambda_{\max,i}$, and reward probability decays monotonically as z increases. The number of discrete depletion states N_i is chosen per patch such that the least-rewarding state has probability approximately bounded below by a target $p_{\min}^{\text{target}}$ (in our experiments $p_{\min}^{\text{target}} = 0.2$).

Actions and action durations. The action set at patch ℓ_t is

$$\mathcal{A}_{\text{for}}(\ell_t) = \{a_{\text{ex}}\} \cup \{a_{\to j} : j \in \mathcal{N}(\ell_t)\}, \tag{5}$$

where a_{ex} denotes exploitation and $a_{\to j}$ denotes travel to destination patch j, and $\mathcal{N}(\ell_t)$ denotes the set of reachable patches from patch ℓ_t. In our free-choice ($P = 3$) instantiation used here, $\mathcal{N}(i) = \{0, 1, 2\} \setminus \{i\}$, i.e. it is fully connected, but the implementation allows arbitrary directed connectivity.

Each decision consumes physical time. Exploitation has fixed duration $D_{\text{ex}} > 0$. Travel from patch i to patch j consumes duration $D_{ij} > 0$, collected into a matrix $D \in \mathbb{R}_{\geq 0}^{P\times P}$ with $D_{ii} = 0$ (symmetrically instantiated in our case). The elapsed time per step is therefore

$$\Delta\tau_t = \begin{cases} D_{\text{ex}}, & a_t = a_{\text{ex}}, \\ D_{\ell_t j}, & a_t = a_{\to j}. \end{cases} \tag{6}$$

Rewards and depletion dynamics. On exploitation ($a_t = a_{\text{ex}}$), the agent receives a stochastic reward

$$r_t \sim \text{Bernoulli}\big(\lambda_{\ell_t}(z_{\ell_t,t})\big). \tag{7}$$

Depletion occurs only upon successful harvest: if $r_t = 1$, the depletion index of the exploited patch increases by an integer jump size $d_{\ell_t} \in \{1, 2, \dots\}$,

$$z_{\ell_t,t+1} = \min\{z_{\ell_t,t} + d_{\ell_t}, N_{\ell_t} - 1\}, \tag{8}$$

and if $r_t = 0$, the exploited patch's depletion state is unchanged. The jump size d_i is patch-specific and sampled per task instance.

Replenishment dynamics (continuous-time recovery with delay). Because actions in the foraging task consume heterogeneous physical durations (exploitation vs. travel), patch recovery is defined in continuous time rather than in a fixed-step discrete simulator. We treat the replenishment of non-resident patches as a continuous-time Markov chain (CTMC) over the discrete depletion state space $z_i \in \{0, \dots, N_i - 1\}$. The dynamics follow a linear pure death process **on the depletion index** (corresponding to patch recovery): recovery events occur with a constant, state-independent rate γ_i and induce a transition $z_i \to z_i - 1$, bounded by an absorbing state at $z_i = 0$ (full replenishment). Because the holding times between recovery events are exponentially distributed ($\tau \sim \text{Exp}(\gamma_i)$), the number of events in any time interval follows a homogeneous Poisson process. This allows us to evolve the environment state via exact sampling rather than iterative approximation. Over an eligible replenishment duration $u_{i,t} \geq 0$, the number of recovery steps $J_{i,t}$ is drawn from the distribution:

$$J_{i,t} \sim \text{Poisson}(\gamma_i u_{i,t}), \qquad \text{with } \gamma_i = \kappa_i/dt, \tag{9}$$

where κ_i is the patch-specific intensity and dt is a base time scale. The state update is the projection of this jump process onto the non-negative integers:

$$z_{i,t+1} = \max\{z_{i,t} - J_{i,t}, 0\}. \tag{10}$$

This formulation ensures that the reward statistics are invariant to the simulation time step, depending only on physical rates and durations. The eligible duration $u_{i,t}$ respects the ecological refractory period δ_i: for a non-resident patch i during a decision step of length $\Delta\tau_t$, $u_{i,t} = \Delta\tau_t$, unless patch i was the one just departed, in which case it accrues eligibility only after the delay, $u_{i,t} = \max(\Delta\tau_t - \delta_i, 0)$.

Task instance parameters. A single foraging task instance is specified by the collection of patch-wise and travel parameters

$$\omega = \{\lambda_{\max,i}, \rho_i, N_i, \kappa_i, d_i, \delta_i\}_{i=0}^{P-1} \cup \{D_{ij}\}_{i,j=0}^{P-1} \cup \{D_{\text{ex}}\}, \tag{11}$$

which are drawn from bounded ranges defining a task distribution $p_{\text{for}}(\omega)$. The reported experiments use $P = 3$ and a fully observed validation regime in which the agent receives the current state variables s_t and static task parameters ω as inputs, allowing us to validate the meta-RL training pipeline on a well-specified continuous-time control problem before introducing partial observability.

Training distribution. For meta-training, each episode samples a new instance $\omega \sim p_{\text{for}}(\omega)$ where parameters are drawn independently to cover a range of foraging dynamics. We sampled maximum reward probabilities $\lambda_{\max,i} \sim \mathcal{U}(0.3, 1.0)$ and fixed the decay ratio $\rho_i = 0.9$. The number of states N_i was derived per patch to ensure the lowest reward probability approximates $p_{\min}^{\text{target}} = 0.2$, with N_i clamped to $[3, 30]$ for numerical stability. Replenishment intensities were sampled as $\kappa_i \sim \mathcal{U}(0.1, 0.3)$, depletion jump sizes as integers $d_i \sim \mathcal{U}\{1, 2, 3\}$, and delays as $\delta_i = k_i^{\text{delay}} dt$ with $k_i^{\text{delay}} \sim \mathcal{U}(1, 5)$. We fixed the base time scale $dt = 3.5\,\text{s}$ and set exploitation duration $D_{\text{ex}} = 2\,dt$. Travel times were parameterised as $D_{ij} = m_{ij} dt$; while the formalism supports arbitrary geometries, our experiments used symmetric travel costs with multipliers $m_{ij} = m_{ji} \sim \mathcal{U}(1, 10)$ for $i \neq j$ (with $D_{ii} = 0$).

Constraint on delays. To ensure that the environment state s_t remains fully Markovian without requiring auxiliary timers for residual refractory periods, we constrain replenishment delays to be resolved within a single travel action. Specifically, if a sampled delay δ_i exceeds the travel time to any neighbour, we clamp it such that $\delta_i \leq \min_{j \neq i} D_{ij}$. This guarantees that by the time the agent completes a transition to a new patch, the previously occupied patch has effectively passed its refractory period and is accumulating replenishment eligibility.

2.4 Agent–Environment Interface

To make the learning problem comparable across domains, we use a common *interface template* in which the recurrent agent receives (i) task-specific observations describing the current environment configuration and state, and (ii) a minimal record of the most recent interaction outcome. At decision step t, the agent constructs an input vector

$$x_t = \big[o_t, \text{onehot}(a_{t-1}), r_{t-1}, \Delta\tau_{t-1}, \varphi_{t-1}\big], \tag{12}$$

updates its recurrent state $h_t = \text{GRU}_\theta(h_{t-1}, x_t)$, and outputs a policy and value estimate from h_t (Section 2.5). Here o_t is the task-specific observation vector, $\Delta\tau_{t-1}$ is the physical time elapsed by the previous action (Section 2.8), and φ_{t-1} is an optional *planning feedback* channel that is nonzero only for maze planning rollouts (Section 2.6), and not included for model-free foraging agents. The previous action and reward are included to support within-episode adaptation in the meta-RL regime, and $\Delta\tau$ makes action-dependent time costs explicit in the agent's inputs.

Maze observations and planning feedback. In the maze task, the observation o_t^{maze} includes (i) the agent's current location $y_t \in \{0, \ldots, 15\}$ encoded as a one-hot vector, (ii) the episode's wall layout W (fixed within an episode) encoded as a binary vector over adjacency edges, and (iii) the current elapsed time τ_t within the 50s episode budget (represented as a scalar normalised by T_{time}). The reward location is not included in input, and it needs to be discovered by the agent during task interaction. For planning-enabled agents, the feedback vector φ_{t-1} summarises the outcome of a prospective planning rollout invoked when $a_{t-1} = a_{\text{plan}}$: it encodes the imagined action sequence (padded/truncated to the maximum rollout depth) together with a binary indicator of whether the imagined trajectory reached the predicted goal. When the previous action is physical ($a_{t-1} \in \mathcal{A}_{\text{phys}}$), we set $\varphi_{t-1} = 0$.

Foraging observations in the full-observability validation regime. In the foraging task, the observation o_t^{for} is designed to validate that the meta-RL pipeline can learn near-optimal control when the underlying Markov decision process is fully specified to the agent. Concretely, o_t^{for} contains: (i) the current patch index $\ell_t \in \{0, \ldots, P-1\}$ encoded one-hot, (ii) the current patch

qualities: the depletion indices $z_{i,t}$ and, equivalently, the implied reward probabilities $\lambda_i(z_{i,t})$ for all patches i, and (iii) the static task parameters ω governing dynamics and time costs: per-patch $\{\lambda_{\max,i}, \rho_i, \kappa_i, d_i, \delta_i\}$, the exploitation duration D_{ex}, and the travel-time matrix D (its non-redundant entries).

We provide the elapsed time since the previous decision $\Delta\tau_{t-1}$ as a separate scalar feature in x_t, rather than normalised episode time, since foraging is treated as a continuing control problem.

