LEARNING TO ACT WHILE THINKING

Anonymous authors
Paper under double-blind review

ABSTRACT

Operating in real-time, dynamic environments requires agents to respond quickly to sudden changes while also planning over long horizons to solve difficult tasks. In such settings, plans take time and energy to compute and the environment does not pause while an agent thinks. As a result, planning should only be done when necessary and plans may arrive stale, as they are computed from a prior state of the world. Such an agent therefore must decide when to request a plan, how to act while waiting for it, and to what extent to follow it once it arrives. We formalize this setting as a real-time meta-reasoning MDP, in which an agent can request a plan from a slow, asynchronous planner but incurs a cost and delay before the plan is returned. Using this formalism, we show when requesting a plan improves an agent's policy given a cost and delay. We then describe our method **A**cting **W**hile **T**hinking (AWT) to train reinforcement learning agents to learn to request plans and act while they are computed. We then evaluate AWT in five real-time, dynamic tasks, and show that learned policies achieve a higher return and use fewer plans than fixed meta-reasoning and heuristic baselines. Finally, we analyze the behavior of the resulting agents and show when they learn to query additional computation.

1 INTRODUCTION

Agents that operate in real-time, dynamic environments must react quickly as the world changes. At the same time, many complex tasks, such as autonomous driving, search and rescue, or surgery, require reasoning over long horizons to effectively accomplish long-term goals. Such long-horizon reasoning requires an agent to search over many possible future states, which may take extensive computation time. In this work, we consider how to enable reinforcement learning (RL) agents to learn to act in real-time, dynamic environments while also effectively using long-horizon planning. We assume the agent has access to a planner, such as Monte Carlo tree search or a vision-language model, that can produce a plan for accomplishing a task goal, but there is a delay before the plan arrives. Additionally, we assume requesting a plan is costly in terms of money or energy, and we model this by penalizing the agent's reward each time the planner is queried.

In such environments, naive behavior while waiting for plans, such as standing still until planning completes, may fail, since other objects can collide with the agent. For example, if an autonomous vehicle is traveling at high speed and is suddenly forced off its route by an unexpected obstacle, it cannot simply stop and wait for its global path planner to return a new route before acting. Furthermore, the delay means that plans may be *stale* by the time they are returned to the agent, because they were computed using an outdated state of the world. To act effectively in such real-time, dynamic domains, where planning takes time, we study how artificial agents can learn both *when* to request a plan and *how* to act while it is being produced and after it arrives. In this setting, there are three challenges to acting effectively. First, the agent must know when to plan and perform meta-reasoning. Successful meta-reasoning requires the agent to know when its current plan is insufficient and when it must pay the cost of requesting a new one. Second, after the agent requests a plan, it must produce effective actions during the computation delay. As the environment is dynamic, it must take actions that prevent failure without access to an updated plan. Third, once the plan arrives, the agent must account for its staleness. Since the plan was computed on a previous state of the world, following it naively may lead to failure, so the agent must selectively follow the plan.

With these challenges in mind, in this work we ask:

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.

Can an RL agent learn to selectively use a slow, deliberative planning routine while acting in a real-time, dynamic domain?

To answer this question, we first formalize this problem as a *real-time meta-reasoning MDP*, in which an agent is given the ability to request a plan from a delayed and computationally expensive planner. We then introduce **A**cting **W**hile **T**hinking (AWT), a method for training agents to meta-reason about when to request delayed plans, how to act while they are being computed, and how to follow them once they arrive. Across five goal-driven, real-time, dynamic environments, AWT produces agents that achieve a higher return and use fewer plans than fixed meta-reasoning and waiting-behavior baselines, and we further ablate AWT's components and analyze the learned agents' behavior in one environment. In summary, our work makes the following contributions:

1. We provide a novel MDP formalism for simultaneous meta-reasoning and acting, and prove when the expected improvement of requesting a plan exceeds its cost.
2. We introduce AWT, an RL method for learning when to invoke an asynchronous planner, how to act effectively while a plan is in flight, and how to follow it on arrival.
3. We empirically show AWT achieves higher return than fixed meta-reasoning and heuristic waiting baselines across five real-time, dynamic environments.

2 Related Work

Variable Computation. A number of works investigate how agents can vary the amount of computation during decision-making. Labiosa & Hanna (2026) account for the difference in time required to generate actions via planning versus model-free acting, and train an RL-based meta-controller to choose between them at each timestep, but do not consider dynamic domains. Several works use RL to switch between action-production mechanisms or vary computation per decision (Kästner et al., 2021; Sharma et al., 2025; Dey et al., 2023; Metelli et al., 2020; Orenstein et al., 2025) or to decide how much computation to perform before acting (Callaway et al., 2018; Hamrick et al., 2017; Budd et al., 2024; Bergamaschi Ganapini et al., 2025), but treat computation as synchronous with no variable latency. In the LLM literature, many works study when to use additional reasoning computation, but do not consider real-time domains (Paglieri et al., 2025; Fang et al., 2025; Sabbata et al., 2025; Hanna & Corrado, 2025). Wen et al. (2025) combines a faster action generator with a slower reasoner, but fix the amount of reasoning per planning step and do not address when to replan.

Real-Time Planning. Several works study asynchronous planning, in which the environment continues to transition while planning occurs. Many approaches focus on producing actions within a time budget or deadline (Thomas et al., 2024; Dionne et al., 2011; Coles et al., 2024; Elboher et al., 2023; Fritz & McIlraith, 2012), while others consider when replanning is worth its time cost (Lemai & Ingrand; Cashmore et al., 2019). However, all of these works rely on heuristics rather than learned policies. In contrast, we learn replanning decisions end-to-end with RL.

Real-Time Reinforcement Learning. Prior work on real-time RL has largely focused on two distinct problems. The first is improving the speed of the learning-acting loop (Hester et al., 2012; Hester & Stone, 2013; Travnik et al., 2018; Caarls & Schuitema, 2016; Dulac-Arnold et al., 2021; Ramstedt & Pal, 2019; Böhm et al., 2022), which is different to our focus on how agents act in real time rather than how quickly they learn. The second is handling the observation-to-action delay introduced by the agent itself (Chen et al., 2020; 2021; Anokin et al., 2025; Lee et al., 2024; Riemer et al., 2025; Liotet et al., 2022; 2021; Du et al., 2022). Our setting differs in that we assume the agent can produce reactive actions quickly (e.g., via local inference) and instead focus on how it should manage the additional latency incurred by invoking an expensive external planner.

3 Formal Model of Acting With A Delayed Planner

In this section we define MDPs and then extend them to real-time meta-reasoning MDPs. We then use this formalism to provide theoretical analysis for when an agent should choose to call a delayed, expensive planner.

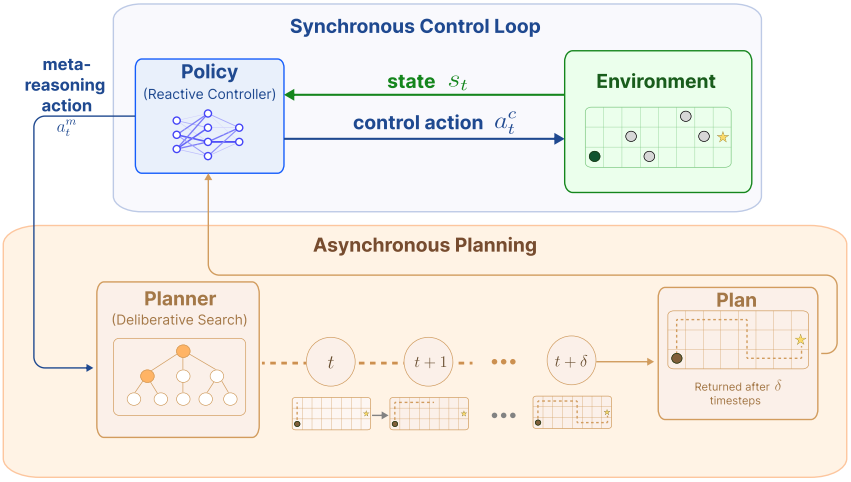


Figure 1: Visualization of the real-time meta-reasoning MDP. In this MDP we formally analyze when policy improvement selects a plan action instead of a control action and propose an RL-based method AWT to train agents to operate in real-time domains with slow, asynchronous planners.

3.1 Markov Decision Processes

Markov Decision Processes (MDPs) are defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ defines the state transition dynamics, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. At each timestep t, the agent observes state $S_t \in \mathcal{S}$ and selects an action $A_t \in \mathcal{A}$ according to a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$. The environment transitions to a new state S_t according to p, and the agent receives reward $R_t = r(S_t, A_t)$. The objective of RL is to learn a policy which maximizes the expected discounted return: $J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t)\right]$ where $\mathbb{E}_\pi$ is the expected distribution of states and actions when following policy π.

3.2 Real-Time Meta-Reasoning MDP

We formalize reactive control with asynchronous planning as the *real-time meta-reasoning MDP* defined by the tuple $(\widetilde{\mathcal{S}}, \widetilde{\mathcal{A}}, \tilde{p}, \tilde{r}, \gamma)$, where $\widetilde{\mathcal{S}}$ is the augmented state space, $\widetilde{\mathcal{A}}$ is the augmented action space, $\tilde{p}$ define the augmented transition dynamics, $\tilde{r}$ is the augmented reward function, and γ is the discount factor as in a standard MDP. Many of the components of a real-time meta-reasoning MDP relate to *plans*, and since p is already used to denote the transition dynamics function, we use ℓ to denote plans and plan-related variables.

