HOMER: RATIONAL META-REASONING FOR HIERARCHICAL REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

A key capability for sequential decision-making agents is *rational meta-reasoning*: the ability to adaptively balance the requisite computing time to solve a problem against the time taken to perform those computations. Rational meta-reasoning agents can compute for the minimum amount of time needed to generate high-quality decisions, thereby avoiding poor decisions from insufficient computation or wasteful computations with marginal utility. In this work, we study meta-reasoning in the context of model-free hierarchical reinforcement learning (HRL) agents, composed of a manager who assigns k-step subgoals to a worker. Since each level in the hierarchy optimizes a different objective, at a different temporal horizon, it raises the question of whether the requisite computation at each level is different? Motivated by this question, we first formalize meta-reasoning for HRL agents, and then introduce HOMER, a hierarchical rational meta-reasoning agent, that instantiates the meta-reasoning process through iterative computations within model-free recurrent neural networks. We then apply HOMER to the offline goal-conditioned RL setting and demonstrate that it successfully attains goal-reaching success rates on par with non-meta-reasoning agents while using far less computation. We then empirically study the adaptive computations deployed at each level of the hierarchy; we observe that the worker's adaptation of compute correlates with distance to subgoals whereas the manager's adaptation of compute correlates with the entropy of the empirical k-step distribution induced by the offline dataset. Finally, we show that a meta-reasoning manager can cache previous computations to improve its goal-reaching success rates and meta-reasoning efficiency.

1 INTRODUCTION

Humans are naturally bounded by the finite amount of time available to solve problems. Under this constraint, we have the remarkable capacity to manage our computational resources and thinking time to solve a problem before executing an action (Good, 1952; Simon, 1972; Griffiths et al., 2019; Griffiths, 2020). For easy problems, we tend to deliberate less since over-thinking can often be unnecessary and wasteful whereas, for hard problems, we deliberate more as under-thinking can lead to poor decisions. In other words, we *meta-reason*: under computational time constraints, we reason about when additional computing time is useful to solve a problem and when it is likely to be wasteful. Similarly, sequential decision-making agents may also have computational time constraints, and they must adapt their computing time appropriately depending on the problem (Horvitz et al., 1989; Russell & Wefald, 1991; Arumugam et al., 2024; Javed & Sutton, 2024).

In this work, we study meta-reasoning for a feudal hierarchical reinforcement learning (HRL) agent with a manager-worker architecture (Dayan & Hinton, 1992). Each level in the hierarchy operates under a distinct objective and problem horizon: the manager must output optimal k-step subgoals towards the long-horizon main goal, while the worker must output optimal primitive actions towards the k-step horizon subgoal. This structure raises the following research question:

Does the hierarchy of decision-making in an* HRL *agent imply a hierarchy of meta-reasoning strategies and computational needs?

While prior work has studied meta-reasoning, they: 1) have exclusively tackled the bandit or flat RL agent setting (Hay et al., 2012; Lin et al., 2015; Callaway et al., 2018), 2) have viewed computation

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

primarily in terms of model-based planning (Chung et al., 2023; Wang et al., 2025; Muppidi et al., 2026), and 3) made restrictive assumptions or rely on expert data to learn the meta-reasoning policy (Hay et al., 2012; Lin et al., 2015; Sung et al., 2021). On the other hand, we investigate our research question by studying HRL agents that use a simple end-to-end method to adapt the requisite compute in model-free recurrent neural networks (RNNs) based on problem complexity.

Hierarchical meta-reasoning agents with model-free RNNs are interesting for several reasons. Model-free RNNs serve as a generic model of computation and have shown promise in challenging environments without requiring an explicit model of the environment, while also encoding planning-like properties in their internal representations (Tamar et al., 2016; Guez et al., 2019; Bush et al., 2025). Their promise and flexibility imply that they deserve to be studied further in the context of meta-reasoning. Hierarchical agents are interesting because they relate to how humans approach problem solving (Solway et al., 2014; Eppe et al., 2022) and have shown promise over flat agents in several challenging environments (Klissarov et al., 2025; Park et al., 2025a;b).

In this work, we investigate hierarchical meta-reasoning and make the following contributions:

1. We formalize the hierarchical RL meta-reasoning problem (Section 4).
2. We design HOMER, an HRL meta-reasoning agent and apply it in the offline goal-conditioned RL setting (Section 5).
3. We empirically demonstrate that HOMER achieves goal-reaching success rates on par with non-meta-reasoning agents, but with much less computation (Section 6).
4. Through empirical analysis, we show that the worker's compute needs correlate with distance to subgoals, whereas the manager's needs correlate with the entropy of the empirical k-step distribution induced by the offline dataset (Section 6).
5. Finally, we show that the manager can improve its meta-reasoning efficiency while maintaining high goal-reaching success rates by reusing previous computations (Section 6)

2 Related Work

In this section, we outline related work in rational meta-reasoning, adaptive computation, recurrent architectures, and hierarchical reinforcement learning.

Rational Meta-reasoning. The area of rational meta-reasoning is well-suited to study computationally-bounded agents that need to adaptively adjust their compute usage (Good, 1952; Simon, 1972; Horvitz et al., 1989; Russell & Wefald, 1991). Hay et al. (2012); Callaway et al. (2018) proposed and provided several theoretical results on various greedy-based meta-reasoning policies. Lin et al. (2015) characterized the difficulty of solving the meta-reasoning problem as polynomially harder than solving the main task and developed approximate solutions based on an existing planning algorithm. Sung et al. (2021; 2024); Budd et al. (2024) proposed algorithms for learning meta-policies for motion and probabilistic planning but leverage an expert dataset to learn their meta-reasoning policy. Muppidi et al. (2026) proposed an algorithm to learn a gating policy that determines how much computational budget to allocate to a planner in a given state. Svegliato et al. (2022) used the rational meta-reasoning formalism to develop a framework for safe autonomous agents. Prior work (Gershman et al., 2015; Griffiths et al., 2019; Griffiths, 2020; Hanna & Corrado, 2025; Chuprov et al., 2026) take a cognitive science view and argue that current agents are unlikely to meta-reason unless computational constraints are imposed upon them. More recently, De Sabbata et al. (2024); Yan et al. (2025); Navardi et al. (2026); Liu et al. (2026) have applied rational meta-reasoning to large language models to improve their abilities to reason and generate plans for language and embodied agents. In our work, we extend the rational meta-reasoning formalism to the model-free hierarchical RL setting, resulting in a hierarchy of rational meta-reasoning problems—one per level of the HRL agent—whose interdependencies, to the best of our knowledge, remain unexplored in the literature.

Adaptive Computation. Adaptive computation can take different forms such as treating amount of compute as a hyperparameter when planning (Schrittwieser et al., 2019; Silver et al., 2018), learning a policy that determines when it should plan and for how long (Labiosa & Hanna, 2026a;b; Wang et al., 2025; Chung et al., 2023; Hamrick et al., 2017), or adapting the amount of recurrence in RNNs based on the prediction loss (Graves, 2017; Banino et al., 2021). In our work, we focus on adaptive

computation in RNNs. In particular, we use PonderNet (Banino et al., 2021), which was originally used for supervised learning, as part of our HRL agent. PonderNet is a simple, fully differentiable algorithm for modeling adaptive computation under computational constraints in RNNs. It essentially adjusts the number of recurrence steps in an RNN based on the task loss function, while ensuring the number of steps used is low.

Model-free Recurrent Architectures. The flexibility of rational meta-reasoning permits computation to be generic. As mentioned above, most work has focused explicitly on planning-based computation, which typically involves simulations of forward dynamics from the current state. In general, however, computation can even be extended to recurrence operations in RNNs (Graves, 2017; Banino et al., 2021). Model-free RNNs allow for flexible adaptive computation on the current state before emitting an action (Tamar et al., 2016; Guez et al., 2019; Ghugare et al., 2026). Prior work has shown RNNs can encode planning-like properties in their representations (Taufeeque et al., 2024; Bush et al., 2025), and that RNNs have been critical in developing agents that can generalize to long-horizon goals (Ghugare et al., 2026). In this work, we introduce our meta-reasoning agent with an LSTM (Hochreiter & Schmidhuber, 1997) for recurrence computations.

Hierarchical RL. Over the years, there have been a variety of formalisms for HRL (Dayan & Hinton, 1992; Parr & Russell, 1997; Sutton et al., 1999; Dietterich, 2000; Barto & Mahadevan, 2003; Klissarov et al., 2025). The two most popular frameworks are feudal-RL (Dayan & Hinton, 1992; Vezhnevets et al., 2017; Nachum et al., 2019) and options (Sutton et al., 1999; Bacon et al., 2017). Recently, feudal-RL agents have shown promise in solving a variety of long-horizon robotics tasks, especially in the offline goal-conditioned RL (GCRL) setting (Kaelbling, 1993; Park et al., 2025a). One quintessential example is Hierarchical Implicit Q-Learning (HIQL) (Park et al., 2023), which learns the manager and worker policies using a single value function learned from offline data. HIQL's demonstrated promise on challenging robotics benchmarks mitigates concerns about its reliability (Park et al., 2025a;b; Ahn et al., 2025; Hyeon et al., 2026), while its simple design enables a clean analysis of each level of the hierarchy. As such, while hierarchical meta-reasoning and our particular approach are applicable to a wide array of agents, we use HIQL as the canonical HRL agent in our study of adaptive, hierarchical computation for offline GCRL.

3 Offline Goal-Conditioned Reinforcement Learning

We consider a sequential decision-making problem formulated as a goal-conditioned Markov decision process (MDP) (Bellman, 1957; Kaelbling, 1993; Puterman, 2014). An MDP is defined as $\mathcal{M} := \langle \mathcal{S}, \mathcal{G}, \mathcal{A}, p, r, \gamma \rangle$, where $\mathcal{S}$ is the state-space, $\mathcal{G} \subseteq \mathcal{S}$ is the goal-space, $\mathcal{A}$ is the action-space, $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function, $r : \mathcal{S} \times \mathcal{G} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor; we refer to $\mathcal{M}$ as the *base* environment MDP. In the GCRL setting, an agent is tasked with reaching $G \in \mathcal{G}$ using a policy $\pi : \mathcal{S} \times \mathcal{G} \to \Delta(\mathcal{A})$. At time-step t, the agent in state S_t takes action A_t, receives sparse reward $R_t = \mathbf{1}_{\{G\}}(S) - 1$, and transitions to S_{t+1}. The value function of π is $v^\pi(s, g) := \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r(S_t, g) \mid S_0 = s]$. The agent learns the optimal policy $\pi^\star$ by maximizing the expected discounted return, $\max_\pi J(\pi) = \max_\pi \mathbb{E}_{G\sim\rho(\mathcal{G}),\zeta\sim\rho^\pi(G)}[\sum_{t=0}^{H-1} \gamma^t R_t]$, where H is the horizon, ρ is the goal distribution and $\rho^\pi(g) = d_0(S_0) \prod_{t=0}^{H-1} \pi(A_t \mid S_t, g) p(S_{t+1} \mid S_t, A_t)$ is the distribution over trajectories ζ, with d_0 denoting the initial-state distribution.