2.5 META-RL AGENT ARCHITECTURE

All agents share the same recurrent meta-RL template: a gated recurrent unit (GRU) backbone that integrates the interaction history, coupled to lightweight readout heads for control (policy) and prediction (value; plus a world model for the maze planning agent).

At decision step t, the agent updates a hidden state $h_t \in \mathbb{R}^H$ from the previous hidden state and the current input vector x_t (Section 2.8):

$$h_t = \text{GRU}_\theta(h_{t-1}, x_t). \tag{13}$$

The hidden state is the agent's sufficient statistic for within-episode adaptation under the task distribution, i.e., π_θ is history-dependent only through h_t.

Policy and Value Heads. We use a single affine readout from h_t to produce both policy logits and a scalar value estimate. Concretely:

$$(\eta_t, V_t) = W_{\text{pv}} h_t + b_{\text{pv}}, \tag{14}$$

where $\eta_t \in \mathbb{R}^{|\mathcal{A}|}$ are the pre-softmax policy logits over the task's action set $\mathcal{A}$, and $V_t \in \mathbb{R}$ is the critic output. The policy is given by:

$$\pi_\theta(a \mid h_t) = \text{softmax}(\eta_t)_a. \tag{15}$$

Action sets are task-specific: $|\mathcal{A}_{\text{maze}}| = 5$ for planning-enabled maze agents (including a_{plan}), $|\mathcal{A}_{\text{maze}}| = 4$ for non-planning maze baselines, and $|\mathcal{A}_{\text{for}}(\ell_t)| = 1 + |\mathcal{N}(\ell_t)| = 3$ (exploit plus travel-to-destination actions). During training, actions are sampled from $\pi_\theta(\cdot \mid h_t)$; during evaluation, we act greedily via $\arg\max_a \eta_{t,a}$ (Section 2.8).

Maze-only predictive head: learned world model for planning. Planning-enabled maze agents include an additional predictive module that serves as an internal world model during prospective rollouts (Section 2.6) and supports auxiliary predictive losses during training. To condition predictions on the chosen action, we form the concatenated feature:

$$u_t = [h_t;\ \text{onehot}(a_t)], \tag{16}$$

and pass it through a two-output multilayer perceptron (MLP) $g_\theta(\cdot)$ to produce categorical logits over grid locations:

$$(\xi_t^{\text{next}}, \xi_t^{\text{goal}}) = g_\theta(u_t), \qquad \xi_t^{\text{next}}, \xi_t^{\text{goal}} \in \mathbb{R}^S, \tag{17}$$

with $S = L^2 = 16$ for a 4×4 maze. These logits define two predictive distributions:

$$\hat{p}_t^{\text{next}} = \text{softmax}(\xi_t^{\text{next}}), \qquad \hat{p}_t^{\text{goal}} = \text{softmax}(\xi_t^{\text{goal}}), \tag{18}$$

where $\hat{p}_t^{\text{next}}$ predicts the next location y_{t+1} given the current hidden state and action, and $\hat{p}_t^{\text{goal}}$ predicts the (latent) goal location g. These predictions are used (i) to drive internal rollouts when $a_t = a_{\text{plan}}$, and (ii) to define the world-model accuracy metrics reported in Figures 2 and 4 (Section 2.9), via argmax classification against the true next location and true goal respectively.

For non-planning maze baselines, we disable this module by setting the predictive-loss weight to zero and removing a_{plan} from the action set (policy renormalisation over physical actions only).

Foraging configuration: model-free control in a fully observed MDP. For the foraging experiments reported here, we use the same GRU backbone and shared policy/value affine head (above), but **no** additional world model head and **no** planning mechanism. The observation/input vector includes the full Markov state and static task parameters (Section 2.3.2), making this a deliberately "fully specified" regime: the agent is not required to infer hidden variables, but must still learn a control strategy that generalises across the distribution of task instances ω. Normative optimal policies and reward-rate ceilings are computed separately via average-reward policy iteration (Section 2.7), and learning success is quantified by efficiency relative to this optimum (Figure 5).

2.6 PLANNING ROLLOUTS FOR MAZE NAVIGATION

Planning-enabled maze agents augment physical interaction with **prospective rollouts**: brief internal simulations that use the learned world model to evaluate multi-step action sequences without changing the external environment state. We reserve the term *rollout* exclusively for this look-ahead planning mechanism (not for ordinary agent–environment interaction trajectories).

Triggering a rollout and time cost. At decision step t, the policy may select the dedicated planning action $a_{\text{plan}} \in \mathcal{A}_{\text{maze}}$. Executing $a_t = a_{\text{plan}}$ freezes the true environment state (walls W, true agent location y_t, and goal g) and initiates an internal simulation of up to L_{plan} imagined physical steps (here $L_{\text{plan}} = 8$). The planning action consumes a fixed physical duration $\Delta\tau_t = 0.3\,\text{s}$, independent of how many imagined steps are simulated; by contrast, physical actions consume $\Delta\tau_t = 1\,\text{s}$ (Section 2.3.1). No external reward is delivered and no external transition occurs at a planning action.

Rollout dynamics in imagined state space. A rollout starts from the agent's current recurrent state h_t and the current real location y_t. The target for planning is the agent's current internal estimate of the goal location, given by the mode of the goal-prediction head:

$$\hat{g}_t = \arg\max_j \, \xi^{\text{goal}}_{t,j}, \tag{19}$$

where ξ^{goal}_t are the goal logits produced by the world model head (Section 2.5). The rollout then iterates imagined steps $\ell = 1, \ldots, L_{\text{plan}}$ in an open-loop manner:

1. **Imagined action selection:** Sample an imagined physical action $\tilde{a}_\ell \in \mathcal{A}_{\text{phys}}$ from the current policy evaluated on the current imagined recurrent state $\tilde{h}_{\ell-1}$ (initially $\tilde{h}_0 = h_t$).
2. **Imagined transition:** Predict the next imagined location distribution using the next-state head conditioned on $(\tilde{h}_{\ell-1}, \tilde{a}_\ell)$, i.e. $\hat{p}^{\text{next}}(\cdot \mid \tilde{h}_{\ell-1}, \tilde{a}_\ell)$, and take its mode as the imagined next location $\tilde{y}_\ell$. Walls W are treated as known and are provided as input during rollouts; thus, if an imagined physical action would cross a wall, the imagined transition is clamped to $\tilde{y}_\ell = \tilde{y}_{\ell-1}$, matching the external dynamics.
3. **Imagined recurrent update:** Advance the imagined recurrent state by feeding the rollout step back through the GRU with the same interface template as real interaction with elapsed time computed as considering the imagined step as physical step, but with rewards set to zero.
4. **Termination:** If $\tilde{y}_\ell = \hat{g}_t$, terminate early and mark the rollout as "goal reached".

This process yields an imagined action sequence $(\tilde{a}_1, \ldots, \tilde{a}_m)$ (with $m \leq L_{\text{plan}}$) and a binary success indicator $q_t \in \{0, 1\}$ denoting whether the imagined trajectory reached the imagined goal.

Rollout feedback encoding The rollout outcome is summarised into a fixed-dimensional feedback vector φ_t that is appended to the next real input x_{t+1} (Section 2.5). Operationally, φ_t concatenates (i) a one-hot encoding of the imagined action sequence, padded with zeros to length L_{plan}, and (ii) the success flag q_t. This provides the GRU with a compact representation of the plan it "considered" and whether it found a route to its internally predicted goal, enabling the recurrent dynamics to condition subsequent physical decisions on the content of imagination.

Gradient blocking through rollouts Rollouts are treated as an internal computation that produces an auxiliary input φ_t but is not itself part of the differentiable computation graph. Concretely, we do **not** backpropagate through the imagined rollout trajectory: gradients from the policy/value losses flow through the GRU and the policy/value head along the **real** interaction trajectory, and predictive losses train the world model head on real transitions, but the rollout generation is a stop-gradient operation. This matches the original Jensen et al. (2024)'s implementation and avoids high-variance gradients through long hypothetical sequences while still allowing rollouts to influence behaviour via the feedback channel.

2.7 NORMATIVE BASELINES FOR FORAGING

The foraging results are reported in terms of **efficiency**, i.e., the ratio between the empirical reward rate achieved by the learned RNN policy and a **theoretical optimal reward rate** for the same environment. We therefore compute (i) an optimal policy for the fully observed foraging SMDP under the **average-reward** criterion, and (ii) simple **threshold heuristics** used as interpretable behavioural baselines in the single-environment analysis (Figure 7).

2.7.1 OPTIMAL POLICY VIA AVERAGE-REWARD POLICY ITERATION

To determine the theoretical ceiling for performance, we model the foraging task as a Semi-Markov Decision Process (SMDP) and solve for the policy that maximises the long-run average reward rate:

$$\rho^\star = \max_\pi \lim_{T\to\infty} \frac{1}{T}\,\mathbb{E}_\pi\left[\sum_{t=0}^{\infty} r_t\,\mathbf{1}\{\tau_t \le T\}\right], \qquad \tau_t := \sum_{k=0}^{t-1}\Delta\tau_k. \tag{20}$$

where r_t is the reward obtained on the t^{th} decision and $\Delta\tau_t$ is the elapsed time of that decision.