State Space. The augmented state space $\widetilde{\mathcal{S}} = \mathcal{S}^e \times \mathcal{S}^\ell$ combines the base environment state space $\mathcal{S}^e$ and the plan state space $\mathcal{S}^\ell$. At each timestep the agent observes $\tilde{s}_t := (s_t^e, s_t^\ell)$, where $s_t^e \in \mathcal{S}^e$ is the current environment state and $s_t^\ell \in \mathcal{S}^\ell$ is the current plan state. The plan state is the tuple $s_t^\ell = (\ell_t, d_t, s_t^r)$, where $\mathcal{S}^\ell = \mathcal{L} \times \mathbb{N} \times \mathcal{S}^e$, and $\mathcal{L}$ is the set of plans the planner can return. The plan $\ell_t \in \mathcal{L}$ is the most recent plan returned by the planner. We leave the representation of ℓ_t general, as it may take different forms in different domains, such as a sequence of waypoints in a navigation domain or a set of instructions in a language-based domain. The delay counter $d_t \in \mathbb{N}$ is the number of timesteps remaining until a requested plan is returned, with $d_t = 0$ when no planning computation is in progress. The request state $s_t^r \in \mathcal{S}^e$ is the environment state from which the most recent planner request was issued. The delay counter and the request state are included so that the augmented state is Markov as the plan state transitions depends both on *when* and from *what state* the plan was called.

Action Space. The augmented action space $\widetilde{\mathcal{A}} = \mathcal{A}^e \times \mathcal{A}^\ell$ combines the base environment control action space $\mathcal{A}^e$ and a binary plan action space $\mathcal{A}^\ell = \{\texttt{request}, \texttt{no-request}\}$, which indicates whether to call the planner. At each timestep t the agent selects both a control action $a_t^e \in \mathcal{A}^e$ and a plan action $a_t^\ell \in \mathcal{A}^\ell$ according to a policy $\pi(\tilde{a}_t \mid \tilde{s}_t)$, where $\tilde{a}_t = (a_t^e, a_t^\ell) \in \widetilde{\mathcal{A}}$.

Planner and Planning Delay. The planner is a map $f : \mathcal{S}^e \to \mathcal{L}$ that takes an environment state and outputs a plan to follow in the environment. The plan state transitions according to $\tilde{p}^\ell$, which is determined by f and a planning delay δ. The delay δ may be fixed or drawn from a distribution $\delta \sim \mathcal{D}$ at the time of each request, which models variable planner execution time. A request is *in-flight* at timestep t when $a_t^\ell = \texttt{request}$ or $d_t > 0$, meaning that the agent either requested a plan at time t or is waiting on one. Given $\tilde{s}_t$ and $\tilde{a}_t$, the plan state components transition as follows:

$$s_{t+1}^r = \begin{cases} s_t^e & \text{if } a_t^\ell = \texttt{request} \\ s_t^r & \text{otherwise,} \end{cases} \tag{1}$$

$$d_{t+1} = \begin{cases} \delta & \text{if } a_t^\ell = \texttt{request} \\ \max(d_t - 1, 0) & \text{otherwise,} \end{cases} \tag{2}$$

$$\ell_{t+1} = \begin{cases} f(s_t^r) & \text{if } a^\ell = \texttt{no-request} \text{ and } d_t = 1 \\ \ell_t & \text{otherwise.} \end{cases} \tag{3}$$

If a plan is in-progress, a planning request resets the delay counter and restarts computation from the current environment state s_t^e. Between a plan request and its arrival the agent must act for δ timesteps without an updated plan. Upon arrival the plan may already be stale, since it was computed from the environment state at request time, which may differ from the current environment state. The environment state transition dynamics and the plan state transition dynamics are independent so the full transition dynamics are defined as

$$\tilde{p}(\tilde{s}_{t+1} \mid \tilde{s}_t, \tilde{a}_t) = p^e(s_{t+1}^e \mid s_t^e, a_t^e)\, p^\ell(s_{t+1}^\ell \mid s_t^e, s_t^\ell, a_t^\ell).$$

Reward. The augmented reward $\tilde{r}(\tilde{s}_t, \tilde{a}_t) = r^e(s_t^e, a_t^e) - r^\ell \cdot \mathbb{1}[a_t^\ell = \texttt{request}]$ combines the base environment reward with a planning cost, where $r^\ell > 0$ is a fixed scalar penalty incurred whenever the planner is invoked.

Relationship to prior formalisms. Our formalism differs from prior meta-reasoning formalisms in that we consider simultaneously selecting actions and computations (Lin et al., 2015) and real-time, dynamic environments (Hanna & Corrado, 2025).

3.3 When Does an Agent Learn to Plan

In a setting where plans are delayed and costly, when should a reward-maximizing agent choose to plan? We answer this question by analyzing when policy iteration would update a policy to use a planning action as opposed to an exclusively control step (Sutton & Barto, 2018). We derive our analysis based on Hanna & Corrado's *thought MDP* (2025). To understand when an agent will choose a planning action, we formally analyze the policy improvement step of policy iteration. While policy iteration is a dynamic programming method, generalized policy iteration underlies the policy improvement step for most RL algorithms. We analyze a state $\tilde{s}$ with no request in-flight, so $d = 0$. We make two assumptions on the policy π, over the set of δ transitions after $\tilde{s}$, during which a plan requested in $\tilde{s}$ has not yet arrived. We write π^e and π^ℓ for the control and planning components of π. For the proof of Proposition 1 see Appendix B.

Assumption 1. The control action a^e does not depend on the delay counter d or the request state s^r, and when no request is in-flight the request action a^ℓ does not depend on the request state, so for all environment states $s^e, s^r, s^q \in \mathcal{S}^e$, all plans ℓ, and all $d, d' \in \mathbb{N}$,

$$\pi^e\big(s^e, (\ell, d, s^r)\big) = \pi^e\big(s^e, (\ell, d', s^q)\big), \qquad \pi^\ell\big(s^e, (\ell, 0, s^r)\big) = \pi^\ell\big(s^e, (\ell, 0, s^q)\big). \tag{4}$$

Assumption 2. For the state $\tilde{s}$ with no request in-flight so $d = 0$ and and a control action a^e, let $\mathcal{W}(\tilde{s}, a^e)$ denote the set of states that can be reached from 1 to δ transitions when a^e is taken in $\tilde{s}$ and π followed after. Over this window, the policy requests no new plan so

$$\pi^\ell(\hat{s}) = \texttt{no-request} \quad \forall\, \hat{s} \in \mathcal{W}(\tilde{s}, a^e). \tag{5}$$

Proposition 1. *Let π be a policy in a real-time meta-reasoning MDP satisfying Assumptions 1 and 2. Let $\tilde{s} = (s^e, (\ell, 0, s^r))$ be a state with no request in-flight. Let $p_{\pi,\ell}^{\delta+1}(\cdot \mid s^e, a^e)$ denote the distribution of states when a plan is received $\delta + 1$ steps in the future, given that a^e is taken in s^e and $\pi^e(\cdot, \ell)$*

is followed until then, and let $\ell' = f(s^e)$ denote the plan received. Suppose the policy improvement step of policy iteration sets $\pi'(\tilde{s}) \leftarrow (a^e, \texttt{request})$, so that the improved policy requests a plan but does not change the control action, then

$$\gamma^{\delta+1}\, \mathbb{E}_{s'' \sim p^{\delta+1}_{\pi,\ell}(\cdot \mid s^e, a^e)} \big[v_\pi(s'', \ell') - v_\pi(s'', \ell) \big] \geq r^\ell. \tag{6}$$

Therefore, an improved policy will request a plan only if the discounted expected improvement over the current plan produces greater return than the immediate reward penalty.

Discussion The planning delay δ reduces the value of a requested plan in two ways. The first is from the discount factor $\gamma^{\delta+1}$, which scales the expected improvement from a new plan by the number of steps it takes to arrive. The second is staleness, which occurs since the new plan $\ell' = f(s^e)$ is computed from the environment state in which it was requested, but the agent cannot act on it until $\delta + 1$ steps later. The new state may be substantially different than the request state which decreases the value of the plan. Both effects decrease the value of a new plan as δ increases while the planning cost r^p is fixed. As a result, for a sufficiently long delay in a dynamic environment, the condition cannot be satisfied and the agent should never request a plan.

4 Acting While Thinking

In this section, we describe a method, Acting While Thinking (AWT), for training agents to perform simultaneous meta-reasoning and reactive control in real-time, dynamic environments. Our method takes as input an environment $\mathcal{M}^e$ that is formalized as a standard MDP and a planner f that returns valid plans for the agent to follow to reach a goal state. From these inputs, AWT constructs an instance of a real-time meta-reasoning MDP as defined in Section 3.2 and applies any off-the-shelf RL algorithm to return a policy π_θ that jointly performs reactive control and meta-reasoning.

Real-time meta-reasoning MDP instantiation. Given the inputs, AWT augments the base MDP observation, o^e_t, with the plan observation, request state, and delay counter, producing the augmented observation $\tilde{o}_t = (o^e_t, o^\ell_t)$ and the augmented action $\tilde{a}_t = (a^e_t, a^\ell_t)$, where a^ℓ_t is the request action used to query the planner. The plan observation $o^\ell_t = (\ell_t, u_t, h_t)$ contains three components: the plan ℓ_t, planner meta-data u_t, and an action history h_t. The plan ℓ_t is the most recent plan returned by the planner. The plan meta-data $u_t = \big(d_t,\ \mathbb{1}[d_t > 0],\ \tau_t\big)$ provides planner status information, where d_t is the planning delay remaining until an in-flight plan is returned, $\mathbb{1}[d_t > 0]$ indicates whether a plan is in-flight, and τ_t gives the time since the last plan was returned. Lastly, h_t contains the last K actions the agent has taken.