In the offline setting, the objective is to find $\pi^\star$ with access only to an offline dataset $\mathcal{D} := \{(S_t, A_t, S_{t+1})\}$. In the two-level feudal-RL setting (Dayan & Hinton, 1992), decision-making is divided between a manager and worker, each of which is represented by policies $\pi_h : \mathcal{S} \times \mathcal{G} \to \Delta(\mathcal{S})$ and $\pi_\ell : \mathcal{S} \times \mathcal{S} \to \Delta(\mathcal{A})$. The manager's action assigns a subgoal in $\mathcal{S}$ for the worker to reach in k time-steps, and the worker selects primitive actions given the assigned subgoal.

4 Problem Formulation: Hierarchical Rational Meta-reasoning

We now formalize hierarchical rational meta-reasoning. For clarity, we do so for a two-level hierarchical agent, but this formulation can be generalized to any number of levels.

Setup. We adapt the meta-level MDP formulation used in prior work (Hay et al., 2012; Lin et al., 2015) for the hierarchical setting. The meta-MDP is denoted by $\mathcal{M}_{\text{meta}} := \langle \mathcal{S}_{\text{meta}}, \mathbf{\Lambda}, p_{\text{meta}}, c_{\text{meta}} \rangle$,

where $\mathcal{S}_{\text{meta}} := \mathcal{S} \times \mathcal{G} \times \mathcal{T}$, $\mathcal{T}$ is the set of *internal computational states* (Russell & Wefald, 1991), including the initial state $\mathrm{T}_0 = \mathbf{0} \in \mathcal{T}$, $\mathbf{\Lambda}$ is the set of computation actions to execute including the end-computation action $\perp$, $c_{\text{meta}} : \mathcal{S}_{\text{meta}} \times \mathbf{\Lambda} \to \mathbb{R}_{<0}$ gives a scalar penalty for executing a computational action, and $p_{\text{meta}} : \mathcal{S}_{\text{meta}} \times \mathbf{\Lambda} \to \Delta(\mathcal{T})$ is the meta-transition function. Intuitively, one can view $\mathcal{T}$ as the internal representations in an arbitrary meta-reasoning architecture (for example, the hidden and cell states in an LSTM), $\mathbf{\Lambda}$ as the set of procedures applied to $\mathcal{T}$ (for example, the update rules in an LSTM), and p_{meta} as the (deterministic) transition of such procedures.

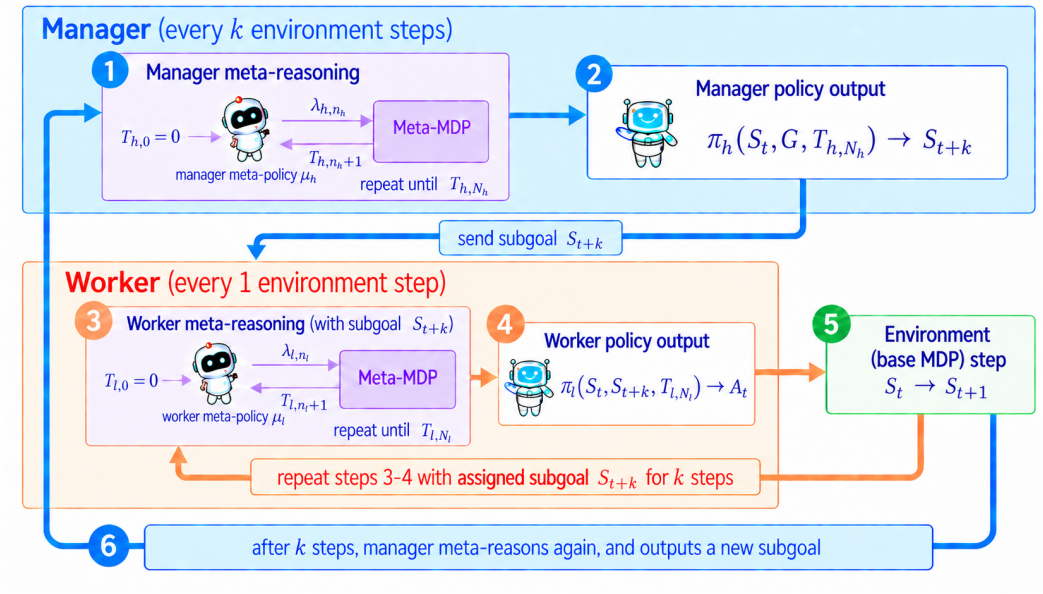

Figure 1: Hierarchical meta-reasoning setup with interaction between the meta-MDP and base MDP.

In the hierarchical rational meta-reasoning problem, each level of the hierarchy has its own meta-MDP. The worker and manager levels are each equipped with *meta-policies*, $\mu_\ell : \mathcal{S} \times \mathcal{S} \times \mathcal{T}_\ell \to \Delta(\mathbf{\Lambda})$ and $\mu_h : \mathcal{S} \times \mathcal{G} \times \mathcal{T}_h \to \Delta(\mathbf{\Lambda})$[1]. At each meta-reasoning time-step, each level's meta-policy decides whether to continue or terminate computing. For each level, we denote the meta-time-step at which its meta-policy terminates by N_ℓ and N_h, respectively. At a given level, the final computation states, T_ℓ and T_h, then influence the associated policies in the base MDP, $\pi_\ell : \mathcal{S} \times \mathcal{S} \times \mathcal{T}_\ell \to \Delta(\mathcal{A})$ and $\pi_h : \mathcal{S} \times \mathcal{G} \times \mathcal{T}_h \to \Delta(\mathcal{S})$, whose definitions are expanded to fold them in explicitly.

Meta-MDP and Base MDP Interaction. We refer the reader to Figure 1 for an illustration of the interaction process. At the base MDP time-step, t, the agent is in state S_t. Interaction then proceeds as follows: **(1)** the manager begins its meta-reasoning process at meta-time-step $n_h = 0$ from compute state $\mathrm{T}_{h,0} = \mathbf{0}$ and executes compute actions $\Lambda_{h,n}$ with μ_h until it terminates in T_{h,N_h}; **(2)** given T_{h,N_h}, the manager policy π_h then outputs the subgoal S_{t+k}; **(3)** given S_{t+k}, the worker begins its own meta-reasoning at meta-time-step $n_l = 0$ from compute state $\mathrm{T}_{\ell,0} = \mathbf{0}$ and executes compute actions $\Lambda_{\ell,n}$ with μ_ℓ until it terminates in T_{ℓ,N_ℓ}; **(4)** given T_{ℓ,N_ℓ}, the worker policy π_ℓ outputs the primitive action; **(5)** the worker proceeds to the next base MDP state S_{t+1} and repeats its meta-reasoning with the previously-assigned subgoal; **(6)** after k worker-steps are over, the manager meta-reasons to output the new subgoal. In this interaction, each level always meta-reasons before selecting a subgoal or action and the frequency of meta-reasoning (k-steps or 1 step) matches that of the respective base policy.

While $\mathrm{T}_0 = \mathbf{0}$ implies that computations are independent across decision-points, it is possible, however, that sharing computation states across decision-points is useful. In particular, we can use a cached τ from a previous decision-point to *warm-start* the compute state at the current decision-point and set $\mathrm{T}_0 = \tau$. In Section 6, we empirically study such caching and uncover potential benefits with regard to both performance and meta-reasoning efficiency.

[1] Λ could be level-specific. For simplicity, we assume they are identical.

Meta-policy Value Function. The meta-value function is defined similarly for both levels. Following prior work (Russell & Wefald, 1991; Callaway et al., 2018), the value function of μ is:

$$v^{\mu}(s, g, \tau) = \mathbb{E}_{\mu, p_{\text{meta}}} \left[\sum_{n=0}^{N} c_{\text{meta}}(s, g, \mathrm{T}_n, \Lambda_n) + v^{\pi}(s, g, \mathrm{T}_N) \,\middle|\, \mathrm{T}_0 = \tau \right] \tag{1}$$

and the optimal meta-policy is $\mu^* = \arg\max_{\mu} v^{\mu}(s, g, \tau), \forall s \in \mathcal{S}, g \in \mathcal{G}, \tau \in \mathcal{T}$. Here, $v^{\pi}(s, g, \tau) = \mathbb{E}_{\pi, p}[r(s, g) + \gamma \mathbb{E}_{\pi, \mu, p, p_{\text{meta}}}[v(s', g, \mathrm{T}'_N)]]$ is the value function of the base policy π. Note that the base policy for each level is fixed. Thus, the value of computation performed by μ is maximized if it can assign a compute state T_N to π such that π's return is maximized.

5 HOMER: Hierarchical Meta-Reasoning Agent Architecture

In this section, we introduce our hierarchical meta-reasoning agent, HOMER (**H**ierarchical Rati**O**nal **ME**ta-**R**easoner). We build HOMER by integrating the following two components: 1) HIQL, a feudal-RL agent (Park et al., 2023) and 2) PonderNet, an adaptive-computation-under-computational-constraints algorithm (Banino et al., 2021).