State space and matrix enumeration. We enumerate the discrete state space $s = (\ell, z_0, \ldots, z_{P-1})$, where ℓ is the agent location and z_i are patch depletion levels, and map each multi-index state to a single flattened state index. For a given environment, we construct the full SMDP tensors: transition probabilities $P(s' \mid s, a)$, transition-conditioned rewards $R(s, a, s')$, and transition-conditioned durations $T(s, a, s')$. These define the expected immediate reward and elapsed time under action a as

$$\bar{r}(s,a) = \sum_{s'} P(s' \mid s,a)\,R(s,a,s'), \qquad \bar{\tau}(s,a) = \sum_{s'} P(s' \mid s,a)\,T(s,a,s'). \tag{21}$$

Policy iteration. We solve for the optimal policy $\pi^\star$ using average-reward policy iteration, which alternates between two steps until convergence:

1. **Policy evaluation:** For a fixed policy π, we analytically solve the average-reward Bellman equation to find the scalar gain ρ_π and the relative value vector v_π:

$$v_\pi(s) = \sum_a \pi(a \mid s)\left[\bar{r}(s,a) - \rho_\pi\,\bar{\tau}(s,a) + \sum_{s'} P(s' \mid s,a)\,v_\pi(s')\right] \tag{22}$$

   subject to an anchor constraint ($v_\pi(s_{\text{ref}}) = 0$). This provides the exact long-run reward rate ρ_π for the current policy.
2. **Policy improvement:** We update the policy to be greedy with respect to the current values:

$$\pi_{\text{new}}(s) = \arg\max_a \left[\bar{r}(s,a) - \rho_\pi\bar{\tau}(s,a) + \sum_{s'} P(s' \mid s,a)v_\pi(s')\right]. \tag{23}$$

This procedure yields the optimal reward rate $\rho^\star$ used as the denominator for efficiency calculations.

2.7.2 THRESHOLD HEURISTIC BASELINES

In addition to the optimal average-reward solution, we evaluate simple threshold rules that operationalize "leave when the current patch is no longer worth exploiting," while deliberately **not** solving the full directed multi-patch control problem.

Let the instantaneous expected exploit gain at patch ℓ be g_ℓ.

- **Single-threshold policy:** Fix a scalar threshold θ. At state $s = (\ell, z)$:
  - If $g_\ell \ge \theta$, choose a_{ex}.
  - Otherwise, leave by selecting a destination $j \in \mathcal{N}(\ell)$ uniformly at random.
- **Patch-specific policy:** Fix separate thresholds $\theta_0, \ldots, \theta_{P-1}$ for each patch.
  - If $g_\ell \ge \theta_\ell$, choose a_{ex}.
  - Otherwise leave, again choosing the destination uniformly among alternatives.

Relation to the Marginal Value Theorem (MVT). Classically, the Marginal Value Theorem predicts that the optimal strategy in simple foraging settings is a single-threshold policy where the leaving threshold is self-consistent, i.e., equal to the global long-run reward rate ($\theta = \rho$). While this provides a theoretical reference, such a fixed point is not guaranteed to exist or be unique in our setting due to the discreteness of state transitions and the specific replenishment dynamics (e.g., if the instantaneous rate at a fully depleted patch remains higher than the global average due to rapid replenishment or high travel costs).

Optimisation via analytical evaluation. Consequently, rather than solving for an MVT fixed point, we find the best possible single and three threshold policies by treating the threshold(s) as free parameters. We perform a grid search over candidate threshold values. For each candidate configuration, we define the induced stationary policy π_{thresh} and compute its exact reward rate $\rho_{\pi_{\mathrm{thresh}}}$ analytically using the policy evaluation step described above. We select the thresholds that maximise this analytically derived rate. This method is robust to cases where strictly self-consistent MVT solutions are undefined, while guaranteed to find them if they are indeed optimal.

2.8 Training Protocols

This section specifies how trajectories were generated, how gradients were computed, and how task-dependent return targets were formed for maze navigation versus continuous-time foraging. Across both tasks, agents were trained with an on-policy advantage actor–critic objective using a shared GRU backbone; the key differences are (i) episode trajectory (full-episode vs. truncated backpropagation through time (BPTT)) and (ii) the return definition (finite-horizon undiscounted sum vs. time-aware discounted bootstrapping as a surrogate for rate maximisation).

2.8.1 Data collection, trajectory, and batching

Maze navigation. Each training update used a batch of B i.i.d. maze episodes sampled from the training maze-family distribution (Sections 2.3.1). Each episode had a fixed time budget ($T_{\text{time}} = 50\,\text{s}$). Physical actions consumed $\Delta\tau_t = 1.0\,\text{s}$ and planning actions consumed $\Delta\tau_t = 0.3\,\text{s}$; the episode terminated when the remaining time budget was exhausted. Because trajectories are short, gradients were computed through the entire episode trajectory (no truncation).

Foraging. Each training update used a batch of B i.i.d. foraging episodes sampled from the training foraging-task distribution (Sections 2.3.2). Each episode consisted of a long stream of $T_{\text{steps}} = 4000$ decision steps sampled from a new environment instance ω at the start of the episode (Section 2.3.2). The long stream is opted to ensure that agents learn a steady-state policy independent of episode horizon. Gradients were computed with truncated backpropagation through time using windows of length $L_{\text{trajectory}} = 200$ steps. Each optimisation step therefore utilized a batch of sequences $\{(x_t, a_t, r_t, \Delta\tau_t)\}_{t=t_0}^{t_0+L_{\text{trajectory}}-1}$ with hidden states carried between windows but detached for backpropagation at window boundaries.

On-policy sampling. During training, actions are sampled stochastically from the learned policy to generate on-policy trajectories. During evaluation, we use greedy action selection. For "non-planning" maze baselines, we remove the planning action by restricting the policy support to $\mathcal{A}_{\mathrm{phys}}$ and renormalising; the planning feedback channel is set to zero throughout.

2.8.2 Advantage actor–critic with entropy regularisation

Let h_t be the GRU hidden state (Section 2.5) and let the shared affine head produce policy logits $\log \pi_\theta(\cdot \mid h_t)$ and value estimate $V_\theta(h_t)$. The per-step actor–critic losses are:

$$\mathcal{L}_\pi(t) = -\hat{A}_t \log \pi_\theta(a_t \mid h_t), \qquad \mathcal{L}_V(t) = \tfrac{1}{2}\big(V_\theta(h_t) - \hat{G}_t\big)^2, \qquad \mathcal{L}_{\text{ent}}(t) = -H\big(\pi_\theta(\cdot \mid h_t)\big), \tag{24}$$

where $\hat{G}_t$ is the return target and $\hat{A}_t = \hat{G}_t - V_\theta(h_t)$ is the advantage estimate (task-dependent via $\hat{G}_t$, defined below). To encourage exploration and prevent premature policy collapse during on-policy training, we add an entropy regulariser for policy:

$$H(\pi_\theta(\cdot \mid h_t)) \;=\; -\sum_{a \in \mathcal{A}} \pi_\theta(a \mid h_t)\, \log \pi_\theta(a \mid h_t). \tag{25}$$

The total loss summed over training steps is:

$$\mathcal{L} = \sum_t \left(c_\pi \mathcal{L}_\pi(t) + c_V \mathcal{L}_V(t) + c_H \mathcal{L}_{\text{ent}}(t)\right) + c_{\text{wm}} \sum_t \mathcal{L}_{\text{wm}}(t), \tag{26}$$

with fixed weights (c_π, c_V, c_H) and an additional world-model term $\mathcal{L}_{\text{wm}}(t)$ used only in the maze planning agent (Subsection 2.8.5). Optimisation used Adam with task-specific learning rates and batch sizes (Subsection 2.8.6).

2.8.3 MAZE RETURN TARGETS

Maze navigation was treated as a finite-horizon problem with a fixed time budget per episode. Return targets were computed as **undiscounted Monte Carlo sums**:

$$\hat{G}_t = \sum_{k=t}^{T_{\text{end}}} r_k, \tag{27}$$

where $r_k \in \{0, 1\}$ is the reward obtained when entering the goal cell (Section 2.3.1), and T_{end} is the final interaction step before time budget exhaustion. The resulting advantage estimate $\hat{A}_t = \hat{G}_t - V_\theta(h_t)$ induces the objective: within the fixed time budget, the agent should maximise total reward count.

2.8.4 FORAGING RETURN TARGETS

Foraging is formally posed as average reward-rate maximisation in continuous time, but for training stability and compatibility with standard actor–critic updates, we trained the RNN with a **time-aware discounted** objective that approximates rate maximisation. Let $\Delta\tau_k$ denote the physical time elapsed by step k (action duration), and let $\beta > 0$ be the continuous-time discount rate (Hz). Define the per-step discount factor as $\gamma_k = \exp(-\beta \cdot \Delta\tau_k)$. We used bootstrapped return targets within each truncated BPTT window. For a window $t \in \{t_0, \ldots, t_0 + L_{\text{trajectory}} - 1\}$, let $T_{\text{end}} = t_0 + L_{\text{trajectory}}$ and define $n_t = T_{\text{end}} - t$ (the remaining steps to the window end). The target at decision step t is

$$\hat{G}_t = \sum_{j=0}^{n_t-1} \left(\prod_{m=0}^{j-1} \gamma_{t+m}\right) r_{t+j} + \left(\prod_{m=0}^{n_t-1} \gamma_{t+m}\right) V_\theta(h_{T_{\text{end}}}), \tag{28}$$

i.e., rewards are accumulated until the end of the trajectory window and bootstrapped from the value at the window boundary. Here $r_{t+j} \in \{0, 1\}$ indicates whether reward was obtained on exploit at that decision, and $\Delta\tau_{t+m}$ depends on whether the agent exploited or travelled (Section 2.3.2). This construction respects physical time: rewards received after longer actions are discounted more strongly through larger $\Delta\tau$.