Policy architecture. To output both a control action and a request action, AWT uses a single network with two heads, $\pi_\theta(\tilde{a} \mid \tilde{o}) = \pi^e_\theta(a^e \mid \tilde{o})\, \pi^\ell_\theta(a^\ell \mid \tilde{o})$, where the two heads share all layers except for a separate final linear layer. The control head outputs an action that matches the environment action space and the planning head outputs two logits for which a softmax produces the probability of requesting a plan.

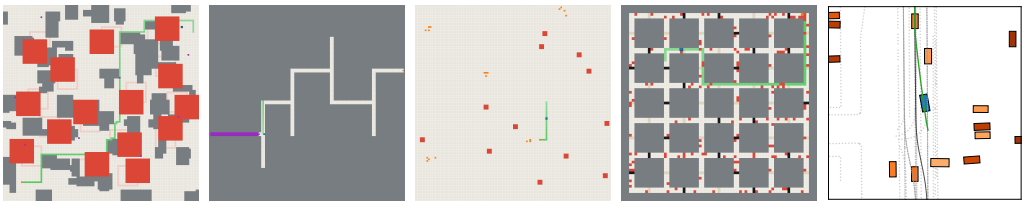
Plan following. After the observation and action spaces are defined, the RL agent must learn how to perform meta-reasoning. An essential component of learning when to request a plan is understanding the value of following a plan, which requires learning *how* to follow it. An agent that has not learned to follow plans only pays the plan cost r^ℓ without receiving the long-horizon benefit of reaching the goal. AWT addresses this challenge by adding an additional shaping reward $r^f(s^e_t, a^e_t, \ell_t)$ that rewards the agent for following its plan. The full reward is therefore $r_t = r^e_t + r^f_t - r^\ell \cdot \mathbb{1}[a^\ell_t = \texttt{request}]$.

5 Experiments

In this section we provide an overview of our environments, describe our experimental setup, and detail our empirical results.

5.1 Environment Suite

Here we describe our suite of five goal-driven environments. For a more detail, see Appendix A.



(a) Evade (b) Flood (c) Salvage (d) Warehouse (e) Waymax

Figure 2: Visual representation of our suite of environments. In all environments the agent is *blue* and the plan is *green*. In all gridworlds, and the goal(s) are *orange*. In Evade, Salvage, and Warehouse, objects which move around the world in a predefined motion are *red*. In Evade and Flood, objects which pursue the agent are *purple*. In Waymax, the desired path is in *black*.

1. **Evade:** The agent must reach a goal cell while routing around fixed obstacles and a set of guards that patrol in loops. A set of pursuers chase the agent and colliding with a guard or pursuer ends the episode. The planner routes around obstacles and guards to the goal.
2. **Flood:** The agent must reach a goal cell through a maze of chained T-branches. A plan provides the direction to take a the next intersection. A flood begins behind the agent and expands down the maze, ending the episode if it catches the agent.
3. **Salvage:** The agent must collect as many goal items as possible that are placed around the world in clusters. The planner routes to the nearest item. Asteroids moved at a fixed velocity and bounce off walls; contacting one ends the episode.
4. **Warehouse:** The agent must reach a goal cell while avoiding collisions with the other agents. Doors open and shut simultaneously and block off paths. The planner routes the agent to the goal given the state of the doors.
5. **Waymax:** The Waymax simulator (Gulino et al., 2023) replayes 90-step, 10Hz logs from real self-driving scenarios. The agent must reach a goal position while following traffic rules. Colliding with other vehicles end the episodes, and we add additional vehicles to increase meta-reasoning pressure.

5.2 Experimental Setup

In the gridworlds, the environment observation $o_t^e \in \mathbb{R}^{W \times W \times C}$ is a window centered on the agent, with C channels, one for each object type, following (Matthews et al., 2024). A cell's value in a given channel is 1 if that object type occupies the cell, and 0 otherwise. The plan ℓ_t adds a plan channel and a plan progress channel. The plan channel sets every cell on the plan to 1 and all other cells to 0. The plan progress channel encodes each cell's progress toward the goal, where the most negative is at the farthest plan cell and increase toward the goal. Let $\ell_t = (x_1, \ldots, x_N)$ be the plan, with each x_i a grid cell on the path to the goal and $x_{i^\star}$ the plan cell nearest the agent. The plan progress channel's value at x_i is $\text{clip}\big((i - i^\star)/H, -1, 1\big)$ for a fixed horizon H, and 0 for cells not on the plan. In Waymax, observation from V-Max (Charraut et al., 2025) which includes the agent's recent trajectory, nearby agents, nearby road points, and traffic lights. The plan state ℓ_t consists of 10 tokens (x, y, p) giving each plan point's local-frame position (x, y) and desired speed p. For the planner meta-data, the delay timer is normalized by the delay δ, so the input is d_t/δ, and the time since the last plan was returned is normalized by the episode length T, so the input is τ_t/T.

Plan following reward. Let $c_t^x = \|x_t^a - x_{i^\star}\|_1$ be the agent's distance to the nearest plan position, and let $c_t^N = N - i^\star$ be the remaining steps along the plan to the goal from $x_{i^\star}$. We define $c_t = c_t^x + c_t^N$ the sum, and the plan-following reward as a a scaled version $r_t^f = \alpha(c_t - c_{t-1})$.

Architecture and training. We train all agents using PPO (Schulman et al., 2017) with the Brax training library (Freeman et al., 2021). In the gridworlds, we use an MLP encoder and in Waymax, we use V-Max implementation of the Wayformer (Nayakanti et al., 2022). Hyperparameters are provided in Appendix D.

Baselines. We compare our trained agents against two types of baselines. The first isolates behavior during the delay with. *Freeze* keeps the agent still in gridworld environments and breaks in Waymax.

	Warehouse			Evade			Flood			Salvage			Waymax		
Method	Ret.↑	Suc.↑	Pln.	Ret.↑	Suc.↑	Pln.	Ret.↑	Suc.↑	Pln.	Ret.↑	Suc.↑	Pln.	Ret.↑	Suc.↑	Pln.
Freeze	2.4	68	0.0	0.4	59	0.2	3.0	26	0.9	17.0	22	4.2	10.5	64	1.5
	$^{+0.1}_{-0.1}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$	$^{+0.2}_{-0.3}$	$^{+1}_{-1}$	$^{+0.1}_{-0.1}$	$^{+0.8}_{-1.1}$	$^{+3}_{-4}$	$^{+0.1}_{-0.2}$	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$
Avoid	2.1	66	0.9	2.1	69	1.1	9.6	59	5.9	19.3	**26**	4.5	10.6	64	1.9
	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$	$^{+3.1}_{-3.5}$	$^{+15}_{-17}$	$^{+0.6}_{-0.7}$	$^{+0.2}_{-0.1}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.1}_{-0.1}$
Short	3.3	**92**	7.5	0.9	**86**	9.2	15.3	88	9.4	7.9	11	23.5	11.3	69	2.2
	$^{+0.1}_{-0.1}$	$^{+1}_{-1}$	$^{+0.1}_{-0.1}$	$^{+0.1}_{-0.1}$	$^{+1}_{-1}$	$^{+0.1}_{-0.1}$	$^{+0.6}_{-0.6}$	$^{+3}_{-3}$	$^{+0.1}_{-0.1}$	$^{+0.1}_{-0.1}$	$^{+1}_{-1}$	$^{+0.1}_{-0.1}$	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$
Medium	4.6	91	3.8	**2.7**	83	4.6	14.5	82	6.6	11.0	2	11.7	9.9	65	0.8
	$^{+0.1}_{-0.1}$	$^{+1}_{-1}$	$^{+0.1}_{-0.1}$	$^{+0.1}_{-0.1}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$	$^{+0.1}_{-0.1}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$	$^{+0.1}_{-0.1}$	$^{+0}_{-0}$	$^{+0.0}_{-0.0}$	$^{+0.3}_{-0.3}$	$^{+2}_{-2}$	$^{+0.0}_{-0.0}$
Long	4.5	86	1.9	2.5	76	2.3	4.7	35	4.0	9.9	0	5.9	7.5	52	0.5
	$^{+0.1}_{-0.1}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$	$^{+0.1}_{-0.2}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$	$^{+0.8}_{-1.0}$	$^{+4}_{-5}$	$^{+0.2}_{-0.2}$	$^{+0.1}_{-0.1}$	$^{+0}_{-0}$	$^{+0.0}_{-0.0}$	$^{+0.3}_{-0.3}$	$^{+2}_{-2}$	$^{+0.0}_{-0.0}$
AWT	**5.7**	90	1.8	**3.0**	76	1.4	**17.9**	**100**	5.9	**20.6**	**27**	4.5	**11.71**	**71**	1.9
	$^{+0.1}_{-0.1}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.1}_{-0.1}$	$^{+0.1}_{-0.1}$	$^{+0}_{-0}$	$^{+0.7}_{-0.7}$	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.1}_{-0.1}$

Table 1: Comparison of the AWT agent to fixed meta-reasoning and heuristic waiting policy baselines. Freeze and Avoid heuristically move during computation. Short, Medium, and Long replan at fixed frequencies. Return, success rate (%), and plans per episode across environment suite. Values are means over $n = 10$ training seeds for gridworlds and $n = 5$ for Waymax because of computational expense, with percentile bootstrap CIs over all training seeds and 10,000 evaluation episodes. Best values are bolded. Plans per episode is reported as a behavior description so it is not bolded.

Avoid heuristically moves the agent away from the closest object in the gridworlds and continues straight at the current speed in Waymax. The second isolates learning when to use compute. We compare to a *Short*, *Medium*, and *Long* replanning interval. All agents receive a plan upon initialization so that the periodic agent does not have to wait for its first request. All baselines use the same observation space and curriculum defined in Section 4.