HIQL. HIQL is a modern instantiation of feudal-RL agent architectures (Dayan & Hinton, 1992; Vezhnevets et al., 2017; Nachum et al., 2019) that has been applied successfully across several challenging offline GCRL environments (Park et al., 2025a). HIQL consists of manager and worker policies and a single value function used to train both policies. Each policy level is trained with advantage-weighted regression (AWR) (Peng et al., 2019) and the value function is trained with expectile regression (Kostrikov et al., 2022). In particular, the minimized loss functions are:

$$\begin{aligned}
\mathcal{L}(v) &= \mathbb{E}_{S \sim \rho^{\mathcal{D}}(\mathcal{S}),\, G \sim \rho(\mathcal{G})} \left[l_{\kappa}^2 \left(r(S, G) + \gamma \bar{v}(S', G) - v(S, G) \right) \right] \\
\widetilde{\mathcal{L}}_h(\pi_h) &= -\mathbb{E}_{(S_t, S_{t+k}) \sim \mathcal{D}, G \sim \rho(\mathcal{G})} \left[e^{\alpha(v(S_{t+k}, G) - v(S_t, G))} \log \pi_h(\phi(S_{t+k}) \mid S_t, G) \right] \\
\widetilde{\mathcal{L}}_{\ell}(\pi_{\ell}) &= -\mathbb{E}_{(S_t, A_t, S_{t+1}, S_{t+k}) \sim \mathcal{D}} \left[e^{\alpha(v(S_{t+1}, S_{t+k}) - v(S_t, S_{t+k}))} \log \pi_{\ell}(A_t \mid S_t, \phi(S_{t+k})) \right]
\end{aligned}$$

where the advantage functions are in terms of states instead of actions, $A^{\text{IQL}}(S_t, S_{t+k}, G) = v(S_{t+k}, G) - v(S_t, G)$ and $A^{\text{IQL}}(S_t, S_{t+1}, S_{t+k}) = v(S_{t+1}, S_{t+k}) - v(S_t, S_{t+k})$, $\rho^{\mathcal{D}}$ denotes the state distribution of the dataset, $\bar{v}$ denotes the target value function (Mnih et al., 2013), $l_{\kappa}^2(x) = |\kappa - \mathbf{1}_{\{x:\, x<0\}}(x)| x^2$ denotes the expectile loss with an expectile κ, α is the temperature parameter, and ϕ denotes the subgoal representation.

PonderNet. PonderNet was originally proposed for adaptive computation in supervised learning tasks (Banino et al., 2021). However, due to its general, simple, and fully differentiable design, we integrate PonderNet with our LSTM to adaptively balance policy optimization and computation amount, where computation is modeled as LSTM recurrent steps. PonderNet is an algorithm that parameterizes a halting distribution over the recurrent steps and learns this distribution during the minimization of the policy loss function. In terms of our formulation in Section 4, the halting distribution is the meta-policy μ. Furthermore, p_{meta} is deterministic since the LSTM's gate operations are deterministic and $\mathbf{\Lambda} = \{0, 1\}$, with random variable Λ_n over two actions at recurrence step n: continue ($\Lambda_n = 0$) and halt ($\Lambda_n = 1$). The meta-policy μ is a learnable halting head whose output is $\theta_n(S, G)$, where $\theta_n(S, G) = \mu(\Lambda_n = 1 \mid S, G, \mathrm{T}_n)$ and $1 - \theta_n(S, G) = \mu(\Lambda_n = 0 \mid S, G, \mathrm{T}_n)$, $\forall\, 0 \leq n \leq N_{\text{max}}$ and $\theta_{N_{\text{max}}} = 1$, and the unconditional probability of halting at recurrence step n for inputs (S, G) is $p_n(S, G) = \theta_n(S, G) \prod_{j=0}^{n-1} (1 - \theta_j(S, G))$. PonderNet also includes a KL term between p and a fixed geometric prior distribution, $p_{\text{geom}}(\theta_p)$, with parameter θ_p. Intuitively, since the mean of $p_{\text{geom}}(\theta_p)$ is $1/\theta_p$, θ_p regulates the halting time.

Under this formulation, for each level, let N_{ℓ} (N_h) be the stochastic halting steps, $p_{\ell,\cdot}$ ($p_{h,\cdot}$) be the halting probabilities implicitly dependent on μ_{ℓ} (μ_h), $\theta_{\ell,p}$ ($\theta_{h,p}$) be the prior parameters, and

$\beta \in \mathbb{R}_{\geq 0}$ be the weight on the KL-divergence term. Then the modified AWR policy losses are:

$$\mathcal{L}_h(\pi_h, p_{h,\cdot}) = -\mathbb{E}_{(S_t, S_{t+k}) \sim \mathcal{D},\, G \sim \rho(\mathcal{G})} \Big[\exp\big(\alpha A^{\text{IQL}}(S_t, S_{t+k}, G)\big)\, \mathbb{E}_{N_h \sim p_{h,\cdot}} [\log \pi_h(\phi(S_{t+k}) \mid S_t, G, \mathrm{T}_{N_h})] + \beta\, \mathrm{KL}(p_{h,\cdot} \parallel p_{\text{geom}}(\theta_{h,p})) \Big]$$

$$\mathcal{L}_\ell(\pi_\ell, p_{\ell,\cdot}) = -\mathbb{E}_{(S_t, A_t, S_{t+1}, S_{t+k}) \sim \mathcal{D}} \Big[\exp\big(\alpha A^{\text{IQL}}(S_t, S_{t+1}, S_{t+k})\big)\, \mathbb{E}_{N_\ell \sim p_{\ell,\cdot}} [\log \pi_\ell(A_t \mid S_t, \phi(S_{t+k}), \mathrm{T}_{N_\ell})] + \beta\, \mathrm{KL}(p_{\ell,\cdot} \parallel p_{\text{geom}}(\theta_{\ell,p})) \Big]$$

Intuitively, for a fixed base policy π at (S, G), PonderNet learns an optimal meta-policy μ^* that places π in a compute state T_N such that $\pi(S, G, \mathrm{T}_N)$ is able to maximize the advantage-weighted log-likelihood of an action (or subgoal), while the KL-divergence term keeps compute constrained by regularizing the distribution towards a geometric prior that favors lower recurrence steps. However, since the base policy is also being trained, these losses essentially train both the decision of when to stop computation and what action to output when computation stops.

6 Experiments

In this section, we describe our empirical setup and results.

6.1 Empirical Setup

We first describe our research questions, algorithms, and environments. We include additional details such as algorithm pseudocode and evaluation protocols in Appendix A.2

Research Questions. In our study, we aim to answer the following research questions:

- **RQ1**: Do meta-reasoning agents achieve comparable performance to non-meta-reasoning agents with less compute?
- **RQ2**: How do the meta-reasoning strategies of an HRL agent differ on a per-level basis?
- **RQ3**: Can cached meta-reasoning states be reused to reduce computation without compromising performance?

Baseline Algorithms. The primary comparisons in our experiments are between flat and HRL agents both with and without meta-reasoning. We form our non-meta-reasoning baseline agent by removing PonderNet from HOMER, resulting in an HRL agent that performs a fixed amount of computation per-decision. We use HIQL and GCIQL (Kostrikov et al., 2022; Park et al., 2025a) as our hierarchical and flat RL agents, respectively. We integrate PonderNet with GCIQL to obtain a flat meta-reasoning agent we call FOAMER (**F**lat Rati**OA**l **ME**ta-**R**easoner). In all cases, *only* the policies are equipped with an LSTM, while the value functions are MLPs. We avoid the challenge of goal representation learning by using low-dimensional oracle representations. We use the publicly available implementations for HIQL and GCIQL and implement our own PonderNet + LSTM architecture.

Environments. We evaluate HOMER and the baselines on several offline GCRL environments from the OGBench suite (Park et al., 2025a), where an agent is tasked with reaching specific task goals, and the evaluation metric is average success rate across the tasks. Given that learning from a poor-coverage dataset is orthogonal to our empirical study, we primarily focus on the 'navigate' datasets for the maze tasks and 'play' datasets for the robotic manipulation tasks.

6.2 Empirical Results

We now analyze HOMER's meta-reasoning capabilities across a variety of environments and from different perspectives. We defer additional empirical results to Appendix A.3.

6.2.1 Meta-Reasoning Task Performance vs. Computation

We begin by answering **RQ1** with Figure 2. It shows the success rate of HOMER and FOAMER against their non-meta-reasoning counterparts for varied compute constraints (θ_p) as a function of compute,

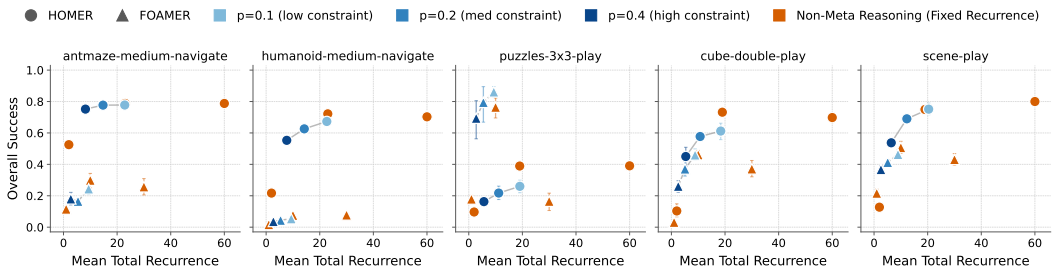

Figure 2: Success vs. total recurrence for both HOMER and FOAMER (with varying compute constraints (θ_p)) with their non-meta-reasoning agent counterparts. The total mean recurrence of the hierarchical agent is the sum of the recurrences at both levels. Higher and to the left is better with the top-left indicating maximum success rate at minimal recurrence. Results are computed over 10 random seeds with the 95% confidence interval (along both axes) computed over different training runs and evaluations.

measured by total number of recurrence steps; that is, higher and further to the left is better. We find that both meta-reasoning agents are able to achieve comparable success rates (blue circles and triangles) with the non-meta-reasoning agent that uses the maximum amount of computation and surpass non-meta-reasoning agents with minimal compute (orange circles and triangle). Simultaneously, HOMER and FOAMER achieve these high success rates with demonstrably less computation than their non-meta-reasoning counterparts. We also find that there does exist a fixed amount of reduced compute which can be allocated to a non-meta-reasoning agent to obtain a success rate comparable to that of a meta-reasoning agent. However, in general, an agent designer will most often lack the prescience needed to set such a hyperparameter appropriately for an arbitrary environment; thus, the adaptive computation afforded by meta-reasoning agents provides a strong motivation for their use.

In puzzles-3x3-play, we found that having computation fixed to the maximum amount actually hurt performance for the flat agent; this result is consistent with prior work suggesting it is possible that high recurrence can still result in poor task performance (Ghugare et al., 2026).

6.2.2 Adaptive Computation in HOMER

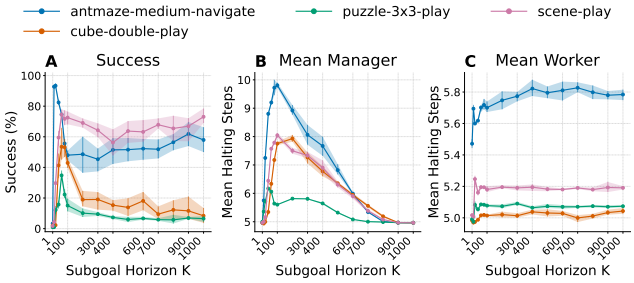

Figure 3: Success rate and mean halting step (manager and worker) as a function of k for the medium-constraint agent ($\theta_p = 0.2$). Each analysis is labeled column-wise: A, B, C. Results are computed over 10 random seeds with the 95% confidence interval computed over different training runs and evaluations.