2.8.5 ADDITIONAL SUPERVISION FOR MAZE PLANNING AGENTS (WORLD-MODEL LOSS)

Planning agents in the maze task learned an internal one-step world model jointly with the policy/value. Given h_t and the selected action a_t, the world-model head predicted (i) the next location and (ii) the goal location (Section 2.5). The auxiliary loss was the sum of cross-entropies:

$$\mathcal{L}_{\text{wm}}(t) = \text{CE}\big(\hat{p}_t^{\text{next}}, \text{onehot}(y_{t+1})\big) + \text{CE}\big(\hat{p}_t^{\text{goal}}, \text{onehot}(g)\big), \tag{29}$$

where y_{t+1} is the true next cell index transitioned to, g is the latent goal cell for the episode, and $(\hat{p}_t^{\text{next}}, \hat{p}_t^{\text{goal}})$ denote the predicted categorical distributions produced by g_θ. This loss was applied at real environment steps; gradients were not propagated through the internal planning rollout itself (Section 2.6).

For non-planning baselines in maze navigation, the planning action was removed by renormalising the policy over physical actions, and the world-model loss weight was set to $c_{\text{wm}} = 0$.

2.8.6 HYPERPARAMETERS

We report the hyperparameters used for each task in the main text:

- **Maze navigation:** 4×4 toroidal mazes, GRU hidden size $N_{\text{hidden}} = 100$, episode length $T = 50\,\text{s}$ planning horizon $L_{\text{plan}} = 8$ (planning agents), batch size 40, learning rate 10^{-3}, loss weights $(c_\pi, c_V, c_H, c_{\text{wm}}) = (1.0, 0.05, 0.05, 0.5)$, no world-model loss ($c_{\text{wm}} = 0$) for non-planner, total training 8×10^6 episodes across 5 random seeds; planning step duration $0.3\,\text{s}$, physical step duration $1.0\,\text{s}$.
- **Foraging:** 3 fully-connected patches, GRU hidden size $N_{\text{hidden}} = 512$, episode length $T = 4000$ steps, truncated BPTT window $L_{\text{trajectory}} = 200$, batch size 128, learning rate 3×10^{-4}, loss weights $(c_\pi, c_V, c_H) = (1.0, 0.5, 0.05)$, no world-model loss ($c_{\text{wm}} = 0$), continuous-time discount rate $\beta = 10^{-3}$ Hz, 10^5 training environments and 200 test environments.

2.9 Evaluation Metrics

2.9.1 Maze navigation analysis

Performance and generalisation metrics. Primary performance is evaluated via Episode Return, defined as the total reward accumulated within the fixed time budget ($T = 50\,\text{s}$). To quantify the specific contribution of the planning mechanism, we compute the Planning Advantage (ΔReward) on matched test sets of 1000 mazes per shortcut family:

$$\Delta\text{Reward} = \mathbb{E}[\text{Reward}_{\text{Plan}}] - \mathbb{E}[\text{Reward}_{\text{NoPlan}}]. \tag{30}$$

We further assess Exploration efficiency by calculating the *Time-to-First-Reward*: the number of physical steps elapsed in an episode before the first reward is encountered, averaged over interaction trajectories. **Planning Fraction** is computed as the ratio of planning actions (a_{plan}) to total actions taken during an episode, tracking the agent's usage of the world model. All the metrics are averaged over interaction trajectories on multiple mazes.

World model diagnostics. To evaluate the internal fidelity of planning agents on out-of-domain (OOD) tasks, we assess the frozen world-model heads on test maze interactions using three specific metrics:

1. **Prediction accuracy:** The fraction of time steps where the model's argmax prediction matches the ground truth for next-state (y_{t+1}) and goal location (g).
2. **Stand-still error rate:** We filter for all incorrect next-state predictions and calculate the fraction where the model predicted no movement ($y_{t+1} = y_t$) despite a valid physical transition occurring.
3. **Working memory stability:** We measure reward-location prediction accuracy as a function of discrete steps elapsed *since the first reward encounter*. Trajectories are aligned at the moment right before discovery (Step 0), and accuracy is computed for subsequent steps (1–9) after reward was discovered, before the second reward hit, to quantify the degradation of the goal representation in recurrent dynamics over time. It is important to note that decision step 1 is the step before teleportation (during which the agent's sampled action is ignored) and step 2 is the one after teleportation.

2.9.2 Foraging analysis

Reward rate and efficiency. Foraging performance is quantified by the Empirical Reward Rate ($\hat{\rho}$), computed as the total accumulated reward divided by the total physical time elapsed in an interaction trajectory of 5000 steps (evaluation steps are longer than training). To isolate steady-state behaviour, a burn-in period (500 steps) is discarded from the trajectory before computation. We define Efficiency as the performance relative to the normative ceiling:

$$\text{Eff} = \frac{\hat{\rho}}{\rho^\star} \times 100\%, \tag{31}$$

where $\rho^\star$ is the optimal average reward rate derived via the average-reward policy iteration method described in Section 2.7.

Parameter sweeps for generalisation. To measure robustness to distributional shift, we evaluated Efficiency on a fixed baseline environment sampled from the centre of the training bounds, and then swept one task parameter at a time outside the training range while holding all other parameters at their baseline values. The baseline task instance ω_0 fixed the patch maxima to $\lambda_{\max,0} = 1.0$, $\lambda_{\max,1} = 0.7$, and $\lambda_{\max,2} = 0.4$, and used shared dynamics across patches: $\kappa_i = 0.2$ and $\delta_i = 10.5\,\mathrm{s}$ for all i, with exploitation duration $D_{\mathrm{ex}} = 7\,\mathrm{s}$ and symmetric travel times $D_{ij} = 17.5\,\mathrm{s}$ for all $i \neq j$. Depletion used $\rho_i = 0.9$ and $d_i = 2$. The specific sweep ranges evaluated are:

- **Inter-patch interval (IPI):** $[1.75, 180]\,\mathrm{s}$ (Training range: 3.5–35 s).
- **Replenishment rate:** $[0.014, 0.43]\,\mathrm{Hz}$ (Training range: 0.028–0.086 Hz).
- **Depletion jump size:** $[1, 10]$ (Training range: 1–3).

behavioural characterisation (leave thresholds). To compare the agent's patch-leaving decisions against the optimal policy and heuristics, we extract Leave Threshold Distributions. We execute the agent on a fixed test environment to extract an extended interaction trajectory (15,000 steps, 2000 burn-in). For every event where the agent chooses to leave a patch, we record the instantaneous reward probability of that patch at the moment of departure. We similarly ran the optimal policy on the same environment with same number of steps and burn-in. These empirical distributions are then compared against the deterministic thresholds of the normative baselines (Optimal, 1-Threshold, and 3-Threshold policies) derived in Section 2.7.

2.10 Software

The full codebase in Python, including environment generation logic, meta-RL training pipelines and average-reward solver implementations, is available at `https://github.com/CMC-lab/rolloutRNN.git`.

3 Results

3.1 Stress-testing Planning against structural novelty

To investigate the prerequisites for neural planning in foraging, we first characterised the stability and generalisation capabilities of a meta-RL planning architecture on a controlled maze navigation benchmark.

For mazes with 3 shortcuts, after training, both planning and non-planning agents converged to a similar asymptotic average reward of approximately 7.0 (Figure 2**a**). Exploration efficiency, measured by the average time-to-first-reward, decreased from 34 steps to 15 steps for both agents, matching the theoretical optimal worst-case exploration distance (Figure 2**b**). The fraction of planning actions sampled by the planning agent initially decreased below chance (0.2) before increasing to a stable plateau of approximately 0.32 after 2×10^6 training episodes (Figure 2**c**). Concurrently, the internal world model achieved a next-location prediction accuracy of ~ 0.9 and a reward-location prediction accuracy of ~ 0.7 on test episodes (Figure 2**d**).

Explicit planning conferred a performance advantage over the non-planning baseline only in environments structurally similar to the training distribution. For agents trained on dense mazes (0 shortcuts), the planning agent achieved a higher average reward than the non-planning baseline on test mazes with mild structural novelty (0 to 3 shortcuts) (Figure 3**a**). However, on test mazes with high structural novelty (5 to 12 shortcuts), the planning agent achieved a lower average reward than the non-planning baseline. This pattern of localised advantage was robust across training conditions. Agents trained on medium-difficulty mazes (6 shortcuts) exhibited a planning advantage on test mazes with 2 to 9 shortcuts, but performed worse than the baseline on mazes with 0–2 and 9–12 shortcuts (Figure 3**b**). Similarly, agents trained on easy mazes (10 shortcuts) showed a planning advantage peaking at 8 shortcuts, which disappeared on test mazes with 0–2 and 9–12 shortcuts (Figure 3**c**).