Evaluation Metrics. We evaluate all policies on three metrics: *Return*, *Success Rate*, and *Plans per episode*. Return combines the base environment reward and the cost of calling the planner. Success rate is the percentage of episodes in which the agent completes the task. Plans per episode is the number of plans used by the agent, whether its replanning policy is learned or fixed. The evaluation reward is the reward defined in Section 4, which combines the environment reward r^e, the plan following reward r^f, and the plan request cost r^ℓ. Periodic agents are trained without r^ℓ and evaluated with it only afterward, since they cannot influence this reward component and we found that including it during training destabilized learning.

5.3 Empirical Analysis

In this section, we analyze whether AWT produces agents which are effective at meta-reasoning and acting during the computation-delay. Table 1 shows the return, success rate, and plans per episode of AWT-trained agents in comparison to fixed meta-reasoning and heuristic delay-behavior agents. In all five environments, AWT-trained agents achieve a higher return than the baselines, showing that, in comparison to the fixed meta-reasoning baselines, the AWT-trained agent's meta-reasoning more effectively balances the cost of planning and the performance improvement than any of the fixed frequencies. In comparison to the heuristic delay-behavior policies, the improved reward shows that in such real-time, dynamic environments, the agent learns effective behavior while waiting to avoid obstacles simultaneously with meta-reasoning. Across the suite of environments, Evade is the only one in which a baseline had a higher success rate, with the Short planning frequency achieving about a 10% higher success rate than the AWT agent. Qualitatively, in Evade the AWT agent learns reactive replanning and requests plans when it can observe that its plan is blocked. In Evade, the environment also has pursuers that follow the agent and cause terminations. Therefore, the agent has to stay in the same area during the waiting period as it does not have an updated plan, which allows the pursuers to catch up to the agent and corner it, causing failures. The only environment in which a heuristic-waiting policy achieves as high a success rate as the AWT-trained agent is Avoid in Salvage. This result occurs because, in that domain, obstacles are relatively sparse, so simply moving away from them is a reasonable behavior. In contrast, in the other environments with more obstacles, simply moving away does not perform as well as the learned behavior.

Ablation	Warehouse			Evade			Flood			Salvage		
	Ret.↑	Suc.↑	Pln.	Ret.↑	Suc.↑	Pln.	Ret.↑	Suc.↑	Pln.	Ret.↑	Suc.↑	Pln.
AWT	**5.7**	**90**	1.8	**3.0**	76	1.4	17.9	**100**	5.9	**20.6**	**27**	4.5
	$^{+0.1}_{-0.1}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.1}_{-0.1}$	$^{+0.1}_{-0.1}$	$^{+0}_{-0}$	$^{+0.7}_{-0.7}$	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$
No planner meta-data	2.4	69	0.0	**3.3**	**78**	1.9	**18.1**	**100**	4.0	**20.7**	**28**	4.5
	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.1}_{-0.1}$	$^{+0.0}_{-0.0}$	$^{+0}_{-0}$	$^{+0.0}_{-0.0}$	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$
No action history	5.0	86	1.5	1.1	65	1.4	17.7	**100**	8.3	18.6	22	4.8
	$^{+0.2}_{-0.3}$	$^{+1}_{-2}$	$^{+0.1}_{-0.1}$	$^{+1.1}_{-1.3}$	$^{+7}_{-8}$	$^{+0.4}_{-0.5}$	$^{+0.1}_{-0.1}$	$^{+0}_{-0}$	$^{+0.6}_{-0.6}$	$^{+0.3}_{-0.4}$	$^{+1}_{-1}$	$^{+0.1}_{-0.1}$
No plan-following reward	1.8	68	1.3	-5.4	0	0.0	-3.1	13	0.7	7.6	**27**	4.6
	$^{+2.0}_{-2.5}$	$^{+18}_{-25}$	$^{+0.4}_{-0.5}$	$^{+0.0}_{-0.0}$	$^{+0}_{-0}$	$^{+0.0}_{-0.0}$	$^{+0.3}_{-0.5}$	$^{+2}_{-3}$	$^{+0.4}_{-0.4}$	$^{+0.1}_{-0.1}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$

Table 2: Ablations of the observation of the AWT method. No planer meta-data scalars removes the information about in-flight plans and time since a plan was last delivered. No action history removes the last K actions. No plan following reward removes the plan following shapwing rewards. Return, success rate (%), and plans per episode are reported as means over $n = 10$ training seeds with 95% percentile bootstrap CIs. Best result is bolded.

5.4 Ablations

In this section, we analyze the ablations of the AWT observation space and the agent's sensitivity to the planning cost. Due to compute cost, we focus our experiments on the gridworld environments. First, Table 2 shows the performance of agents trained without individual components of the AWT observation space. These results show that the full AWT agent performs the best, or close to the best, in all domains. Interestingly, the plan-timing scalars only impact the return in Warehouse, suggesting that this is the only domain which requires timing to perform well in. The action history improves return in all of the environments, as it helps prevent the agent from looping actions and provides extra information on when a plan was last called. Lastly, the plan following reward did not improve return in Flood or Salvage, likely because the plans in Salvage are of shorter horizon than those in Evade and Warehouse. Second, in Table 3, we show the results of varying the planning cost across domains. As the cost to plan increases, the number of plans per episode generally decreases, and the success rate also decreases, showing that as the plan cost increases, the agent learns to use fewer plans, which drops the success rate. This result seems to represent an empirical validation of Proposition 1 as the cost to plan increases, the value of an additional plan decreases, and thus the agent uses fewer plans. Interestingly, the reward and success rate in Warehouse and Salvage only decreases slightly as the plan cost increases, while the performance in Evade and Flood decreases substantially, possibly because Warehouse and Salvage are easier environments, whereas in the more difficult Evade environment, a small drop in plans causes a large drop in success due to the pursuers.

5.5 Behavior Analysis

In Table 4, we provide three evaluations to determine what behavior the AWT agents in Evade learn. We chose Evade because it was the most challenging environment for the AWT-trained agent, as it is the only environment in which a baseline had a higher success rate. The *Asks when route is*

Setting	Warehouse			Evade			T-maze			Salvage		
	Ret.	Suc.	Pln.	Ret.	Suc.	Pln.	Ret.	Suc.	Pln.	Ret.	Suc.	Pln.
Cost 0.1	6.5	90	1.8	4.1	79	2.1	17.9	100	5.9	22.6	29	5.1
	$^{+0.1}_{-0.1}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$	$^{+0.1}_{-0.1}$	$^{+1}_{-1}$	$^{+0.1}_{-0.1}$	$^{+0.1}_{-0.1}$	$^{+0}_{-0}$	$^{+0.7}_{-0.7}$	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.1}_{-0.1}$
Cost 0.5	5.7	90	1.8	3.0	76	1.4	6.2	45	1.7	20.6	27	4.5
	$^{+0.1}_{-0.1}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.1}_{-0.1}$	$^{+3.7}_{-3.0}$	$^{+20}_{-16}$	$^{+0.8}_{-0.7}$	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$
Cost 1.0	4.4	87	1.6	-0.0	56	0.0	0.6	20	0.5	18.5	26	4.3
	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.1}_{-0.1}$	$^{+0.2}_{-0.2}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$	$^{+1.2}_{-1.1}$	$^{+6}_{-5}$	$^{+0.5}_{-0.5}$	$^{+0.1}_{-0.1}$	$^{+1}_{-1}$	$^{+0.0}_{-0.0}$

Table 3: Ablation of the cost to plan on agent behavior. The full AWT method was trained and evaluated at three plan costs: 0.1, 0.5, and 1.0. Return, success percent, and plans per episode are reprted as means with 95% bootstrapped CIs reported over 10 seeds and 10,000 evaluation episodes.

		Request Plan Every N steps		
	AWT	$N = 20$	$N = 40$	$N = 80$
Plans per episode	1.4 $^{+0.07}_{-0.07}$	9.2 $^{+0.08}_{-0.08}$	4.6 $^{+0.04}_{-0.05}$	2.3 $^{+0.03}_{-0.03}$
Asks when route is blocked (%)	68 $^{+2.0}_{-2.1}$	6 $^{+0.4}_{-0.4}$	11 $^{+0.5}_{-0.5}$	20 $^{+1.2}_{-1.2}$
Steps holding a blocked plan (%)	41 $^{+1.3}_{-1.3}$	25 $^{+0.8}_{-0.7}$	31 $^{+0.7}_{-0.7}$	42 $^{+0.9}_{-1.0}$

Table 4: Behavior analysis on the Evade environment for $n = 10$ training seeds with 95% percentile bootstrapped CIs. Plans per episode is the average number of requests per episode. Asks when routed is blocked is the percent of plans requested where the agent can see a blocked route. Steps holding a blocked plan is the average number of steps/episode where the agent's plan is blocked.

blocked metric shows what percent of requests were made when the agent could observe that its plan was blocked. Well over half of the requests by the AWT agent occurred when it could identify a blocked path, suggesting that the agent learns to follow its plan as far as it can until it requests a new one. Interestingly, this means it is learning *reactive* meta-reasoning instead of *proactive* meta-reasoning. The periodic baseline that requests every 20 steps performs *proactive* reasoning, as the fixed frequency requests a new plan before the agent sees that its plan is blocked, which we can see from the 62% difference in the route-is-blocked metric. However, while the agent performs seemingly intelligent meta-reasoning, it still has a very high percentage of steps in which it holds a blocked plan in comparison to the $N = 20$ and $N = 40$ baselines. This result suggests that the small view window limits the agent's ability to avoid blocked plans.