As the manager and worker agents in feudal-HRL optimize different objectives, we run HOMER for a wide range of k in order to study its impact on meta-reasoning for **RQ2**. In Figure 3, we plot the mean recurrence of the manager and worker as a function of k. Columns B and C show that the manager and worker both adapt their compute usage depending on k: the worker's compute generally increases as k gets larger and then quickly plateaus, while the manager's compute follows an inverted-U-like shape. The observed computational needs of the worker agent are consistent with prior work on flat RL, which suggests that reaching farther goals typically requires more compute (Ghugare et al., 2026).

To explain the computational needs of the manager, we hypothesize that, as k gets larger, the number of k-step subgoals observed in the dataset increases and intuitively requires the manager to meta-reason for longer to determine the appropriate subgoal. Eventually, once k is large enough, the manager just outputs the main goal, which intuitively requires less meta-reasoning since the manager needs to only compute an identity mapping. We test this hypothesis by comparing the manager's compute and the conditional entropy (Shannon, 1948; Cover & Thomas, 2012) of the empirical

k-step distribution induced by the dataset, given the current state and goal — $\mathcal{H}(S_{t+k} \mid S_t, G)$ — both as a function of k on humanoidmaze and antmaze. We present the results in Figure 4 and notably, do empirically observe that the manager's adaptive compute expenditure does closely track the conditional entropy $\mathcal{H}(S_{t+k} \mid S_t, G)$ with high Spearman correlations, lending support to our hypothesis. The observed asymmetry in adaptive computation strategies across the manager and worker suggests that meta-reasoning is level-specific, and that each level may have varied computational demands, with a particular dependence on the subgoal problem horizon, k.

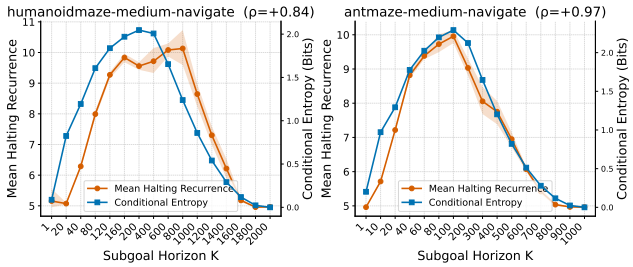

Figure 4: Mean halting recurrence step vs. conditional entropy $\mathcal{H}(S_{t+k} \mid S_t, G)$ with Spearman correlation on humanoidmaze-medium-navigate and antmaze-medium-navigate. Compute results are over 10 random seeds with the 95% confidence interval computed over different training runs and evaluations. Conditional entropy is computed analytically for the fixed navigate dataset (see Appendix A.2).

6.2.3 Warm-Starting Meta-Reasoning Computation

In Section 4, we suggested that warm-starting τ_0 with a previously computed τ could make meta-reasoning more efficient, since that τ may encode partial computations that future meta-reasoning phases can reuse. The next set of experiments aimed at answering **RQ3** assesses this hypothesis, where the internal states are the hidden and the cell states of the LSTM.

Warm-starting Current Input Computation. For any input (s, g), HOMER starts with its internal compute state $\tau_0 = \mathbf{0}$ and produces its final compute state $\tau_N(s, g)$. As an extreme case of warm-starting, we initialize HOMER with $\tau_0 = \tau_N(s, g)$ when processing (s, g) again. If the meta-level states are being utilized in a consistent manner to encode the state of computation, warm-starting should result in a computation terminating within fewer steps; this effect is indeed what we observe empirically in Figure 5. Column A of Figure 5 shows that warm-starting always terminated meta-reasoning sooner

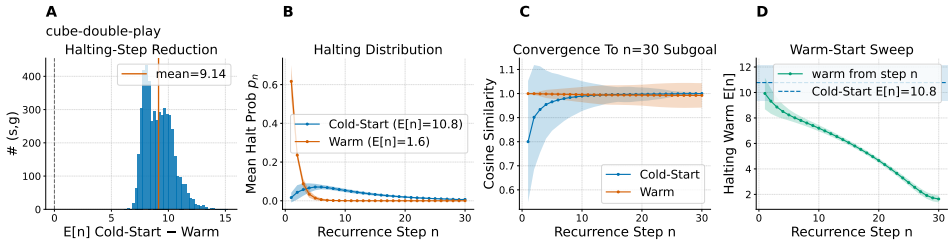

Figure 5: Statistics when warm-starting HOMER's low-constraint manager ($\theta_p = 0.1$) on cube-double-play. $\mathbb{E}[n]$ denotes the mean halting recurrence step. Each analysis is labeled column-wise: A, B, C, D. Results are from sampling 5000 (s, g) pairs, with the mean and standard deviation shown.

(HOMER halted ≈ 9 recurrent steps earlier than a cold-start agent across all sampled (s, g) pairs). In column B, we show the halting distribution over recurrent steps, and find that warm-starting drastically skews the halting distribution towards lower recurrence steps. In column C, we evaluate how quickly the warm-started manager's output converges to the subgoal emitted by the cold-started manager at τ_N, as measured by cosine similarity. We find that the warm-started network converges to it at $n = 1$, whereas the cold-started network takes the full $n = 30$ steps, as anticipated. Finally, in column D, we sweep over warm-starts $\tau_0 = \tau_n$ for $n \in 1, \ldots, 30$. We observe that the halting time decreases almost monotonically with n, since τ_n encodes partially completed computations.

Warm-starting Successive Decisions. In this set of experiments, we explore whether a cold-start trained meta-reasoning manager can increase its success rate and decrease its total computation during deployment through warm-starting. We focus on the low-constraint manager ($\theta_p = 0.1$) since it is likely to benefit the most from warm-starting. Let $\tau_{t,n}$ denote the internal state at the manager's decision time-step t and recurrence step n, and let N denote the stochastic halting time.

We evaluate three initialization methods: 1) *cold-start*, $\tau_{t,0} = \mathbf{0}$; 2) *warm*, $\tau_{t,0} = \tau_{t-1,N}$, which warm-starts to the τ from the previous decision; and 3) *reset-i*, which warm-starts but resets $\tau_{t,0} = \mathbf{0}$ every i manager decision time-steps.



Figure 6: Success (A, right is better) vs. average manager recurrence per decision (B, left is better) vs. per episode (C, left is better) with warm-starting HOMER's low-constraint manager ($\theta_p = 0.1$) on antmaze-medium-navigate (top) and puzzle-3x3-play (bottom). $\star$ denotes improvement over cold-start. HOMER uses $k = 50$ but it produces a subgoal every step. Results are averaged over 10 random seeds, with 95% confidence intervals computed from deploying a single trained agent over different seeded evaluations.

Figure 6 shows that warm-starting can reduce computation while maintaining or even improving success (variants marked $\star$). However, pure warm-starting can yield zero success (for example, in antmaze) despite reducing compute, confirming that stale computations can harm future decisions. Overall, these results suggest that computationally-bounded meta-reasoning agents can cache prior computations to improve performance and compute efficiency.

7 Limitations and Future Work

While our work takes a step towards understanding model-free iterative computation for hierarchical meta-reasoning agents, several limitations motivate future work. First, we assume that the environment is static during meta-reasoning. A non-static environment is more realistic and could present more pressure on the agent to meta-reason efficiently. Second, the observed correlation between the manager's compute usage and the conditional entropy of the k-step future state distribution suggests that mechanisms for entropy reduction could bring fruitful reductions in compute requirements; one natural choice for doing so is through performance-preserving state abstractions over goals themselves (Li et al., 2006; Abel et al., 2016; 2018; 2019) as well as the specific, multi-step paths needed to reach them (Konidaris, 2019; Abel et al., 2020). Third, while this work focuses on offline GCRL, extending our setup to the full online RL setting would raise other interesting questions around the delicate interplay of meta-reasoning and exploration, at each level in the hierarchy. Finally, our empirical study proceeded in the cleanest and simplest feudal-RL agent hierarchy, consisting of only one manager and one worker; future work may naturally explore similar studies on the intricacies of meta-reasoning with more complex agent hierarchies (Levy et al., 2019), particularly as they pertain to teams of decision-making agents (Pappu et al., 2026; Mieczkowski et al., 2026).

8 Conclusion

In this work, we took the first step towards studying adaptive computation across levels within a hierarchical decision-making agent by formalizing hierarchical rational meta-reasoning and introducing HOMER, a model-free RNN-based hierarchical meta-reasoning agent for offline GCRL. Our work has several critical implications for computationally-bounded agents (Arumugam et al., 2024; Javed & Sutton, 2024). First, as modern decision-making agents proliferate into vastly more complex environments, agent designers should endow agents with the requisite meta-reasoning capabilities to avoid insufficient or even wasteful deployment of computational resources. Second, the observed asymmetry in adaptive computation across the hierarchy suggests that, as systems grow more complex and operate across varying temporal horizons, problem horizons must be chosen carefully to balance computational efficiency and performance. Third, compute-bounded agents should cache prior computations whenever possible in order to bootstrap off previous decisions and enjoy savings on compute resources. Overall, our work advances the goal of building meta-reasoning agents that balance the computation required for a problem against its utility for decision-making.

AI USE STATEMENT

In this work, we used generative AI tools for plotting scripts, training job scripts, code comments and documentation, reviewing implementations of algorithms, and tightening sentence-level writing. We have not used generative AI tools for literature review, formulating mathematical claims, providing critical ingredients for mathematical claims, refining hypotheses, designing or providing feedback on research methodology or experiments, and interpreting results; and generating synthetic datasets and assistance with translation are not applicable to this work. Additionally, we used generative AI tools to generate Figure 1. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Our work is largely focused on fundamental RL questions applied to benchmark problems widely used in the RL research community. We do acknowledge that RL is a dual-use technology that can potentially have both good and bad broader societal consequences depending on how the agent makes decisions.

REPRODUCIBILITY STATEMENT

We make the following efforts to make our work reproducible. We include: 1) the source code as part of our submission; 2) snippets of code used in our experiments in Appendix A.1; 3) the training details, evaluation protocols, and dataset details in Appendix A.2.

REFERENCES

David Abel, David Hershkowitz, and Michael Littman. Near Optimal Behavior via Approximate State Abstraction. In *International Conference on Machine Learning*, pp. 2915–2923. PMLR, 2016.