The poor performance of dense-trained planning agents in structurally novel, sparse environments coincided with a degradation in world model prediction accuracy. For the agent trained on dense mazes, the world model's next-state prediction accuracy on test mazes declined from ~ 1.0 on

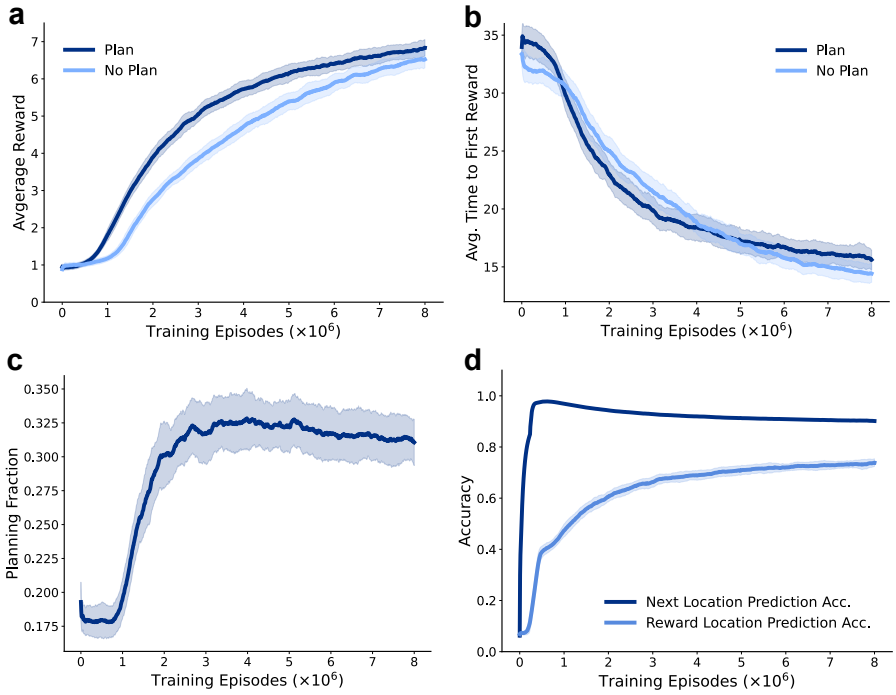

Figure 2: **Training dynamics and world model acquisition in the maze navigation task** ($n_{\text{sc}}^{train} = 3$)**. (a)** Average reward achieved on test mazes ($n_{\text{sc}}^{test} = 3$) across 8×10^6 training episodes. Shaded regions denote $\pm$ s.d. across five seeds. Both Planning (dark blue) and Non-Planning (blue) agents converge to a similar asymptotic mean reward (~ 7.0). **(b)** Average steps to first reward, representing exploration efficiency. Both agents converge to ~ 15 steps, while the planning agent exhibits marginally lower efficiency at convergence due to the temporal cost of planning steps. **(c)** Evolution of the *Plan* action probability during training. The fraction initially drops below chance (0.2), reflecting suppression of planning due to a poor initial world model, before rising to stabilise at ~ 0.32, indicating the successful learning of planning behaviour. **(d)** Accuracy of the internal world model heads on test episodes. Next-location prediction (dark blue) rapidly reaches near-perfect accuracy (~ 0.9) before slight degradation. Reward-location prediction (light blue) plateaus at ~ 0.7, bounded by the necessity of initial random exploration before the first reward encounter.

0-shortcut mazes to ~ 0.7 on 12-shortcut mazes (Figure 4**a**). The fraction of "stand-still" errors, defined as next-state predictions of no movement during a valid state transition, increased sigmoidally with task sparsity, peaking at ~ 0.9 on the 12-shortcut mazes (Figure 4**b**). Furthermore, the stability of reward location memory degraded in novel environments. While reward-location prediction accuracy remained near 1.0 for the duration of the episode following the first reward on near-training mazes (0 and 4 shortcuts), this accuracy declined over interaction steps 3–9 following reward receipt on sparse out-of-distribution mazes (8 and 12 shortcuts) (Figure 4**c**).

3.2 Capacity for Strategic Foraging

To determine whether the underlying meta-RL architecture possesses the representational capacity for strategic foraging, we trained model-free agents in a continuous-time 3-patch environment with full state observability.

Meta-RL agents with full observability successfully learned to maximise reward rates in continuous-time foraging environments, approaching theoretical optimality. Over the course of 1×10^5 training episodes, the RNN agent consistently improved its average reward rate on test environments, rising from an initial value of < 0.025 to oscillate around 0.045 by the end of training (Figure 5**a**). This performance gain was driven by the minimisation of the policy gradient loss, which decreased from an initial value of ~ 12 to stabilise around ~ 5 (Figure 5**b**). Crucially, the agent's efficiency, defined as the ratio of its empirical reward rate to the theoretical optimal rate derived via average-reward

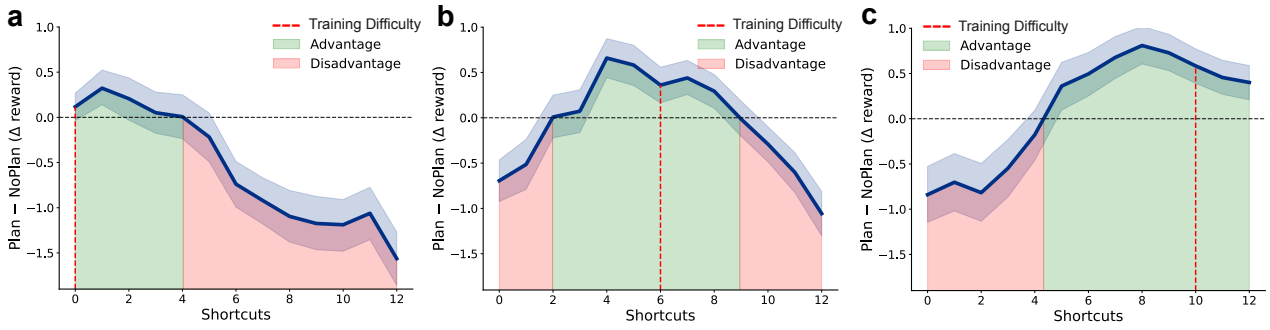

Figure 3: **Explicit planning confers a generalisation advantage only in task families structurally similar to the training distribution.** **(a)** Performance difference ($\Delta\text{Reward} = \text{Reward}_{\text{Plan}} - \text{Reward}_{\text{NoPlan}}$) on test mazes ranging from 0 to 12 shortcuts (increasing sparsity) for agents trained on dense mazes (0 shortcuts). Positive values (green shade) indicate a planning advantage; negative values (red shade) indicate a disadvantage. Planning is beneficial only for near-transfer tasks (0–4 shortcuts) and detrimental for far-transfer tasks (8–12 shortcuts). **(b)** Performance difference for agents trained on medium mazes (6 shortcuts). The planning advantage is localised to a window of structural similarity (∼2–9 shortcuts) around the training density. **(c)** Performance difference for agents trained on sparse mazes (10 shortcuts). The planning advantage shifts to easier, sparser mazes (5–12 shortcuts), confirming that the benefit of the planning module is highly specific to the training distribution's structure.

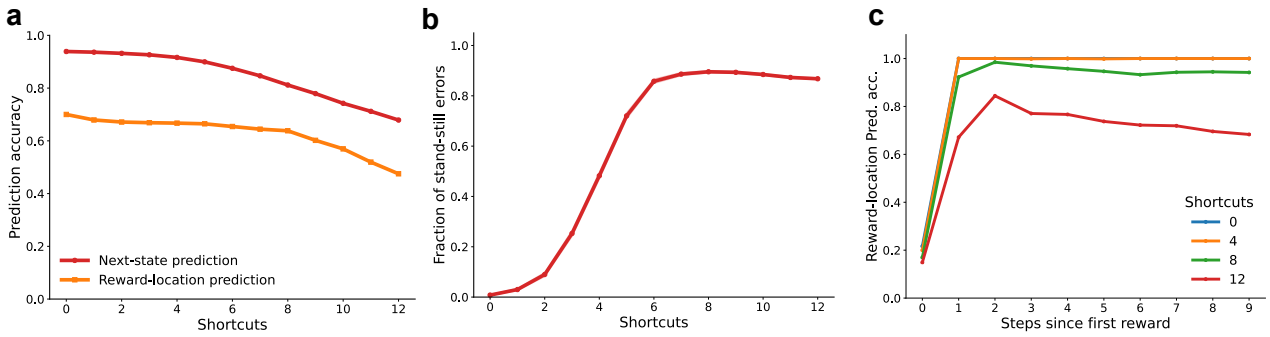

Figure 4: **Performance collapse in novel environments is driven by world model instability and mispredicted constraints** ($n_{\text{sc}}^{train} = 0$)**. (a)** World model prediction accuracy for the dense-trained agent evaluated across the full range of test difficulties (0–12 shortcuts). Both next-state (red) and reward-location (orange) prediction accuracies degrade as structural novelty increases. **(b)** Fraction of "stand-still" prediction errors: instances where the world model predicts no movement despite a successful state transition, as a function of maze sparsity. The sigmoidal rise to ~ 0.9 indicates that the agent mispredicts walls in open spaces when operating out-of-distribution. **(c)** Stability of working memory dynamics in novel environments. Reward-location prediction accuracy is plotted against steps since the first reward encounter until next reward is encountered. For familiar tasks (0, 4 shortcuts), accuracy is maintained at 1.0 (stable dynamics). For novel, sparse tasks (8, 12 shortcuts), accuracy rises and then degrades over steps 3–9, indicating that the recurrent dynamics is unstable to maintain the location of the reward in working memory for exploitation in subsequent steps.

policy iteration, improved from an initial random baseline of 60% to approximately 90% at the final training checkpoint (Figure 5**c**). Although the learning curve exhibited variance, with transient dips in efficiency at intermediate checkpoints, the final convergence to 90% indicates that the architecture is capable of approximating the optimal control policy for the continuous-time foraging Markov decision process.