6 Discussion and Future Work

Our empirical results demonstrated that RL agents can learn to simultaneously perform meta-reasoning and reactive control in dynamic, real-time domains. In this section, we identify limitations of this work and outline directions for future research. First, as described in Section 5.5, our learned agents seem to perform *reactive* meta-reasoning. While this this behavior yields a higher return than baselines in our environments, it has a lower success rate than the $N = 20$ fixed-frequency baseline in the Evade environment suggesting a promising direction in enabling agents to perform *proactive* meta-reasoning. Second, all of our experiments use oracle planners that return optimal plans. This experimental choice lets us isolate the meta-reasoning problem, but in many settings, plans are likely to be suboptimal even before they become stale. Therefore, a possible future direction is enabling agent to learn to use suboptimal plans. Lastly, all of the planners used in our experiments are classical search planners. Currently, there is much research on reasoning in LLMs and VLAs; however, most of this work does not focus on real-time, dynamic settings, suggesting a similar method AWT applied to language-based planners may enable use in real-time, dynamic domains.

7 Conclusion

As agents are deployed in settings where computation is both slow and costly, enabling them to allocate computation while effectively handling a changing environment will lead to more general and useful agents. In this work, we studied a setting in which an agent acts in real-time, dynamic domains with access to a computationally expensive and delayed planner. We introduced the real-time meta-reasoning MDP and used this formalism to show when policy improvement would update an action to request a plan. We then introduced AWT, a procedure for training RL agents to act in such settings with access to delayed and costly planners. We showed that, in a suite of five real-time, dynamic environments, our method produced agents that achieved higher return than heuristic meta-reasoning and delay-control policies. We then ablated our method and showed the importance of each major component on reward and success rate. Lastly, we analyzed the behavior of the trained agents on one of our domains to characterize the meta-reasoning policy it learns.

AI USE STATEMENT

In this work, we used generative AI tools for assisting in the formatting of proofs, implementing methods, supporting qualitative and thematic data analysis. Generative AI was specifically used to format the latex in the proof, write environment code, and write evaluation data plotting code. We have not used generative AI tools for helping develop theoretical models or conceptual frameworks, designing or providing feedback on research methodology or experiments, formulating mathematical claims, providing critical ingredients for proving mathematical claims, proposing or refine hypotheses, interpreting results and generating synthetic data sets, assisting with translation, cleaning and reformat dataset, are not applicable to this work. Additionally, we used generative AI tools for creating or modifying scientific figures or images, creating or editing software code, creating of artifacts, brainstorming, and editing to improve readability. Specifically, generative AI was used to produce the latex table formatting, editing environment code, and checking for grammar errors. We have reviewed all AI-assisted work. We checked or rewritten all LLM-generated text for readability. All code was verified and tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

All code used in experiments will be released upon acceptance.

REFERENCES

Ivan Anokin, Rishav Rishav, Matthew Riemer, Stephen Chung, Irina Rish, and Samira Ebrahimi Kahou. HANDLING DELAY IN REAL-TIME REINFORCEMENT LEARNING. 2025.

M. Bergamaschi Ganapini, M. Campbell, F. Fabiano, L. Horesh, J. Lenchner, A. Loreggia, N. Mattei, F. Rossi, B. Srivastava, and K. B. Venable. Fast, slow, and metacognitive thinking in AI. *npj Artificial Intelligence*, 1(1):27, October 2025. ISSN 3005-1460. doi: 10.1038/s44387-025-00027-5. URL https://www.nature.com/articles/s44387-025-00027-5.

Matthew Budd, Bruno Lacerda, and Nick Hawes. Stop! Planner Time: Metareasoning for Probabilistic Planning Using Learned Performance Profiles. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(18):20053–20060, March 2024. ISSN 2374-3468. doi: 10.1609/aaai.v38i18.29983. URL https://ojs.aaai.org/index.php/AAAI/article/view/29983.

Peter Böhm, Pauline Pounds, and Archie C. Chapman. Non-blocking Asynchronous Training for Reinforcement Learning in Real-World Environments. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10927–10934, October 2022. doi: 10.1109/IROS47612.2022.9981333. URL https://ieeexplore.ieee.org/abstract/document/9981333. ISSN: 2153-0866.

Wouter Caarls and Erik Schuitema. Parallel Online Temporal Difference Learning for Motor Control. *IEEE Transactions on Neural Networks and Learning Systems*, 27(7):1457–1468, July 2016. ISSN 2162-2388. doi: 10.1109/TNNLS.2015.2442233. URL https://ieeexplore.ieee.org/abstract/document/7131566.

Frederick Callaway, Sayan Gul, Paul M. Krueger, Thomas L. Griffiths, and Falk Lieder. Learning to select computations, August 2018. URL http://arxiv.org/abs/1711.06892. arXiv:1711.06892 [cs].

Michael Cashmore, Andrew Coles, Bence Cserna, Erez Karpas, Daniele Magazzeni, and Wheeler Ruml. Replanning for Situated Robots. *Proceedings of the International Conference on Automated Planning and Scheduling*, 29:665–673, July 2019. ISSN 2334-0843. doi: 10.1609/icaps.v29i1.3534. URL https://ojs.aaai.org/index.php/ICAPS/article/view/3534.

Valentin Charraut, Waël Doulazmi, Thomas Tournaire, and Thibault Buhet. V-Max: A Reinforcement Learning Framework for Autonomous Driving, July 2025. URL http://arxiv.org/abs/2503.08388. arXiv:2503.08388 [cs.LG].

Baiming Chen, Mengdi Xu, Zuxin Liu, Liang Li, and Ding Zhao. Delay-Aware Multi-Agent Reinforcement Learning for Cooperative and Competitive Environments, August 2020. URL http://arxiv.org/abs/2005.05441. arXiv:2005.05441 [cs.LG].

Baiming Chen, Mengdi Xu, Liang Li, and Ding Zhao. Delay-aware model-based reinforcement learning for continuous control. *Neurocomputing*, 450:119–128, August 2021. ISSN 0925-2312. doi: 10.1016/j.neucom.2021.04.015. URL https://www.sciencedirect.com/science/article/pii/S0925231221005427.

Robert Cohn, Edmund Durfee, and Satinder Singh. Comparing Action-Query Strategies in Semi-Autonomous Agents. *Proceedings of the AAAI Conference on Artificial Intelligence*, 25(1):1102–1107, August 2011. ISSN 2374-3468. doi: 10.1609/aaai.v25i1.7992. URL https://ojs.aaai.org/index.php/AAAI/article/view/7992.

Robert Cohn, Edmund Durfee, and Satinder Singh. Planning Delayed-Response Queries and Transient Policies under Reward Uncertainty. 2012.

Robert Cohn, Satinder Singh, and Edmund Durfee. Characterizing EVOI-Sufficient k-Response Query Sets in Decision Problems. 2014.

Andrew Coles, Erez Karpas, Andrey Lavrinenko, Wheeler Ruml, Solomon Eyal Shimony, and Shahaf Shperberg. Planning and Acting While the Clock Ticks. *Proceedings of the International Conference on Automated Planning and Scheduling*, 34:95–103, May 2024. ISSN 2334-0843. doi: 10.1609/icaps.v34i1.31465. URL https://ojs.aaai.org/index.php/ICAPS/article/view/31465.

Sombit Dey, Assem Sadek, Gianluca Monaci, Boris Chidlovskii, and Christian Wolf. Learning Whom to Trust in Navigation: Dynamically Switching Between Classical and Neural Planning. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5235–5242, October 2023. doi: 10.1109/IROS55552.2023.10342308. URL https://ieeexplore.ieee.org/document/10342308/. ISSN: 2153-0866.

Austin Dionne, Jordan Thayer, and Wheeler Ruml. Deadline-Aware Search Using On-Line Measures of Behavior. *Proceedings of the International Symposium on Combinatorial Search*, 2(1):39–46, 2011. ISSN 2832-9163. doi: 10.1609/socs.v2i1.18199. URL https://ojs.aaai.org/index.php/SOCS/article/view/18199.

Keliang Du, Luhan Wang, Yu Liu, Haiwen Niu, Shaoxin Huang, and Xiangming Wen. Random-Delay-Corrected Deep Reinforcement Learning Framework for Real-World Online Closed-Loop Network Automation. *Applied Sciences*, 12(23):12297, January 2022. ISSN 2076-3417. doi: 10.3390/app122312297. URL https://www.mdpi.com/2076-3417/12/23/12297.

Gabriel Dulac-Arnold, Nir Levine, Daniel J. Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. An empirical investigation of the challenges of real-world reinforcement learning, March 2021. URL http://arxiv.org/abs/2003.11881. arXiv:2003.11881 [cs.LG].

Amihay Elboher, Ava Bensoussan, Erez Karpas, Wheeler Ruml, Shahaf S. Shperberg, and Solomon E. Shimony. A Formal Metareasoning Model of Concurrent Planning and Execution, March 2023. URL http://arxiv.org/abs/2303.02664. arXiv:2303.02664 [cs].

Gongfan Fang, Xinyin Ma, and Xinchao Wang. Thinkless: LLM Learns When to Think, June 2025. URL http://arxiv.org/abs/2505.13379. arXiv:2505.13379 [cs].

C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation, 2021. URL http://github.com/google/brax.

Christian Fritz and Sheila McIlraith. Generating Optimal Plans in Highly-Dynamic Domains, May 2012. URL http://arxiv.org/abs/1205.2647. arXiv:1205.2647 [cs].

Cole Gulino, Justin Fu, Wenjie Luo, George Tucker, Eli Bronstein, Yiren Lu, Jean Harb, Xinlei Pan, Yan Wang, Xiangyu Chen, John D. Co-Reyes, Rishabh Agarwal, Rebecca Roelofs, Yao Lu, Nico Montali, Paul Mougin, Zoey Yang, Brandyn White, Aleksandra Faust, Rowan McAllister, Dragomir Anguelov, and Benjamin Sapp. Waymax: An Accelerated, Data-Driven Simulator for Large-Scale Autonomous Driving Research, October 2023. URL http://arxiv.org/abs/2310.08710. arXiv:2310.08710 [cs.RO].

Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay Policy Learning: Solving Long-Horizon Tasks via Imitation and Reinforcement Learning, October 2019. URL https://arxiv.org/abs/1910.11956v1.

Jessica B. Hamrick, Andrew J. Ballard, Razvan Pascanu, Oriol Vinyals, Nicolas Heess, and Peter W. Battaglia. Metacontrol for Adaptive Imagination-Based Optimization, May 2017. URL https://arxiv.org/abs/1705.02670v1.

Josiah P. Hanna and Nicholas E. Corrado. When Can Model-Free Reinforcement Learning be Enough for Thinking?, October 2025. URL http://arxiv.org/abs/2506.17124. arXiv:2506.17124 [cs].

Todd Hester and Peter Stone. TEXPLORE: real-time sample-efficient reinforcement learning for robots. *Machine Learning*, 90(3):385–429, March 2013. ISSN 0885-6125, 1573-0565. doi: 10.1007/s10994-012-5322-7. URL http://link.springer.com/10.1007/s10994-012-5322-7.

Todd Hester, Michael Quinlan, and Peter Stone. RTMBA: A Real-Time Model-Based Reinforcement Learning Architecture for robot control. In *2012 IEEE International Conference on Robotics and Automation*, pp. 85–90, May 2012. doi: 10.1109/ICRA.2012.6225072. URL https://ieeexplore.ieee.org/document/6225072. ISSN: 1050-4729.

Martin Klissarov, Akhil Bagaria, Ziyan Luo, George Konidaris, Doina Precup, and Marlos C. Machado. Discovering Temporal Structure: An Overview of Hierarchical Reinforcement Learning, June 2025. URL http://arxiv.org/abs/2506.14045. arXiv:2506.14045 [cs.AI].

Linh Kästner, Johannes Cox, Teham Buiyan, and Jens Lambrecht. All-in-One: A DRL-based Control Switch Combining State-of-the-art Navigation Planners, September 2021. URL http://arxiv.org/abs/2109.11636. arXiv:2109.11636 [cs].

Adam Labiosa and Josiah P. Hanna. When to Plan: Learning to Select Between Reactive Control and Deliberative Planning. 2026.

Donghun Lee, Bongseok Kim, Soonhyung Kwon, Ngoc-Duc Nguyen, Min Kyu Sim, and Young Il Lee. Reinforcement Learning-Based Control of DC-DC Buck Converter Considering Controller Time Delay. *IEEE Access*, 12:118442–118452, 2024. ISSN 2169-3536. doi: 10.1109/ACCESS.2024.3448535. URL https://ieeexplore.ieee.org/document/10646892/.

Solange Lemai and Felix Ingrand. Interleaving Temporal Planning and Execution in Robotics Domains.

Christopher H. Lin, Andrey Kolobov, Ece Kamar, and Eric Horvitz. Metareasoning for Planning Under Uncertainty, May 2015. URL http://arxiv.org/abs/1505.00399. arXiv:1505.00399 [cs].

Pierre Liotet, Erick Venneri, and Marcello Restelli. Learning a Belief Representation for Delayed Reinforcement Learning. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, July 2021. doi: 10.1109/IJCNN52387.2021.9534358. URL https://ieeexplore.ieee.org/abstract/document/9534358. ISSN: 2161-4407.

Pierre Liotet, Davide Maran, Lorenzo Bisi, and Marcello Restelli. Delayed Reinforcement Learning by Imitation. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 13528–13556. PMLR, June 2022. URL https://proceedings.mlr.press/v162/liotet22a.html.

Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Jackson, Samuel Coward, and Jakob Foerster. Craftax: A Lightning-Fast Benchmark for Open-Ended Reinforcement Learning, June 2024. URL http://arxiv.org/abs/2402.16801. arXiv:2402.16801 [cs.LG].

Alberto Maria Metelli, Flavio Mazzolini, Lorenzo Bisi, Luca Sabbioni, and Marcello Restelli. Control Frequency Adaptation via Action Persistence in Batch Reinforcement Learning. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 6862–6873. PMLR, November 2020. URL https://proceedings.mlr.press/v119/metelli20a.html.

Ofir Nachum, Shixiang (Shane) Gu, Honglak Lee, and Sergey Levine. Data-Efficient Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/hash/e6384711491713d29bc63fc5eeb5ba4f-Abstract.html.

Nigamaa Nayakanti, Rami Al-Rfou, Aurick Zhou, Kratarth Goel, Khaled S. Refaat, and Benjamin Sapp. Wayformer: Motion Forecasting via Simple & Efficient Attention Networks, July 2022. URL http://arxiv.org/abs/2207.05844. arXiv:2207.05844 [cs.CV].

Adrian Orenstein, Jessica Chen, Gwyneth Anne Delos Santos, Bayley Sapara, and Michael Bowling. Toward Agents That Reason About Their Computation, October 2025. URL http://arxiv.org/abs/2510.22833. arXiv:2510.22833 [cs].

Davide Paglieri, Bartłomiej Cupiał, Jonathan Cook, Ulyana Piterbarg, Jens Tuyls, Edward Grefenstette, Jakob Nicolaus Foerster, Jack Parker-Holder, and Tim Rocktäschel. Learning When to Plan: Efficiently Allocating Test-Time Compute for LLM Agents, September 2025. URL http://arxiv.org/abs/2509.03581. arXiv:2509.03581 [cs].

Simon Ramstedt and Christopher Pal. Real-Time Reinforcement Learning, December 2019. URL http://arxiv.org/abs/1911.04448. arXiv:1911.04448 [cs.LG].

Matthew Riemer, Gopeshh Subbaraj, Glen Berseth, and Irina Rish. ENABLING REALTIME REINFORCEMENT LEARNING AT SCALE WITH STAGGERED ASYNCHRONOUS INFERENCE. 2025.

C. Nicolò De Sabbata, Theodore R. Sumers, Badr AlKhamissi, Antoine Bosselut, and Thomas L. Griffiths. Rational Metareasoning for Large Language Models, June 2025. URL http://arxiv.org/abs/2410.05563. arXiv:2410.05563 [cs].

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017. URL http://arxiv.org/abs/1707.06347. arXiv:1707.06347 [cs].

Vishnu Dutt Sharma, Jeongran Lee, Matthew Andrews, and Ilija Hadžic. Hybrid Classical/RL Local Planner for Ground Robot Navigation. 2025.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, second edition, 2018. ISBN 9780262039246.

Devin Wild Thomas, Wheeler Ruml, and Solomon Eyal Shimony. Real-time Safe Interval Path Planning. *Proceedings of the International Symposium on Combinatorial Search*, 17:161–169, June 2024. ISSN 2832-9163. doi: 10.1609/socs.v17i1.31554. URL https://ojs.aaai.org/index.php/SOCS/article/view/31554.

Jaden B. Travnik, Kory W. Mathewson, Richard S. Sutton, and Patrick M. Pilarski. Reactive Reinforcement Learning in Asynchronous Environments. *Frontiers in Robotics and AI*, 5, June 2018. ISSN 2296-9144. doi: 10.3389/frobt.2018.00079. URL https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2018.00079/full.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. FeUdal Networks for Hierarchical Reinforcement Learning, March 2017. URL http://arxiv.org/abs/1703.01161. arXiv:1703.01161 [cs.AI].

Yule Wen, Yixin Ye, Yanzhe Zhang, Diyi Yang, and Hao Zhu. Real-Time Reasoning Agents in Evolving Environments, November 2025. URL http://arxiv.org/abs/2511.04898. arXiv:2511.04898 [cs.AI].

A Environment Specifications

In this section we describe the environments used for empirical experimentation.

A.1 Evade

Evade is a 120×120 grid where the agent must reach a goal (orange). The agent's actions are $\mathcal{A}^e = \{\text{Up, Down, Right, Left, Stay}\}$. The observation window is 11×11 with channels of At reset, a randomly drawn set of obstacles (gray) is drawn such that obstacles cover 30% of the open cells. There are 15 guards (red) which are 15×15 cells each. Each guard has a patrol loop that is a square of 12 steps. All guards share a single movement clock in which at a uniform time between 60 and 120 steps they all move to the next corner of their patrol loop (light red). Guards move at half the speed of the agent. Additionally, there are 9 pursuer of size 1×1 in the world (purple). The pursuers move at half of the speed of the agent and take a step to attempt to decrease the distance between themselves and the agent. Guards and pursuers can move over obstacles while the agent cannot. The planner (green) routes to the goal and avoids both obstacles and pursuers. The reward is summarized as follows: 0.01 * plan-follow - 0.02 per step - 0.5 per request - 5.0 on collision + 10.0 on reaching the goal.

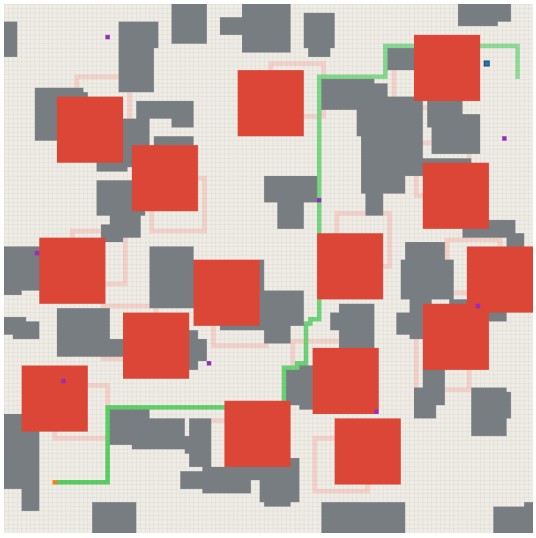


Figure 3: Example render of Evade.