David Abel, Dilip Arumugam, Lucas Lehnert, and Michael Littman. State Abstractions for Lifelong Reinforcement Learning. In *International Conference on Machine Learning*, pp. 10–19. PMLR, 2018.

David Abel, Dilip Arumugam, Kavosh Asadi, Yuu Jinnai, Michael L Littman, and Lawson LS Wong. State Abstraction as Compression in Apprenticeship Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3134–3142, 2019.

David Abel, Nate Umbanhowar, Khimya Khetarpal, Dilip Arumugam, Doina Precup, and Michael Littman. Value Preserving State-Action Abstractions. In *International Conference on Artificial Intelligence and Statistics*, pp. 1639–1650. PMLR, 2020.

Hongjoon Ahn, Heewoong Choi, Jisu Han, and Taesup Moon. Option-aware temporally abstracted value for offline goal-conditioned reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Dilip Arumugam, Mark K Ho, Noah D Goodman, and Benjamin Van Roy. Bayesian Reinforcement Learning with Limited Cognitive Load. *Open Mind*, 8:395–438, 2024.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pp. 1726–1734, 2017. URL https://ojs.aaai.org/index.php/AAAI/article/view/10916.

Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to ponder, 2021. URL https://arxiv.org/abs/2107.05407.

Andrew G Barto and Sridhar Mahadevan. Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

Richard Bellman. A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 6: 679–684, 1957.

Matthew Budd, Bruno Lacerda, and Nick Hawes. Stop! planner time: Metareasoning for probabilistic planning using learned performance profiles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 20120–20128, May 2024. doi: 10.1609/aaai.v38i18.29983. URL https://ojs.aaai.org/index.php/AAAI/article/view/29983.

Thomas Bush, Stephen Chung, Usman Anwar, Adrià Garriga-Alonso, and David Krueger. Interpreting emergent planning in model-free reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=DzGe40glxs.

Frederick Callaway, Sayan Gul, Paul M. Krueger, Thomas L. Griffiths, and Falk Lieder. Learning to select computations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Stephen Chung, Ivan Anokhin, and David Krueger. Thinker: Learning to plan and act. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://openreview.net/forum?id=mumEBl0arj.

Sergei Chuprov, Richard D. Lange, Leon Reznik, Paulo Shakarian, Raman Zatsarenko, and Dmitrii Korobeinikov. Position: Artificial intelligence needs meta intelligence – the case for metacognitive AI. *arXiv preprint arXiv:2605.15567*, 2026.

Thomas M Cover and Joy A Thomas. *Elements of Information Theory*. John Wiley & Sons, 2012.

Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. In *Neural Information Processing Systems*, 1992. URL https://api.semanticscholar.org/CorpusID:2801572.

C. Nicolò De Sabbata, Theodore R. Sumers, Badr AlKhamissi, Antoine Bosselut, and Thomas L. Griffiths. Rational metareasoning for large language models. *arXiv preprint arXiv:2410.05563*, 2024.

Thomas G Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

Manfred Eppe, Christian Gumbsch, Matthias Kerzel, and et al. Intelligent problem-solving as integrated hierarchical reinforcement learning. *Nature Machine Intelligence*, 4:11–20, 2022. doi: 10.1038/s42256-021-00431-7.

Samuel J. Gershman, Eric Horvitz, and Joshua B. Tenenbaum. Computational rationality: A converging paradigm for intelligence in brains, minds, and machines. *Science*, 349(6245):273–278, 2015. doi: 10.1126/science.aac6076.

Raj Ghugare, Michał Bortkiewicz, Alicja Ziarko, and Benjamin Eysenbach. On the role of computation in reinforcement learning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=HKzMnBX5B2.

Irving John Good. Rational decisions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 14(1):107–114, January 1952. doi: 10.1111/j.2517-6161.1952.tb00104.x.

Alex Graves. Adaptive computation time for recurrent neural networks, 2017. URL https://arxiv.org/abs/1603.08983.

Thomas L. Griffiths. Understanding human intelligence through human limitations. *Trends in Cognitive Sciences*, 24(11):873–883, 2020. doi: 10.1016/j.tics.2020.08.005.

Thomas L Griffiths, Frederick Callaway, Michael B Chang, Erin Grant, Pieter M Krueger, and Falk Lieder. Doing more with less: Meta-reasoning and meta-learning in humans and machines. *Current Opinion in Behavioral Sciences*, 29:24–30, 2019.

Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sébastien Racaniére, Theophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, Greg Wayne, David Silver, and Timothy P. Lillicrap. An investigation of model-free planning. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2464–2473. PMLR, 6–9 Jun 2019. URL https://proceedings.mlr.press/v97/guez19a.html.

Jessica B. Hamrick, Andrew J. Ballard, Razvan Pascanu, Oriol Vinyals, Nicolas Heess, and Peter W. Battaglia. Metacontrol for adaptive imagination-based optimization. In *International Conference on Learning Representations (ICLR)*, 2017. URL https://arxiv.org/abs/1705.02670.

Josiah P. Hanna and Nicholas E. Corrado. When can model-free reinforcement learning be enough for thinking? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=elfvik1mkg.

Nicholas Hay, Stuart J. Russell, Solomon Eyal Shimony, and David Tolpin. Selecting computations: Theory and applications. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI 2012)*, pp. 334–343. AUAI Press, 2012.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

Eric Horvitz, Gregory Cooper, and David Heckerman. Reflection and action under scarce resources: Theoretical principles and empirical study. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pp. 1121–1127, Detroit, MI, USA, 1989.

Jinu Hyeon, Woobin Park, Hongjoon Ahn, and Taesup Moon. Action-sufficient goal representations. In *International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2601.22496.

Khurram Javed and Richard S. Sutton. The big world hypothesis and its ramifications for artificial intelligence. In *Finding the Frame: An RLC Workshop for Examining Conceptual Frameworks*, 2024. URL https://openreview.net/forum?id=Sv7DazuCn8.

Leslie Pack Kaelbling. Learning to achieve goals. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1094–1099, 1993. URL https://www.semanticscholar.org/paper/Learning-to-Achieve-Goals-Kaelbling/6df43f70f383007a946448122b75918e3a9d6682.

Martin Klissarov, Akhil Bagaria, Zeyu Luo, George Konidaris, Doina Precup, and Marlos C. Machado. Discovering temporal structure: An overview of hierarchical reinforcement learning. *arXiv preprint arXiv:2506.14045*, 2025.

George Konidaris. On the Necessity of Abstraction. *Current Opinion in Behavioral Sciences*, 29: 1–7, 2019.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline Reinforcement Learning with Implicit Q-Learning. In *International Conference on Learning Representations*, 2022.

Adam Labiosa and Josiah P. Hanna. Learning to act while thinking. In *Proceedings of the RL in Big Worlds Workshop at RLC*, 2026a.

Adam Labiosa and Josiah P. Hanna. When to plan: Learning to select between reactive control and deliberative planning. In *Reinforcement Learning Conference (RLC)*, 2026b. URL https://arxiv.org/abs/2607.16421.

Andrew Levy, Robert Platt, and Kate Saenko. Hierarchical Reinforcement Learning with Hindsight. In *International Conference on Learning Representations*, 2019.

Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstraction for MDPs. In *Proceedings of the International Symposium on Artificial Intelligence and Mathematics (ISAIM / AI&M)*, 2006.

Christopher H. Lin, Andrey Kolobov, Ece Kamar, and Eric Horvitz. Metareasoning for planning under uncertainty. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pp. 1601–1609. AAAI Press, 2015. URL http://ijcai.org.

Shuaijun Liu, Feiyang You, Xingwei Chen, and Ningxin Su. When replanning becomes the bottleneck: Budgeted replanning for embodied agents. In *International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2608.01428.

Elizabeth Mieczkowski, Alexander Ku, Tiwalayo Eisape, Dilip Arumugam, John Matters, Katherine M Collins, Ilia Sucholutsky, and Thomas L Griffiths. Improving the Efficiency of Language Agent Teams with Adaptive Task Graphs. *arXiv preprint arXiv:2605.06320*, 2026.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Aneesh Muppidi, Firas Darwish, Dylan Cope, João F. Henriques, and Jakob Nicolaus Foerster. Finding the time to think: Learning planning budgets in real-time RL, 2026. URL https://arxiv.org/abs/2606.26463.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=H1emus0qF7.

Mozhgan Navardi, Shang Gao, Mikolaj Walczak, Fernando Camacho, and Tinoosh Mohsenin. Metareasoning for edge-cloud collaborative LLM planning for efficient autonomous navigation. *ACM Trans. Embed. Comput. Syst.*, 25(3), may 2026. ISSN 1539-9087. doi: 10.1145/3765750. URL https://doi.org.

Aneesh Pappu, Batu El, Hancheng Cao, Carmelo di Nolfo, Yanchao Sun, Meng Cao, and James Zou. Multi-Agent Teams Hold Experts Back. In *Forty-third International Conference on Machine Learning*, 2026.

Seohong Park, Dibya Ghosh, Benjamin Eysenbach, and Sergey Levine. HIQL: Offline goal-conditioned RL with latent states as actions. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=cLQCCtVDuW.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. OGBench: Benchmarking offline goal-conditioned RL. In *International Conference on Learning Representations (ICLR)*, 2025a. URL https://openreview.net/forum?id=M992mjgKzI.

Seohong Park, Kevin Frans, Deepinder Mann, Benjamin Eysenbach, Aviral Kumar, and Sergey Levine. Horizon reduction makes RL scalable. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025b.

Ronald Parr and Stuart Russell. Reinforcement Learning with Hierarchies of Machines. *Advances in Neural Information Processing Systems*, 10, 1997.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning. *arXiv preprint arXiv:1910.00177*, 2019.

Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

Stuart Russell and Eric Wefald. Principles of metareasoning. *Artificial Intelligence*, 49(1–3):361–395, 1991. doi: 10.1016/0004-3702(91)90015-C. URL https://www.sciencedirect.com/science/article/pii/000437029190015C.

Julian Schrittwieser et al. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588:604–609, 2019. URL https://arxiv.org/abs/1911.08265.

Claude E Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404.

Herbert A. Simon. Theories of bounded rationality. In C. B. McGuire and Roy Radner (eds.), *Decision and Organization*, pp. 161–176. North-Holland Publishing Company, Amsterdam, 1972.

Alec Solway, Carlos Diuk, Natalia Córdova, Debbie Yee, Andrew G Barto, Yael Niv, and Matthew M Botvinick. Optimal behavioral hierarchy. *PLoS computational biology*, 10(8):e1003779, 2014. doi: 10.1371/journal.pcbi.1003779.