The learned policy generalised robustly to task parameters within the training distribution but degraded when parameters were extrapolated significantly beyond training bounds. We evaluated the

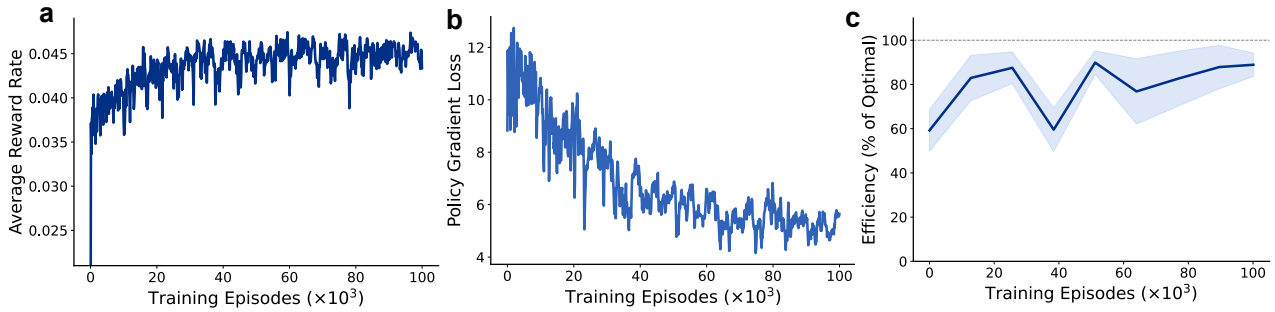

Figure 5: **Training dynamics and efficiency of meta-RL agents in continuous foraging.** **(a)** Average reward rate achieved on 200 test environments across training updates. The reward rate rises from < 0.025 to oscillate around 0.045, indicating successful learning. **(b)** Policy gradient loss during training. The loss decreases from ~ 12 to ~ 5, reflecting the optimisation of the actor's policy. **(c)** Efficiency of the RNN agent across training, calculated as the percentage of the theoretical optimal reward rate (derived via policy iteration) achieved on test environments. The agent improves from an initial baseline of 60%, to a final efficiency of $\sim 90\%$, demonstrating that the network can approximate the optimal policy.

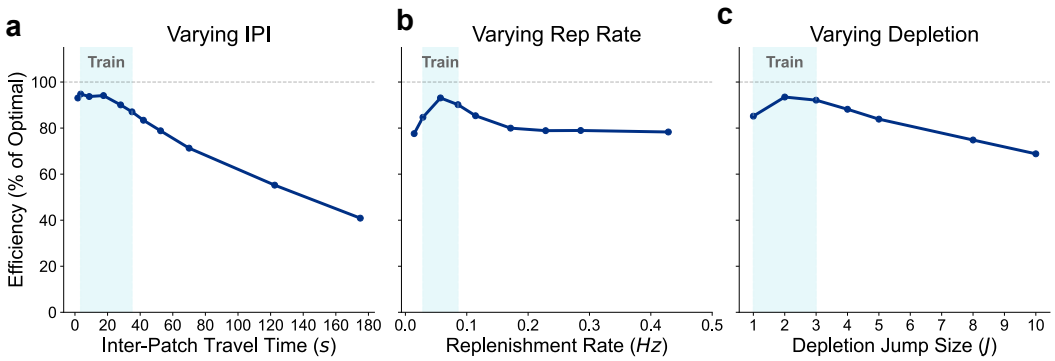

Figure 6: **Generalisation of the meta-learned foraging agent across task parameters.** **(a)** Efficiency as a function of inter-patch interval (IPI). The agent maintains high efficiency ($\sim 90\%$) within the training range (shaded blue, 3.5s–35s) but performance degrades on far out-of-distribution tasks (180s). **(b)** Efficiency as a function of replenishment rate. Performance peaks at $\sim 95\%$ in the centre of the training regime and remains robust ($\sim 80\%$) across the tested range. **(c)** Efficiency as a function of depletion jump size. Efficiency peaks at $\sim 95\%$ for jumps of size 2 and degrades gradually to $\sim 70\%$ for large jumps (size 10), indicating robust generalisation near the training distribution, but degradation on far out-of-distribution tasks.

generalisation of the learned meta-policy by systematically varying task parameters around a central baseline configuration. The agent maintained high efficiency ($\sim 90\%$) for inter-patch intervals (IPI) ranging from 1.75s to 20s, covering the majority of the training range (3.5s to 35s) (Figure 6**a**). However, performance degraded outside this window, dropping to $\sim 40\%$ efficiency at an IPI of 180s. Similarly, efficiency remained robust ($> 80\%$) across a wide range of replenishment rates (0.028 Hz to 0.43 Hz) and depletion jump sizes (1 to 3), peaking at $\sim 95\%$ efficiency in the center of the training regime (Figure 6**b,c**). The agent's performance only declined significantly when depletion jumps were increased to 10 ($> 3\times$ the training maximum), demonstrating that the model-free policy captured a generalised solution valid across the trained parameter space.

The RNN agent significantly outperformed heuristic threshold policies, achieving reward rates second only to the theoretical optimum. In a specific test environment characterised by asymmetric patch parameters (Figure 7**a**), the RNN achieved an average reward rate of 0.0569, surpassing both the optimal three-threshold policy (0.0512) and the optimal one-threshold policy (0.0441) (Figure 7**b**). The RNN's performance gap to the theoretical optimum (0.0615) was relatively small, further validating the good efficiency observed in the broader test set. By outperforming the three-

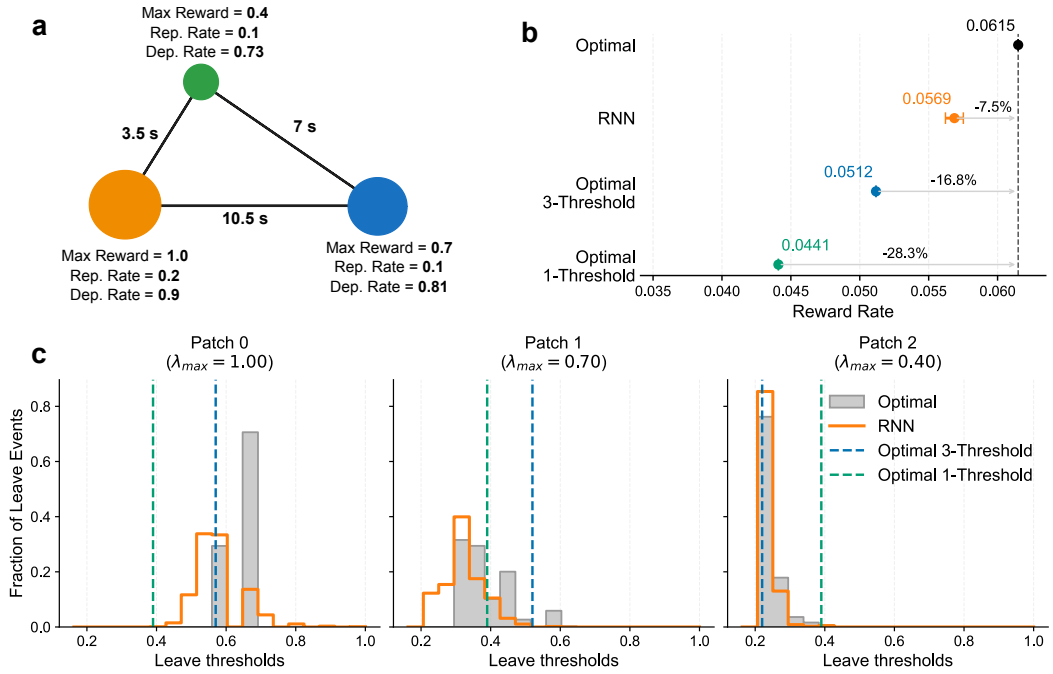

Figure 7: **Comparison of the RNN policy against normative baselines and optimal behaviour.** **(a)** Schematic of the 3-patch test environment with asymmetric parameters used for detailed policy analysis. **(b)** Average reward rates for the Optimal policy (0.0615), RNN (0.0569), Optimal 3-Threshold (0.0512), and Optimal 1-Threshold (0.0441). The RNN outperforms both heuristic threshold policies, achieving a reward rate closest to the theoretical optimum. **(c)** Distributions of patch-leaving thresholds (reward probability at departure) for Patches 0, 1 and 2. The RNN (orange) closely matches the optimal policy's (grey) leaving distribution for Patch 2 and is relatively similar for Patches 0 and 1, exhibiting a slightly more conservative (lower threshold) distribution for these patches compared to the optimal.

threshold heuristic, the RNN demonstrated that it had learned a more sophisticated control strategy than a simple fixed-threshold patch leaving.

Analysis of patch-leaving decisions revealed that the RNN's strategy closely approximated the optimal policy's switching thresholds. We compared the distribution of reward probabilities at the moment of patch departure for the RNN and the optimal agent. For Patch 2, the RNN's leave threshold distribution was nearly identical to that of the optimal policy (Figure 7**c**). For Patch 0, the RNN identified the leave threshold at ~ 0.6 similar to the optimal agent and the three-threshold heuristic, but missing the higher leave threshold identified by the optimal agent. For Patch 1, the RNN's leave distribution overlapped with the optimal thresholds but, similar to Patch 0, exhibited a "tail" of lower values, indicating a slightly more conservative strategy where the agent occasionally stayed longer than optimal. Overall, the structural similarity between the RNN's leaving statistics and those of the optimal agent suggests that the meta-RL training procedure converged to a policy that mimics the optimal solution's dynamics.