A.2 Flood

In Flood, the agent moves through a series of chained T-mazes to reach a goal (orange). The agent's actions are $\mathcal{A}^e = \{\text{Up, Down, Right, Left, Stay}\}$. The observation window is 5×5 with channels of walls, the goal, and the flood. Each hallway has a width 3, with the first horizontal hallway having a length of 40, and all other horizontal hallways after that a length of 16 to 30 uniformly drawn. Each vertical hallways has a length of 25. There are 2 to 4 branches in each map.

There is a lethal flood behind the agent and if it catches the agent the episode ends. The flood (purple) moves at the same speed as the agent, and the agent has a head start of 20 steps. The plan (green) tells and agent which direction to walk at the upcoming intersection but only goes to the end of the current hallway, where a new plan has to be requested for the next intersection. The reward is summarized as follows: 0.05 * plan-follow - 0.1 per request - 5.0 on failure + 10.0 on the goal.

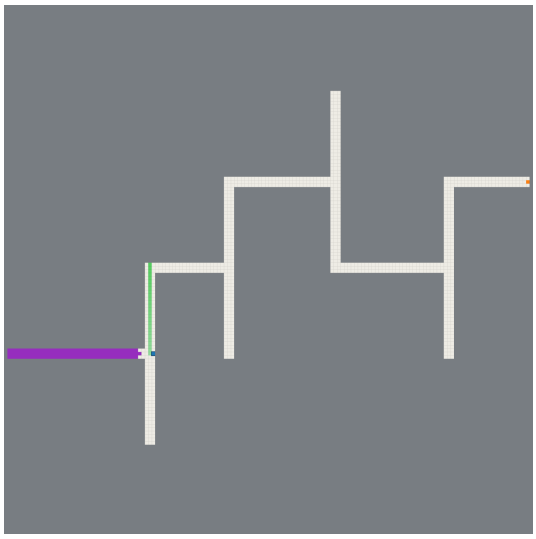


Figure 4: Example render of Flood.

A.3 SALVAGE

Salvage is a 120×120 grid of open space. The agent's actions are $\mathcal{A}^e = \{\text{Up, Down, Right, Left, Stay}\}$. The observation window is 13×13 with channels of walls, items, and asteroids. In the grid are placed 5 clusters with centers at least 35 cells apart where each cluster has 4 goal items (orange) associated with it. In the environment there are also twelve 3×3 asteroids (red) in the environment. Each is initialized to a random velocity and direction and reflects off of all of the walls. Contacting an asteroid ends the episode. The reward is summarized as follows: 0.04 * plan-follow - 0.02 per step + 1.0 per item - 0.1 per request - 5.0 on failure + 10.0 when every item is collected.

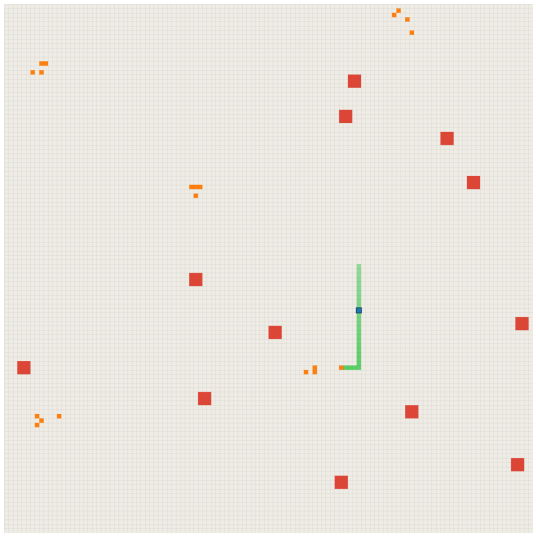


Figure 5: Example render of Salvage.

A.4 WAREHOUSE

Warehouse is a 73×73 grid of cells where the agent must reach a goal (orange). The agent's actions are $\mathcal{A}^e = \{\text{Up, Down, Right, Left, Stay}\}$. The observation window is 5×5 with channels of walls, doors, other agents, other agent's in the previous timestep, the goal. In the world there are

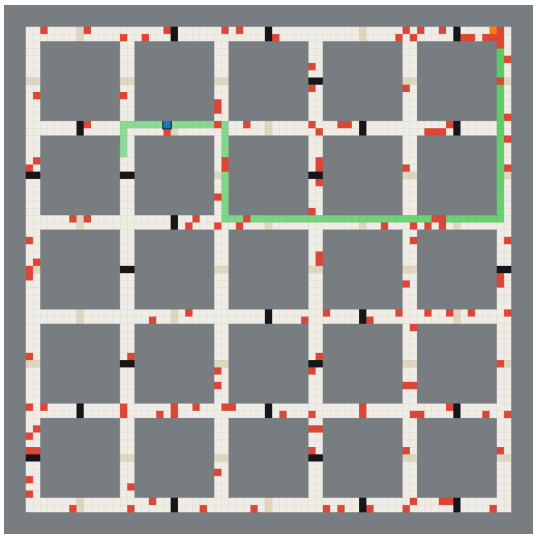


Figure 6: Example render of Warehouse.

6 rows by 6 columns of hallways for the agents to walk in. Each lane is two cells wide and all other agents except for the learned agent move on the right side of the lane. There are 150 of the non-learned agents (red) which randomly walk on the lane graph, at each intersection choosing a random direction to move. There is a single door (black) at the middle of each hallway, where a closed door blocks passage through that hallway. At all times, there are 25 doors closed but the closing is setup such that there is always a path between any two points. At every step, with probability 1/60, the door layout and regenerated, so the door configuration holds for a N steps and then is recreated. The planner (green) routes the agent to the goal from the request location following the lane rules. The reward components are summarized as follows: 0.01 * plan-follow - 0.02 per step - 0.5 per request - 5.0 on robot collision + 10.0 on reaching the goal.

A.5 WAYMAX

Waymax Gulino et al. (2023) is a self driving simulator developed by Waymo. The agent has 35 actions available, 5 acceleration options and 7 steering options. The observation includes other agents, the previous positions, paths, the roadgraph, and traffic lights. In it, real 9-second self-driving car logs are replayed at 10Hz. In each log, there is a set of paths the agent can drive on called SDC paths. Contacting another agent or leaving the lane causes the episode to end. At the beginning of each episode, we randomly sample a path as the ”goal” path for our agent to drive on, and randomly sample a starting location from the beginnings of the other possible paths. Additionally, as we only use 100 scenes, to prevent simple memorization, we randomly sample a speed for each logged agent to run at from 0.6x to 1.4x speed. The planner routes the agent from its current plan onto the chosen SDC path and avoids other vehicles. To add meta-reasoning pressure, we add in a car traveling slowly at some point in the log in front of our vehicle from 1 to 5 second into the 9 second episode. After the agent reaches that point, we resample a new SDC path, i.e., a new lane, for the agent to follow. However, at that point the agent does not know the correct lane and thus, must request a new plan to lead to the new lane. Success is defined if the agent makes it more than half way on the SDC path while being in the correct lane by episode end. The reward is built on the reward in V-Max (Charraut et al., 2025) with the addition of the plan following reward, the plan requesting penalty, and the success bonus.

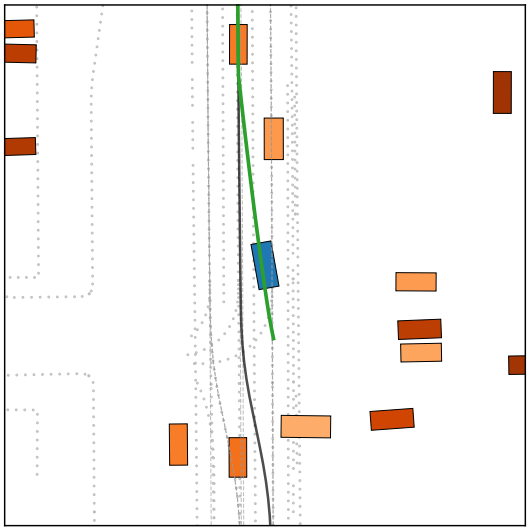


Figure 7: Example render of Waymax.

	Warehouse	Evade	T-maze	Salvage	Waymax
Env steps	75M	75M	75M	75M	200.0M
Plan delay (steps)	15	15	15	15	10
Max episode steps	500	500	600	500	80
Movement actions	5	5	5	5	35
Training seeds	10	10	10	10	5

Table 5: Per-environment training budget and episode setup. All gridworlds were trained with seeds 10-19 and Waymax was trained with seeds 15-19.

	Warehouse	Evade	T-maze	Salvage	Waymax
Plan request charge	0.5	0.5	0.1	0.5	0.5
Step cost	0.02	0.02	0	0.02	0
Progress scale	0.01	0.01	0.05	0.04	0.10
Goal reward	10.0	10.0	10.0	10.0	10.0
Item reward	—	—	—	1.0	—
Collision penalty	5.0	5.0	5.0	5.0	5.0

Table 6: Reward and cost structure per environment.

	Warehouse	Evade	T-maze	Salvage	Waymax
SHORT	20	20	20	20	20
MEDIUM	40	40	30	40	40
LONG	80	80	40	80	60

Table 7: Baseline plan frequency hyperparamaters.

B When Does an Agent Learn to Plan Proof

Here we provide the proof for Proposition 1 in section 3.3. For convenience we repeat the assumptions and proposition again.

We analyze a state $\tilde{s}$ with no request in-flight, so $d = 0$. We make two assumptions on the policy π, over the set of δ transitions after $\tilde{s}$, during which a plan requested in $\tilde{s}$ has not yet arrived. We write π^e and π^ℓ for the control and planning components of π.