Yoonchang Sung, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Learning when to quit: Meta-reasoning for motion planning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3432–3439. IEEE, 2021.

Yoonchang Sung, Shahaf S. Shperberg, Qi Wang, and Peter Stone. Effort allocation for deadline-aware task and motion planning: A metareasoning approach. *CoRR*, abs/2410.05828, 2024. URL https://arxiv.org/abs/2410.05828.

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

Justin Svegliato, Connor Basich, Sandhya Saisubramanian, and Shlomo Zilberstein. Metareasoning for safe decision making in autonomous systems. In *2022 International Conference on Robotics and Automation (ICRA)*, pp. 11073–11079. IEEE Press, 2022. doi: 10.1109/ICRA46639.2022.9811887. URL https://doi.org/10.1109/ICRA46639.2022.9811887.

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 2154–2162, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Mohammad Taufeeque, Philip Quirke, Maximilian Li, Chris Cundy, Aaron David Tucker, Adam Gleave, and Adrià Garriga-Alonso. Planning in a recurrent neural network that plays Sokoban. *arXiv preprint arXiv:2407.15421*, 2024.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. FeUdal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3540–3549. PMLR, 2017. URL https://proceedings.mlr.press/v70/vezhnevets17a.html.

Kevin A. Wang, Jerry Xia, Stephen Chung, and Amy Greenwald. Dynamic thinker: Optimizing decision-time planning with costly compute. In *The Seventeenth Workshop on Adaptive and Learning Agents (ALA)*, 2025.

Hanqi Yan, Linhai Zhang, Jiazheng Li, Zhenyi Shen, and Yang He. Position: LLMs need a bayesian meta-reasoning framework for more robust and generalizable reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 82360–82383. PMLR, 2025. URL https://proceedings.mlr.press/v267/yan25g.html.

A APPENDIX

In the Appendix, we include more details regarding training details, evaluations, hyperparameters, and code.

A.1 LSTM AND PONDERNET CODE

We present snippets of our LSTM and PonderNet implementation below.

```python
import jax
import jax.numpy as jnp


def pondernet_halt(halt_logits: jnp.ndarray, prior_halt_prob: float):
    """Turn N per-step halting logits into a proper halting distribution over a FINITE budget.

    halt_logits: [N, ...]  one logit per computation step (from a small head on that step's state).
    prior_halt_prob: geometric prior's per-step halting probability (mean steps ~= 1 / prior_halt_prob).

    Returns:
      p          [N, ...]  halting distribution, sums to 1 over the N axis.
      exp_steps  [...]     expected number of steps, E[n] = sum_n n * p_n.
      kl         [...]     KL(p || geometric prior) -- the compute-cost regularizer.
    """
    n_steps = halt_logits.shape[0]
    lam = jax.nn.sigmoid(halt_logits)
    lam = lam.at[-1].set(1.0)

    # p_n = lam_n * prod_{m<n} (1 - lam_m)
    survive = jnp.cumprod(
        jnp.concatenate([jnp.ones_like(lam[:1]), 1.0 - lam[:-1]], 0),
        axis=0
    )
    p = lam * survive

    steps = jnp.arange(1, n_steps + 1).reshape(
        (-1,) + (1,) * (p.ndim - 1)
    )
    exp_steps = jnp.sum(p * steps, axis=0)

    prior = prior_halt_prob * (1.0 - prior_halt_prob) ** jnp.arange(n_steps)
    prior = (prior / prior.sum()).reshape(
        (-1,) + (1,) * (p.ndim - 1)
    )
    kl = jnp.sum(
        p * (jnp.log(p + 1e-8) - jnp.log(prior + 1e-8)),
        axis=0
    )
    return p, exp_steps, kl


def pondernet_loss(
    step_losses: jnp.ndarray,
    p: jnp.ndarray,
    kl: jnp.ndarray,
    beta: float
```

```
):
    """The PonderNet training objective: expected task loss under the halting distribution,
    plus a compute-cost penalty. step_losses: [N, ...] per-step task loss (e.g. -log pi(a|s_n))."""
    expected_loss = jnp.sum(p * step_losses, axis=0)
    return expected_loss + beta * kl
```

Listing 1: PonderNet adaptive-halting mechanism (Banino et al., 2021).

```
from typing import Sequence
import flax.linen as nn
import jax.numpy as jnp
from pondernet import pondernet_halt   # Listing 1


class LSTM(nn.Module):
    """D-deep, N-repeat grid of LSTM-style cells (Guez et al. 2019), adapted so the number of
    repeats N is *adaptive* per input via PonderNet. Cells reset to zero on every call
    (stateless across environment steps).

    Each cell at (depth d, repeat n) sees:
      x        : encoded input, fed to every cell (obs-skip)
      h_below  : output of the layer BELOW in this repeat (bottom layer wraps to the top layer's
                 previous-repeat output -- a "top-down" connection)
      h_self   : this layer's own output from the PREVIOUS repeat
    """
    hidden_dim: int
    depth: int
    num_rec_steps: int
    prior_halt_prob: float = 0.1

    @nn.compact
    def __call__(self, x):
        x = nn.LayerNorm()(nn.Dense(self.hidden_dim)(x))
        cells = [
            nn.Dense(4 * self.hidden_dim, name=f'cell_{d}')
            for d in range(self.depth)
        ]
        halt_head = nn.Dense(
            1,
            bias_init=nn.initializers.constant(-1.0),
            name='halt'
        )

        c = [
            jnp.zeros((*x.shape[:-1], self.hidden_dim))
            for _ in range(self.depth)
        ]
        h = [
            jnp.zeros((*x.shape[:-1], self.hidden_dim))
            for _ in range(self.depth)
        ]

        readouts, halt_logits = [], []

        for _ in range(self.num_rec_steps):
            h_below = h[-1]

            for d in range(self.depth):
```

```
                    gate_in = jnp.concatenate(
                        [x, h_below, h[d]], axis=-1
                    )
                    f, i, o, g = jnp.split(
                        cells[d](gate_in), 4, axis=-1
                    )

                    f = nn.sigmoid(f)
                    i = nn.sigmoid(i)
                    o = nn.sigmoid(o)
                    g = jnp.tanh(g)

                    c[d] = f * c[d] + i * g
                    h[d] = o * jnp.tanh(c[d])
                    h_below = h[d]

                readouts.append(
                    jnp.concatenate([x, h[-1]], axis=-1)
                )
                halt_logits.append(
                    halt_head(h[-1])[..., 0]
                )

            readouts = jnp.stack(readouts, axis=0)

            p, exp_steps, kl = pondernet_halt(
                jnp.stack(halt_logits, 0),
                self.prior_halt_prob
            )

            return readouts, p, exp_steps, kl


    def act(readouts, p):
        """Inference: read out at the step nearest the expected halting
        ↪  count E[n]."""
        n_star = jnp.clip(
            jnp.round(
                (jnp.arange(1, p.shape[0] + 1) * p).sum(0)
            ) - 1,
            0,
            p.shape[0] - 1
        ).astype(jnp.int32)

        return jnp.take_along_axis(
            readouts,
            n_star[None, ..., None],
            axis=0
        )[0]
```

Listing 2: LSTM: a flat-vector LSTM with an adaptive step count from PonderNet.

A.2 Experimental Details

We include additional experimental details in this section.

A.2.1 LSTM Architecture

While we use the standard LSTM architecture, we make a few changes from the conventional implementation:

1. **Input structure.** A standard LSTM threads recurrent state along a *sequence* of distinct external inputs, ours instead applies repeated recurrent computation to a *single* input, using the recurrence as internal "thinking" steps rather than steps over time.
2. **Top-down connections across depth.** In a stacked LSTM, layer d receives layer $d-1$'s *current*-step output only. In ours, the bottom layer of each repeat instead receives the *top* layer's output from the *previous* repeat, creating a top-down feedback path absent from standard stacked LSTM. This idea was taken from DRC (Guez et al., 2019).
3. **Observation skip-connections.** Ours concatenates the input x into every cell's gate input at every depth and repeat. A standard LSTM feeds the current sequence element only at the bottom layer. This idea was also taken from DRC (Guez et al., 2019).
4. **Statefulness.** A standard LSTM's cell and hidden states persist across the sequence being processed. Ours re-initializes all cell and hidden states to zero on every call, since it performs single-input, stateless iterative computation rather than carrying memory across time.

These modifications were not necessary for our findings related to compute but they helped improve the agent performance for both meta-reasoning and non-meta-reasoning agents.

A.2.2 Training Details

RL Algorithms. We use the publicly available implementations of HIQL and GCIQL found at: `https://github.com/seohongpark/ogbench`.

Infrastructure. We run our experiments on an internal university cluster consisting of the following GPUs: Tesla P100-PCIE-16GB, NVIDIA GeForce RTX 2080 Ti, NVIDIA A100-SXM4-40GB, NVIDIA A100-SXM4-80GB, NVIDIA L40, NVIDIA L40S, NVIDIA H100 80GB HBM3, and NVIDIA H200, where we used at least 20GB of memory. Our experiments ran in around 4 hours per training run, which is similar to the time reported by Park et al. (2025a).

Hyperparameters. We use the standard hyperparameters used in the OGBench suite (Park et al., 2025a).

Hyperparameter	Value
Learning rate	0.0003
Optimizer	Adam
# gradient steps	500000
Minibatch size	1024
MLP dimensions	$(512, 512, 512)$
LSTM depth	1
Nonlinearity	GELU
Target smoothing coefficient	0.005
Discount factor γ	0.995 (antmaze, humanoid), 0.99 (others)
GCIQL expectile κ	0.9
HIQL expectile κ	0.7
HIQL subgoal step k	100 (humanoidmaze), 50 (others)
Policy $(p^{\mathcal{D}}_{\text{cur}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{rand}})$ ratio for $p^{\mathcal{D}}_{\text{mixed}}$	$(0, 1, 0, 0)$
Value $(p^{\mathcal{D}}_{\text{cur}}, p^{\mathcal{D}}_{\text{traj}}, p^{\mathcal{D}}_{\text{geom}}, p^{\mathcal{D}}_{\text{rand}})$ ratio for $p^{\mathcal{D}}_{\text{mixed}}$	$(0.2, 0, 0.5, 0.3)$

Table 1: Common hyperparameters.