4 Discussion

This study examined whether task-trained recurrent neural networks can support time-costed control in two settings that isolate complementary computational demands: (i) a maze navigation benchmark used to probe explicit prospective computation via a learned rollout mechanism, and (ii) a continuous-time multi-patch foraging task designed to emphasise reward-rate control under heterogeneous action durations. Across both settings, the core objective was to establish a practical meta-reinforcement learning workflow in which recurrent dynamics implement within-episode adaptation,

then use controlled distribution shifts and normative baselines to characterise what is learned and where it breaks.

In the maze navigation task, planning-enabled agents learned to deploy the explicit "Plan" action only after the auxiliary world model became sufficiently accurate, indicated by an early suppression of planning below chance followed by a sustained rise in planning frequency during training (Figure 2**c–d**). Despite this learned planning behaviour, planning and non-planning agents converged to similar asymptotic rewards on the held-out test set drawn from the training family (Figure 2**a**), and both approached similar exploration efficiency in time-to-first-reward (Figure 2**b**). These results indicate that, within the training distribution, a recurrent policy without explicit rollouts can amortise much of the required computation, leaving limited headroom for explicit online planning to improve final performance.

Generalisation analyses revealed that any planning benefit was localised to task families that were structurally similar to the training distribution, whereas performance degraded under larger distribution shifts. When evaluated across shortcut-defined maze families, the planning advantage was positive only within a bounded window around the training difficulty, and became negative on far-transfer families (Figure 3**a–c**). Mechanistically, this collapse coincided with a systematic degradation of the learned world model heads as structural novelty increased (Figure 4**a**). Errors were dominated by stand-still mispredictions in sparse mazes (Figure 4**b**), and the agent's reward-location memory became less stable after the first reward in far-transfer environments, with accuracy degrading over subsequent steps (Figure 4**c**). Together, these findings support the interpretation that the failure mode under far transfer is not solely a policy mismatch but involves instability of internal predictive dynamics when the recurrent neural network interacts outside the structural regime it was trained on.

In the continuous-time foraging task, the primary outcome was validation that a model-free recurrent agent can learn reward-rate control in a fully observed setting for the implemented environment distribution. Across training, the meta-RL agent increased reward rate on held-out environments and reached approximately 90% of the optimal reward rate computed by policy iteration, albeit with instability across checkpoints (Figure 5**a–c**). The learned policy generalised well within the training bounds of key parameters but degraded under substantial extrapolation, with particularly marked sensitivity to large increases in inter-patch travel time (Figure 6**a**) and more gradual degradation under large depletion jumps (Figure 6**c**). In a detailed single-environment analysis, the recurrent policy achieved reward rates closer to the optimal policy than heuristic one- or three-threshold baselines, and its patch-leaving statistics broadly resembled the optimal policy's leaving distributions while showing conservative deviations in some patches (Figure 7**a–c**). This indicates that the learned policy captures structured, patch-specific control that cannot be reduced to a single global leave threshold.

Finally, although both tasks were formulated using time-aware interaction variables (action durations and elapsed time), the results primarily support task-specific conclusions rather than a strong claim about a unifying formalism. In maze navigation, action time costs mainly determine a trade-off between planning and acting within a fixed time budget; in foraging, time enters the objective directly via reward-rate evaluation and via a discounted surrogate used for stable learning. The overall contribution is therefore empirical: (i) a replication and stress test of a recurrent planning architecture under controlled distribution shift, identifying failure signatures tied to predictive and working-memory instability (Figures 3–4), and (ii) a continuous-time foraging environment with normative optimal baselines in which meta-trained recurrent agents can approach optimal reward-rate control under full observability (Figures 5–7), providing a foundation for subsequent work that removes omniscience and revisits explicit planning mechanisms in foraging-relevant regimes.

4.1 Maze navigation: distribution-specific benefits of explicit planning and mechanisms of failure under far transfer

In the maze benchmark, explicit rollouts did not produce a large asymptotic advantage over a recurrent baseline trained without planning. Both agents converged to similar reward and exploration efficiency (Figure 2**a–b**), despite the planning agent acquiring a usable one-step world model (Figure 2**d**) and maintaining a non-trivial planning-action frequency (Figure 2**c**). This pattern is consistent with the idea that, for the training family, a sufficiently expressive recurrent policy can amor-

tise the requisite computations: exploration, goal memory after the first hit, and shortest-path-like exploitation, without requiring additional online simulation at decision time. In that sense, these experiments separate two questions: whether a trained planner *uses* and benefits from planning, versus whether planning is *functionally necessary* when a non-planning recurrent agent is trained under matched conditions. These results contextualise the reported poor performance of planner agents when rollouts were suppressed (Jensen et al., 2024), since they indicate only that the planner has learned to rely on rollouts for maze navigation during training, not an intrinsic advantage of planning.

The shortcut-family analysis clarifies when rollouts provide functional benefit: planning advantage is localised to test families that remain structurally similar to the training distribution (Figure 3**a–c**). This supports a conservative interpretation in which explicit planning is helpful in a near-transfer regime where the learned world model remains accurate enough for imagined trajectories to provide actionable information, but where the recurrent baseline still benefits from additional on-the-fly computation beyond its cached policy. The absence of a uniform advantage across difficulty also cautions against a purely behavioural reading ("planning helps on hard tasks"), since the relevant axis in these experiments is not difficulty alone but distributional match between training-induced internal representations and the test family's transition structure.

Under far transfer, the key result is not merely that world model accuracy drops (Figure 4**a**), but that the model errors take a form that is particularly damaging for a rollout-based controller. The rise in stand-still mispredictions with increasing sparsity indicates that the learned transition head systematically under-predicts movement in open environments, effectively inserting spurious constraints into imagined dynamics (Figure 4**b**). Because rollouts query the model repeatedly, this error mode can be amplified across simulated steps, biasing the agent away from valid paths and wasting its limited time budget on uninformative internal computation. In parallel, the degradation of reward-location memory after the first reward in sparse mazes suggests that far-transfer failure also involves instability of recurrent working-memory dynamics, not only a miscalibrated transition predictor (Figure 4**c**). A plausible coupled mechanism is that unreliable rollouts generate corrupted summary inputs (planned action sequence and success flag) that the recurrent state must integrate with external observations, thereby destabilising the goal-memory trace needed for exploitation. This is a concrete alternative to a just-so "overthinking" interpretation: the issue is not that deliberation is intrinsically harmful under novelty, but that deliberation can become harmful when the internal simulator and the recurrent state occupy an out-of-distribution regime where errors are structured and self-reinforcing.

Several targeted analyses would discriminate these mechanisms without introducing new results claims. First, one can test whether planning-generated inputs drive reward-memory degradation by suppressing planning at test time (while keeping the trained weights fixed) and then re-evaluating both behaviour and the world-model heads (cf. Figure 4**c**). If the reward-location predictions stabilise when rollouts are disabled, this would implicate rollout summaries as a destabilising input stream rather than treating head degradation as an unavoidable consequence of maze sparsity. Second, a possible probe is an "oracle planner" that interacts with the real environment during planning, rather than a learned world model. While this would likely diagnose whether failures are driven by model error per se, it changes the computational contract of the task: in a fixed time-budget setting where planning steps are cheaper than physical steps, oracle interaction during "planning" effectively grants additional environment queries and can be viewed as a form of privileged action. For that reason, its principal value here is diagnostic: separating "planning algorithm/architecture" limitations from "learned world model" limitations, rather than serving as a fair performance baseline for planning as internal simulation.

These findings also motivate experiments that vary resource constraints to isolate the computational role of planning. If the non-planning agent matches the planner because it has sufficient capacity to cache the solution, then reducing recurrent capacity (or otherwise limiting amortisation) should create regimes in which rollouts become beneficial. A controlled sweep over hidden-state size for both agents, paired with the same shortcut-family transfer evaluation (Figure 3**a–c**), would test whether explicit planning functions as a compensatory mechanism when the recurrent controller is under-capacity. This would move the interpretation from "planning is fragile" to the more precise claim that planning can be beneficial when it supplies computation that the reactive policy cannot represent or generalise with its available memory.

Taken together, the maze results sharpen the scope of what Jensen-style rollouts currently demonstrate. The architecture is a rare end-to-end instantiation of meta-trained recurrent planning with learned internal simulation, and it has compelling links to replay-like phenomena in neuroscience (Jensen et al., 2024). At the same time, the transfer failures highlight a structural limitation: using the same recurrent substrate for external control and internal simulation, with a world model read out from that same state, couples model error to policy updates and to memory stability (Figure 4). This motivates future variants that relax the assumption that planning and acting share identical dynamics and representational form, either by introducing explicit multi-timescale recurrent computation (self-ticking or hierarchical recurrent stacks) or by shifting planning into a learned latent space that is less tied to task-specific state encodings (Ha and Schmidhuber, 2018; Wang et al., 2025).

4.2 Continuous-time foraging: validated regime, objective alignment, and generalisation limits

The foraging experiments provide limited validation that a meta-trained RNN policy can approximate high-reward-rate control when given full observability of the environment state and access to the task parameters that define the transition dynamics. In this setting, optimal performance is well-defined via average-reward policy iteration on the induced fully observed Markov decision process, enabling an efficiency metric given by the ratio of the achieved reward rate to the optimal reward rate. Using this evaluation, the meta-RL agent reached $\sim 90\%$ efficiency on held-out environments sampled within the training regime (Figure 5**c**) and achieved reward rates closer to the optimum than both one-threshold and three-threshold heuristics in a specific asymmetric environment (Figure 7**b**). The distributions of leaving thresholds further suggest that the learned policy captures components of the optimal switching structure, closely matching the optimal leaving distribution for one patch while exhibiting a conservative tail for others (Figure 7**c**), indicating that the learned control is more structured than fixed-threshold baselines but still imperfect relative to the optimal solution.