Assumption 1. The control action a^e does not depend on the delay counter d or the request state s^r, and when no request is in-flight the request action a^ℓ does not depend on the request state, so for all environment states $s^e, s^r, s^q \in \mathcal{S}^e$, all plans ℓ, and all $d, d' \in \mathbb{N}$,

$$\pi^e\big(s^e, (\ell, d, s^r)\big) = \pi^e\big(s^e, (\ell, d', s^q)\big) \tag{7}$$

and

$$\pi^\ell\big(s^e, (\ell, 0, s^r)\big) = \pi^\ell\big(s^e, (\ell, 0, s^q)\big). \tag{8}$$

Assumption 2. For the state $\tilde{s}$ with no request in-flight so $d = 0$ and and a control action a^e, let $\mathcal{W}(\tilde{s}, a^e)$ denote the set of states that can be reached from 1 to δ transitions when a^e is taken in $\tilde{s}$ and π followed after. Over this window, the policy requests no new plan so

$$\pi^\ell(\hat{s}) = \texttt{no-request} \quad \forall\, \hat{s} \in \mathcal{W}(\tilde{s}, a^e). \tag{9}$$

Proposition 1. *Let π be a policy in a real-time meta-reasoning MDP satisfying Assumptions 1 and 2. Let $\tilde{s} = (s^e, (\ell, 0, s^r))$ be a state with no request in-flight. Let $p^{\delta+1}_{\pi,\ell}(\cdot \mid s^e, a^e)$ denote the distribution of states when a plan is received $\delta + 1$ steps in the future, given that a^e is taken in s^e and $\pi^e(\cdot, \ell)$ is followed until then, and let $\ell' = f(s^e)$ denote the plan received. Suppose the policy improvement step of policy iteration sets $\pi'(\tilde{s}) \leftarrow (a^e, \texttt{request})$, so that the improved policy requests a plan but does not change the control action, then*

$$\gamma^{\delta+1}\, \mathbb{E}_{s'' \sim p^{\delta+1}_{\pi,\ell}(\cdot \mid s^e, a^e)}\big[v_\pi(s'', \ell') - v_\pi(s'', \ell)\big] \geq r^\ell. \tag{10}$$

Therefore, an improved policy will request a plan only if the discounted expected improvement over the current plan produces greater return than the immediate reward penalty.

Proof. Policy iteration sets $\pi'(\tilde{s}) \leftarrow \arg\max_{\tilde{a} \in \tilde{\mathcal{A}}} q_\pi(\tilde{s}, \tilde{a})$, where

$$q_\pi\big(\tilde{s}, (a^e, a^\ell)\big) = \begin{cases} r^e(s^e, a^e) + \gamma\, \mathbb{E}_{s' \sim p^e(\cdot \mid s^e, a^e)}\big[v_\pi(s', (\ell, 0, s^r))\big] & \text{if } a^\ell = \texttt{no-request}, \\ r^e(s^e, a^e) - r^\ell + \gamma\, \mathbb{E}_{s' \sim p^e(\cdot \mid s^e, a^e)}\big[v_\pi(s', (\ell, \delta, s^e))\big] & \text{if } a^\ell = \texttt{request}. \end{cases} \tag{11}$$

We refer to taking $(a^e, \texttt{request})$ in $\tilde{s}$ and following π thereafter as the planning policy, and to taking $(a^e, \texttt{no-request})$ in $\tilde{s}$ and following π thereafter as the non-planning policy.

We refer to a policy which is updated to request a plan in $\tilde{s}$ as the planning policy, and a policy which does not request a plan in $\tilde{s}$ as a non-planning policy. Let $S_0 = s^e$ and let $S_1, \ldots, S_{\delta+1}$ denote the environment states generated by taking a^e in s^e and then following $\pi^e(\cdot, \ell)$, so that $p^{\delta+1}_{\pi,\ell}(\cdot \mid s^e, a^e)$ is the distribution of $S_{\delta+1}$. By Assumption 2 neither policy requests a plan during the window, so no new plan arrives within it and both condition on ℓ throughout. By Assumption 1 both select the same control actions $\pi^e(\cdot, \ell)$, despite different delay counter and request state values. Since p^e depends only on the environment state and the control action, the two policies induce the same distribution over $S_1, \ldots, S_{\delta+1}$ and accumulate the same rewards over the window.

Let

$$G^{\mathrm{w}} \doteq \sum_{k=1}^{\delta} \gamma^{\,k-1}\, r^e\big(S_k, \pi^e(S_k, \ell)\big) \tag{12}$$

denote the discounted reward accumulated over the window.

At the end of the window the planning policy receives the updated plan $\ell' = f(s^e)$, while the non-planning policy still conditions on ℓ. Both then have $d = 0$ and no request in-flight, and by Assumption 1 neither conditions its control action on the request state. The value at $S_{\delta+1}$ therefore depends on the plan state only through the plan.

Let $\mathbb{E}_{\pi,\ell}$ denote expectation over $S_1, \ldots, S_{\delta+1}$. Separating the reward accumulated during the window from the expected value after it gives

$$\mathbb{E}_{s' \sim p^e(\cdot \mid s^e, a^e)}\big[v_\pi(s', (\ell, \delta, s^e))\big] = \mathbb{E}_{\pi,\ell}\big[G^{\mathrm{w}}\big] + \gamma^{\delta}\, \mathbb{E}_{s'' \sim p^{\delta+1}_{\pi,\ell}}\big[v_\pi(s'', \ell')\big], \tag{13}$$

$$\mathbb{E}_{s' \sim p^e(\cdot \mid s^e, a^e)}\big[v_\pi(s', (\ell, 0, s^r))\big] = \mathbb{E}_{\pi,\ell}\big[G^{\mathrm{w}}\big] + \gamma^{\delta}\, \mathbb{E}_{s'' \sim p^{\delta+1}_{\pi,\ell}}\big[v_\pi(s'', \ell)\big], \tag{14}$$

where $p^{\delta+1}_{\pi,\ell}$ abbreviates $p^{\delta+1}_{\pi,\ell}(\cdot \mid s^e, a^e)$.

Since policy improvement selected the planning action we have $q_\pi(\tilde{s}, (a^e, \texttt{request})) \geq q_\pi(\tilde{s}, (a^e, \texttt{no-request}))$. Substituting the two identities into the definition of q_π gives

$$\begin{aligned} r^e(s^e, a^e) - r^\ell + \gamma\, \mathbb{E}_{\pi,\ell}\big[G^{\mathrm{w}}\big] + \gamma^{\delta+1}\, \mathbb{E}_{s'' \sim p^{\delta+1}_{\pi,\ell}}\big[v_\pi(s'', \ell')\big] \\ \geq\; r^e(s^e, a^e) + \gamma\, \mathbb{E}_{\pi,\ell}\big[G^{\mathrm{w}}\big] + \gamma^{\delta+1}\, \mathbb{E}_{s'' \sim p^{\delta+1}_{\pi,\ell}}\big[v_\pi(s'', \ell)\big]. \end{aligned} \tag{15}$$

The immediate reward and the window reward term appear on both sides and cancel, and rearranging the remaining terms gives

$$\gamma^{\delta+1}\, \mathbb{E}_{s'' \sim p^{\delta+1}_{\pi,\ell}(\cdot \mid s^e, a^e)}\big[v_\pi(s'', \ell') - v_\pi(s'', \ell)\big] \geq r^\ell. \tag{16}$$

The planning action is therefore selected only if the expected improvement from the returned plan, discounted by the $\delta + 1$ transitions taken before receiving it, is at least the immediate cost of the request. □

C Additional Related Work

In this section we provide additional topics that relate to our work but are less relevant for understanding where our work fits into the broader literature.

Hierarchical Goal-Conditioned RL. Our method has structural similarities to hierarchical goal-conditioned RL (Klissarov et al., 2025), in which a high-level policy produces goals that a low-level policy is trained to achieve (Vezhnevets et al., 2017; Nachum et al., 2018; Gupta et al., 2019). However, standard hierarchical approaches generally assume that a new goal is created instantaneously, with no cost or delay between the request and its arrival. In our setting, invoking the planner incurs both a cost and a delay, during which the environment continues to change and the low-level policy must continue acting without an updated plan, and any plan received may be stale.

Learning to Query for Assistance

Requesting a plan from a delayed planner is similar to the idea of learning to query a human for assistance. Cohn et al. (2011) and Cohn et al. (2014) both look into what query should be asked from a human to maximize the accuracy of the reward function but do not consider delays, dynamic domains or using standard RL for learning such behavior. Cohn et al. (2012) consider what if there is a delay between when assistance is requested and returned and consider how to select a query and a behavior to use while waiting, while in this work we consider a single policy which acts while waiting and requests plans.

D TRAINING HYPERPARAMTERS

Hyperparameter	Value
Algorithm	
Algorithm	PPO (Brax)
Optimizer	Adam
Learning rate	3×10^{-4}
Discount γ	0.997
GAE λ	0.99
Clip range ϵ	0.2
Entropy coefficient	0.01
Plan-head entropy weight	1.0
Plan-head advantage weight	1.0
Max gradient norm	0.5
Reward scaling	1.0
Normalize observations	no
Normalize advantage	yes
Rollout and batching	
Parallel environments	2048
Unroll length	32
Batch size	256
Minibatches per batch	8
Epochs per batch	4
Env steps per PPO iteration	65,536
Evaluations per run	8
Deterministic evaluation	yes
Protocol	
Training seeds	10 gridworlds/5 Waymax
Evaluation episodes	2048 per seed

Table 8: Training hyperparameters shared by every environment and every method.

Component	Grid worlds	Waymax
Encoder	MLP (`flat`)	Wayformer (`vmax_wayformer`)
Encoder	256×3, ReLU	—
Policy head	256	256
Value head	256	256
Activation	ReLU	ReLU

Table 9: Network architecture for gridworlds and Waymax.