Datasets. We use the 'navigate' and 'play' datasets from the OGBench suite (Park et al., 2025a). These consist of 1M transitions. The 'navigate' dataset is generated by a noisy expert policy that navigates the maze by repeatedly reaching randomly sampled goals. The 'play' dataset is generated by a scripted policy that repeatedly performs different tasks depending on the environment (scene-v0, cube-double-v0, puzzle-3x3-v0).

Evaluation Protocol. We follow the evaluation protocol in OGBench (Park et al., 2025a). We periodically evaluate the performance (goal success rate in percentage) of each agent on each test-time goal by deploying the agent for 20 rollouts every 100K training steps, and report the average success across all tasks. Success is determined by averaging the success over the last three rollouts. Since agents are trained for 500K training steps, we average the success rate at steps 300K, 400K, and 500K training steps. Therefore, success is measured over 300 rollouts (3 evaluation epochs $\times$ 5 test-time goals $\times$ 20 rollouts).

Caching Representations in LSTMs. Our LSTM of depth 1 consisted of both the hidden and cell state. When we used caching in our experiments, we cached both the hidden and cell state of the LSTM.

Conditional Entropy Calculation. In Section 6.2.2, we analytically computed the conditional entropy using a count-based method for humanoidmaze and antmaze since their maze-like structure made them amenable to such calculation. We now describe the procedure we used to calculate it.

Given a dataset of trajectories, we use every valid current state index t and sample M future goal indices t_g from the same trajectory by uniformly sampling from its remaining portion. The same sampled start–goal examples are reused across all subgoal horizons k. For each horizon k, the target index is

$$t_{\text{target}} = \min(t + k, t_g).$$

We discretize the start, goal, and target positions into 4×4 maze cell units as done in OGBench. To stick to the same notation, we write these cells as S_t, G, and S_{t+k}, respectively, where S_{t+k} denotes the clipped target at t_{target}. Let $N^{(k)}_{S_t,G,S_{t+k}}$ denote the number of examples with start cell S_t, goal cell G, and target cell S_{t+k} at horizon k. The empirical conditional distribution is

$$\hat{p}_k(S_{t+k} \mid S_t, G) = \frac{N^{(k)}_{S_t,G,S_{t+k}}}{N_{S_t,G}}, \qquad N_{S_t,G} = \sum_{S_{t+k}} N^{(k)}_{S_t,G,S_{t+k}}.$$

We estimate conditional entropy in bits as

$$\hat{H}_k(S_{t+k} \mid S_t, G) = -\sum_{S_t,G:\, N_{S_t,G}>0} \frac{N_{S_t,G}}{N} \sum_{S_{t+k}:\, N^{(k)}_{S_t,G,S_{t+k}}>0} \hat{p}_k(S_{t+k} \mid S_t, G) \log_2 \hat{p}_k(S_{t+k} \mid S_t, G),$$

where $N = \sum_{S_t,G} N_{S_t,G}$ is the total number of sampled examples. The inner summation is the familiar entropy term. The outer summation and weighting is for taking a weighted average across all sampled goals. In our experiments, we sampled $M = 50$ goals, and found that varying this number did not affect results.

A.3 Additional Results

We include additional completed results in this section.

A.3.1 Adaptive Computation

In this section, we include the humanoid result in Figure 7, which was excluded in Figure 3.

A.3.2 Warm-Starting Meta-Reasoning Computation

In this section, we include the additional warm-starting experiments that were conducted in the other environments. They follow the same setup as in Section 6.2.3.

A.3.3 Visualizing Adaptive Compute

In this section, we visualize how the compute usage of the worker and manager adapt depending on the (s, g) inputs for a fixed k. We do so in the maze environments, where the (s, g) visualization is more intuitive.

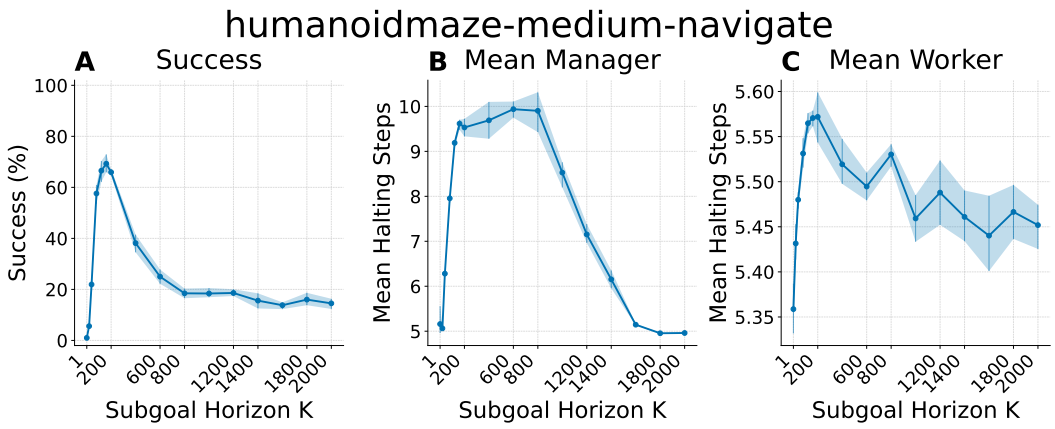

Figure 7: Success rate and mean halting step (manager and worker) as a function of k for the medium-constraint agent ($\theta_p = 0.2$). Each analysis is labeled column-wise: A, B, C. Results are computed over 10 random seeds with the 95% confidence interval computed over different training runs and evaluations.

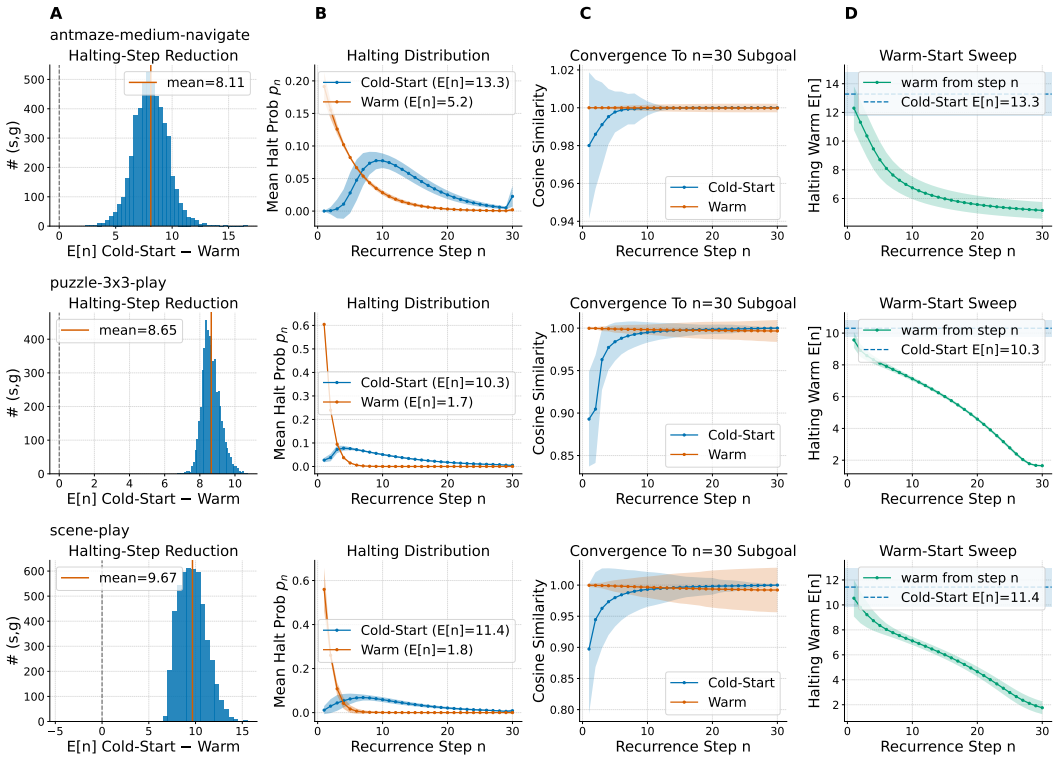

Figure 8: Statistics when warm-starting HOMER's low-constraint manager ($\theta_p = 0.1$) on antmaze-medium-navigate, puzzle-3x3-play, and scene-play-v0 (humanoidmaze is in Figure 9). $\mathbb{E}[n]$ denotes the mean halting recurrence step. Each analysis is labeled column-wise: A, B, C, D. Results are from sampling 5000 (s, g) pairs, with the mean and standard deviation shown.

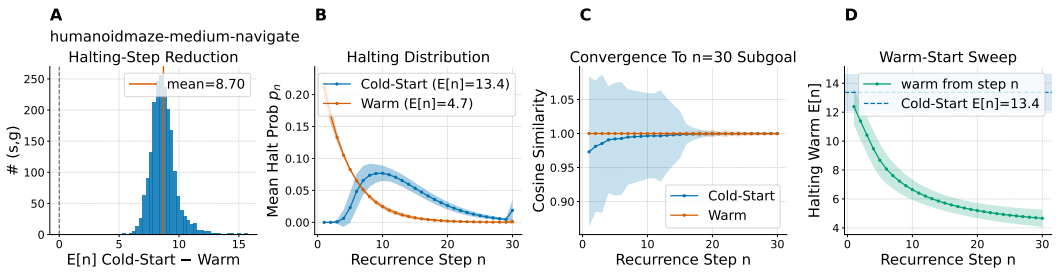

Figure 9: Statistics when warm-starting HOMER's low-constraint ($\theta_p = 0.1$) on humanoidmaze-medium-navigate. $\mathbb{E}[n]$ denotes the mean halting recurrence step. Each analysis is labeled column-wise: A, B, C, D. Results are from sampling 5000 (s, g) pairs, with the mean and standard deviation shown.