A central implementation issue in continuing foraging control is objective alignment: the normative quantity of interest is the long-run reward rate (rewards per unit time), whereas standard episodic deep-reinforcement learning objectives optimise discounted or finite-horizon returns, which can introduce sensitivity to episode length. Average-reward reinforcement learning provides a principled formalisation, but learning an average-reward critic typically requires estimating the environment's gain alongside differential values, and naive gain estimation can inject high variance into temporal-difference targets. For training stability, the implementation therefore used a time-aware discounted surrogate with per-step discounting $\gamma_t = \exp(-\beta \Delta\tau_t)$, combined with n-step bootstrapping under truncated backpropagation through time. This preserves the constraint that longer actions discount future rewards more strongly and is consistent with continuous-time discounting formulations in reinforcement learning (Bradtke and Duff, 1994). However, it remains an approximation: a fixed $\beta > 0$ induces a finite effective horizon in physical time, so the learned policy is not optimised directly for the average-reward limit. The empirical comparison to average-reward optimal policies is therefore not only a performance metric but also an initial validation that this surrogate objective can produce policies that score well under the intended reward-rate criterion within the tested regime (Figure 5**c** and Figure 7**b**).

The parameter-sweep results indicate robust generalisation near the training distribution and systematic degradation under strong extrapolation, especially for travel-time scaling (Figure 6**a**). Performance remained high across most of the trained inter-patch travel-time range but dropped sharply for very large travel times, suggesting that the learned policy does not implement a scale-invariant representation of opportunity cost and is calibrated to the temporal statistics seen during training. The observed generalisation profile is consistent with a policy that has learned a training distribution-tuned approximation to the optimal control rather than a general algorithmic rule that extrapolates across time scales without additional inductive bias or broader training coverage. The bias of meta-trained agents to their training distribution presents an important nuance to interpret any downstream analysis of their learned dynamics for mechanistic claims.

4.3 Future work

A first priority is to make the average-reward objective approximation explicit and experimentally characterised rather than implicit. One direct axis is to vary β and quantify the trade-off between

training stability and reward-rate optimality under evaluation, testing whether efficiency monotonically improves as β decreases or whether smaller β induces instability through higher-variance critic targets. In parallel, more faithful average-reward training variants can be tested by introducing explicit gain estimation (separately learned average reward) and differential-value learning, with careful variance-control strategies, to determine when the additional complexity yields measurable benefits over the discounted surrogate. A recently introduced average-reward soft actor-critic method (Adamczyk et al., 2025) could also be experimented with.

A second priority is a staged roadmap that progressively removes the agent's omniscience in patch foraging while retaining principled ceilings for comparison. The fully observed regime is useful as a capacity and pipeline validation (Figures 5–7), but ecological foraging requires inference over hidden patch states and often over latent task parameters. A controlled progression can separate difficulties due to partial observability from difficulties due to long-horizon control: (i) hide other patch states but keep deterministic replenishment and known parameters so the optimal inference is computable; (ii) hide task parameters so the agent must infer dynamics online; (iii) provide belief-state summaries as inputs to isolate control from inference; and (iv) move toward reward-and-action-history-only inputs as the most ecologically constrained setting. For each stage, the goal is to pair meta-RL agents with the strongest feasible normative or algorithmic baselines (exact solvers when available, approximate solvers otherwise) to keep interpretation grounded in performance ceilings rather than in heuristics.

A third priority is to reintroduce planning for foraging in a way that is well-suited for continuing tasks. The open question is what form of prospective computation is learnable and useful under meta-RL: explicit rollouts over simulated discrete patch states; over belief states; or use hierarchical temporal abstraction where planning occurs using "temporal options" at a more abstract timescale beyond travel times (Sutton et al., 1999). Concretely, planning-enabled variants can be prioritised around testable claims: whether planning improves reward rate under (a) increased nonstationarity, (b) increased uncertainty about patch dynamics, or (c) constrained recurrent capacity that limits amortisation. These tests align with the broader hypothesis that deliberative computation should be recruited when cached policies are insufficient and that replay-like substrates can support either online or offline computation depending on task demands.

To conclude, we replicate and stress-test a learned-rollout planning agent and introduce a continuous-time foraging benchmark with normative reward-rate optima. The maze results delimit when planning helps and identify failure signatures under distribution shift, while the foraging results validate that meta-trained RNNs can achieve near-optimal reward-rate control under full observability. These benchmarks now make it possible to study, in a controlled and principled way, how planning might interact with partial observability and uncertainty in foraging-relevant settings.

ACKNOWLEDGEMENTS

I am deeply grateful to Dr. Roxana Zeraati for directly supervising this rotation. Her guidance was consistently demanding in the best way: she pushed me to become a more careful researcher, especially in how I manage my research progress and communicate ideas clearly. I will carry forward the advice, standards, and practical skills I gained through our work together. I thank Prof. Peter Dayan for welcoming me into the AGPD lab and for fostering such a stimulating research environment. I am also grateful to Dr. Shervin Safavi for sharing his code organisation structure, Dr. Andrew Webb for his prompt support with compute infrastructure, and Ali Gholamzadeh for many insightful discussions on RL and cognition. Finally, I thank Kris Jensen for open-sourcing the maze agent implementation, which was instrumental in enabling the rapid replication and extension of the benchmark within my rotation timeframe.

REFERENCES

Adamczyk, J., Makarenko, V., Tiomkin, S., and Kulkarni, R. V. (2025). Average-Reward Soft Actor-Critic. arXiv:2501.09080 [cs].

Binz, M., Dasgupta, I., Jagadish, A., Botvinick, M., Wang, J. X., and Schulz, E. (2023). Meta-Learned Models of Cognition. arXiv:2304.06729 [cs].

Bradtke, S. and Duff, M. (1994). Reinforcement Learning Methods for Continuous-Time Markov Decision Problems. In *Advances in Neural Information Processing Systems*, volume 7. MIT Press.

Charnov, E. L. (1976). Optimal foraging, the marginal value theorem. *Theoretical Population Biology*, 9(2):129–136.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016). RL$ˆ2$: Fast Reinforcement Learning via Slow Reinforcement Learning. arXiv:1611.02779 [cs].

Eldar, E., Lièvre, G., Dayan, P., and Dolan, R. J. (2020). The roles of online and offline replay in planning. *eLife*, 9:e56911.

Ha, D. and Schmidhuber, J. (2018). World Models. arXiv:1803.10122 [cs].

Hall-McMaster, S., Dayan, P., and Schuck, N. W. (2021). Control over patch encounters changes foraging behavior. *iScience*, 24(9):103005.

Hall-McMaster, S. and Luyckx, F. (2019). Revisiting foraging approaches in neuroscience. *Cognitive, Affective, & Behavioral Neuroscience*, 19(2):225–230.

Harhen, N. C. and Bornstein, A. M. (2023). Overharvesting in human patch foraging reflects rational structure learning and adaptive planning. *Proceedings of the National Academy of Sciences*, 120(13):e2216524120. Publisher: Proceedings of the National Academy of Sciences.

Jensen, K. T., Hennequin, G., and Mattar, M. G. (2024). A recurrent network model of planning explains hippocampal replay and human behavior. *Nature Neuroscience*, 27(7):1340–1348. Publisher: Nature Publishing Group.

Kurth-Nelson, Z., Economides, M., Dolan, R., and Dayan, P. (2016). Fast sequences of non-spatial state representations in humans. *Neuron*, 91(1):194–204.

Mahadevan, S. (1996). Average reward reinforcement learning: Foundations, algorithms, and empirical results.

Mikulik, V., Delétang, G., McGrath, T., Genewein, T., Martic, M., Legg, S., and Ortega, P. A. (2020). Meta-trained agents implement Bayes-optimal agents. arXiv:2010.11223 [cs].

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. arXiv:1602.01783 [cs].

Mobbs, D., Trimmer, P. C., Blumstein, D. T., and Dayan, P. (2018). Foraging for foundations in decision neuroscience: insights from ethology. *Nature Reviews Neuroscience*, 19(7):419–427.

Stephens, D. W., . K. J. R. (1988). Stephens, d. w., and j. r. krebs. foraging theory. princeton university press. *Journal of Mammalogy*, 69(4):877–877.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211.

Wang, G., Li, J., Sun, Y., Chen, X., Liu, C., Wu, Y., Lu, M., Song, S., and Yadkori, Y. A. (2025). Hierarchical Reasoning Model. arXiv:2506.21734 [cs].

Wang, J. X., Kurth-Nelson, Z., Kumaran, D., Tirumala, D., Soyer, H., Leibo, J. Z., Hassabis, D., and Botvinick, M. (2018). Prefrontal cortex as a meta-reinforcement learning system. *Nature Neuroscience*, 21(6):860–868. Publisher: Nature Publishing Group.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2017). Learning to reinforcement learn. arXiv:1611.05763 [cs].

Zeraati, R. (2025). Optimal Foraging by Learning the World Model. *Cognitive Computational Neuroscience (CCN) 2025*.