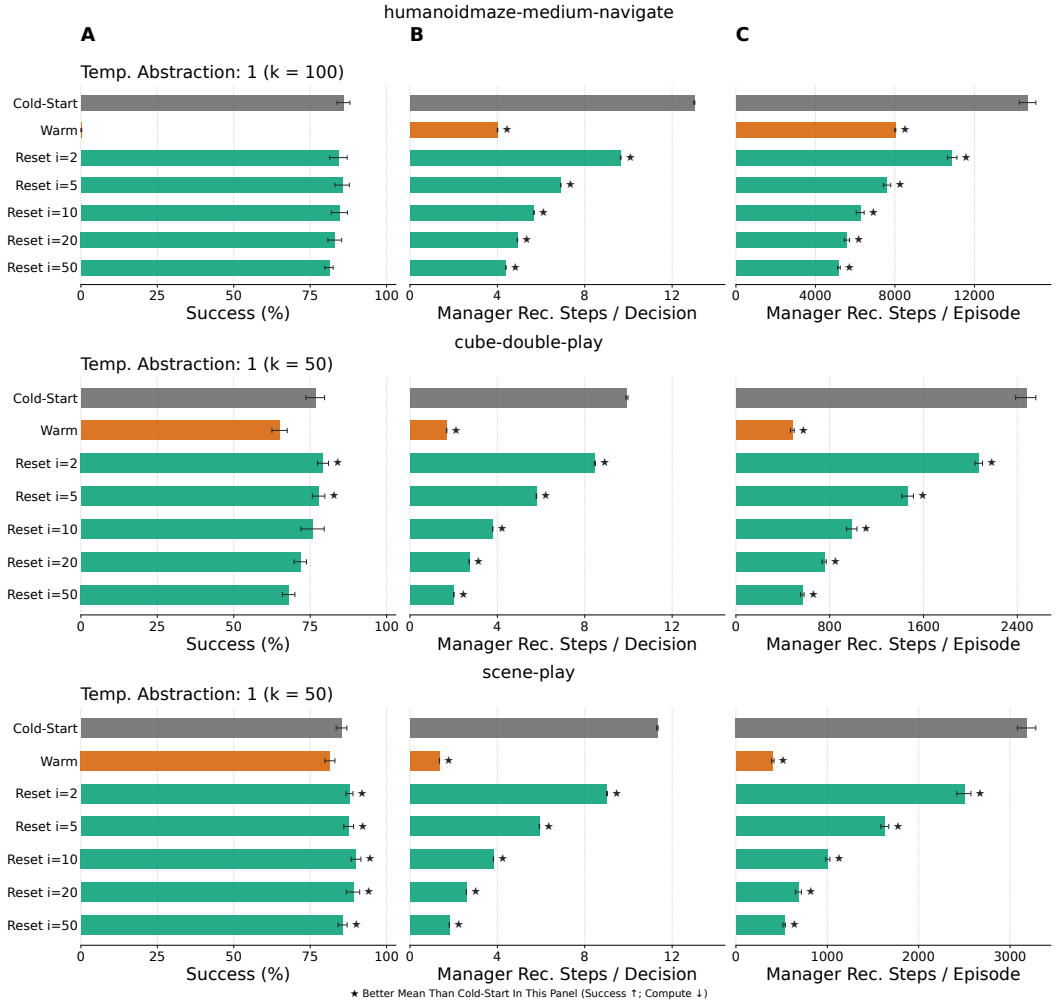

Figure 10: Success (column A, right is better) vs. average manager recurrent steps per decision (column B, left is better) vs. per episode (column C, left is better) with warm-starting HOMER's low-constraint manager ($\theta_p = 0.1$) on antmaze-medium-navigate (top), cube-double-play (middle), and scene-play (bottom). HOMER produces a subgoal k-steps away but does so every environment step. Results are averaged over 10 random seeds, with 95% confidence intervals computed from deploying a single trained agent over different seeded evaluations.

We present the following analysis, which we visualize in Figure 11 and Figure 12. For a fixed subgoal horizon k, we sample (s, g) pairs from the training data, query the trained manager's and worker's expected number of recurrent steps, $\mathbb{E}[N \mid s, g]$, and record these values. We then visualize the highest- and lowest-$\mathbb{E}[N]$ pairs, which concretely tells us what kinds of (s, g) configurations demand the most vs. least meta-reasoning.

From the visuals, we can see that for a fixed k, the manager also adapts its compute based on the distance between (s, g). The manager uses more recurrence for far goals, whereas it uses less recurrence for nearby goals. The worker is similar, but weakly so. This finding about the worker aligns with the results in Section 6.2.2, where we varied the subgoal problem horizon k. In those results, we saw that the worker's computation needs were based on distance to goals but plateaued very quickly. Additionally, the actual compute used (in terms of the magnitude of the recurrence) was only marginally different as we varied the subgoal horizon, k).

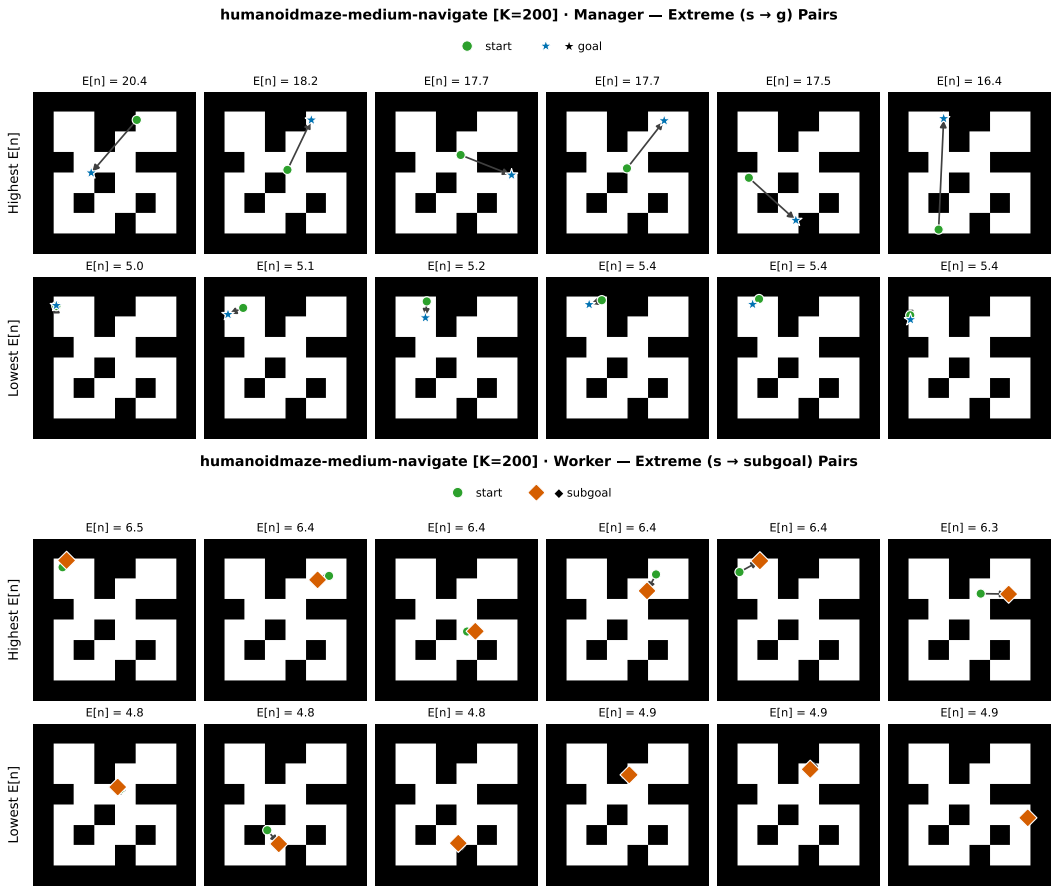

Figure 11: Extreme $(s \to g)$ pairs by manager recurrence (top) and worker recurrence (bottom) on humanoid-medium-navigate. Each row shows the top-6 highest- and bottom-6 lowest-recurrence pairs sampled from the training distribution, drawn as start → target arrows on the environment; recurrence is averaged over 10 seeds.

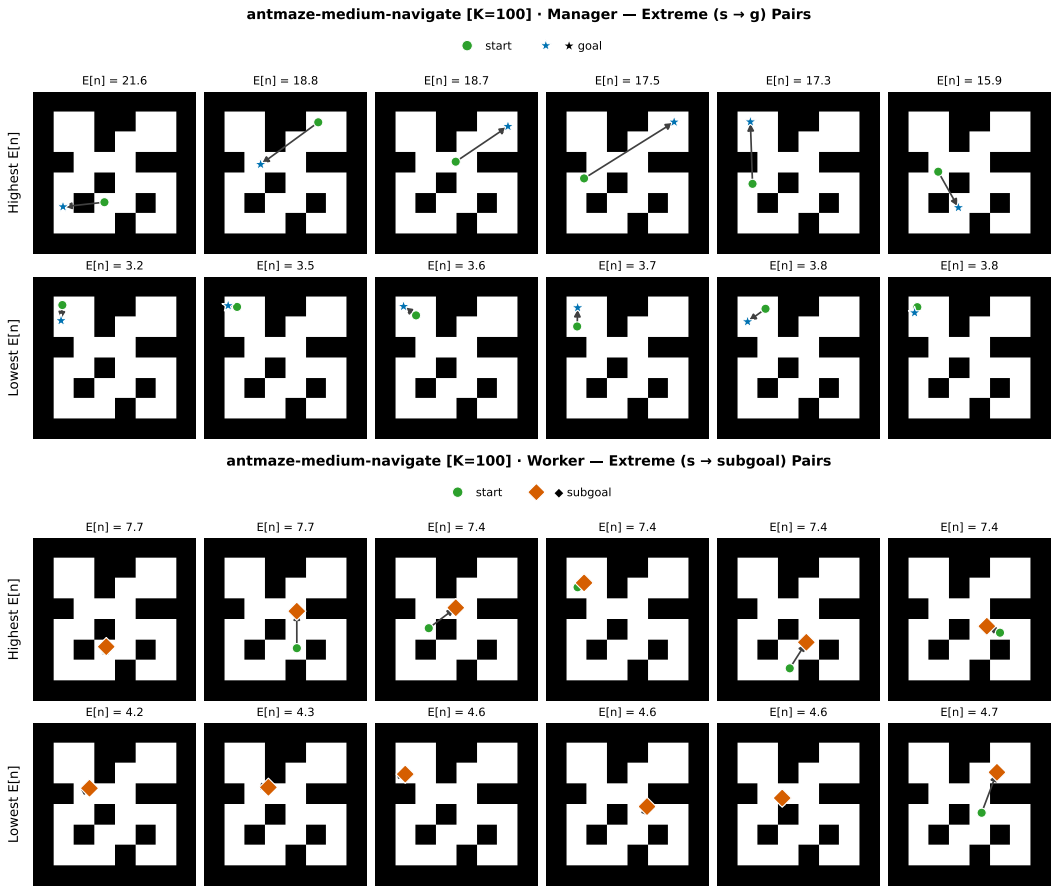

Figure 12: Extreme $(s \to g)$ pairs by manager recurrence (top) and worker recurrence (bottom) on antmaze-medium-navigate. Each row shows the top-6 highest- and bottom-6 lowest-recurrence pairs sampled from the training distribution, drawn as start → target arrows on the environment; recurrence is averaged over 10 seeds.