Learning to Act While Thinking

Adam Labiosa[1], **Josiah P. Hanna**[1]
labiosa@wisc.edu, jphanna@cs.wisc.edu

[1]**Department of Computer Sciences, University of Wisconsin–Madison, USA**

Abstract

Agents operating in the real world must balance fast, reactive control against slow, deliberative planning, especially when planning is costly and the environment keeps changing while a plan is computed. In this work, we study the problem of simultaneously acting with fast, reactive control and meta-reasoning about when to invoke a slow, asynchronous planner as the Real-Time Meta-Reasoning MDP. In this setting, the agent can, at any time, request a plan to help it accomplish its task. However, the plan is costly and arrives only after a delay, during which the environment continues to change. We propose a curriculum learning-based reinforcement learning (RL) method to train agents that simultaneously learn when to invoke the planner, how to act while waiting for a response, and how to follow the plan once it arrives. We evaluate our method in dynamic, real-time environments where the agent cannot accomplish the task without successfully applying all three behaviors. In two partially observable dynamic tasks we show that our method achieves a higher success rate and uses fewer plans than fixed meta-reasoning and heuristic baselines.

1 Introduction

In large, complex environments, a policy with finite capacity that directly maps observations to actions cannot capture everything necessary for effective control across the entire state space (Javed & Sutton, 2024). One way to mitigate such capacity limitations is to iteratively reason about future actions through planning or reasoning. Iterative reasoning allows for an agent to *compute* an optimal action as opposed to requiring the optimal action to *memorized*. Intuitively, an agent which can search or reason can spend computation to figure out action to take even if it does not already know. Much recent work has focused on this idea by scaling test-time reasoning and planning to improve performance (OpenAI et al., 2024; Geiping et al., 2025; Snell et al., 2025).

While test-time reasoning has produced impressive results, most work on scaling test-time compute treats additional computation as free. However, in the real-world planning takes time, as the environment does not pause while an agent computes. For example, a self-driving car whose planner takes even a second to reason will not be able to react fast enough to a pedestrian who walks into the road a short distance in front of it. This challenge exists whenever an agent must act in a continuously changing environment, from social navigation (Kruse et al., 2013), to autonomous driving (Wenda Xu et al., 2012), such domains are common in the real world. As a result, in real-time situations, naively scaling up compute may cause task failure or physical harm.

With these considerations in mind, the goal of this work is to produce agents which satisfy two real-world requirements. First, agents must act in real time, responding immediately as the environment changes. Second, agents must be able to invoke additional computation for deliberation when reactive control alone is insufficient. To enable such behaviors, we train RL agents to simultaneously reason about when to invoke slow, expensive planners and control actions in dynamic, real-time environments. Agents which perform well in such settings must fulfill three capabilities. First, they

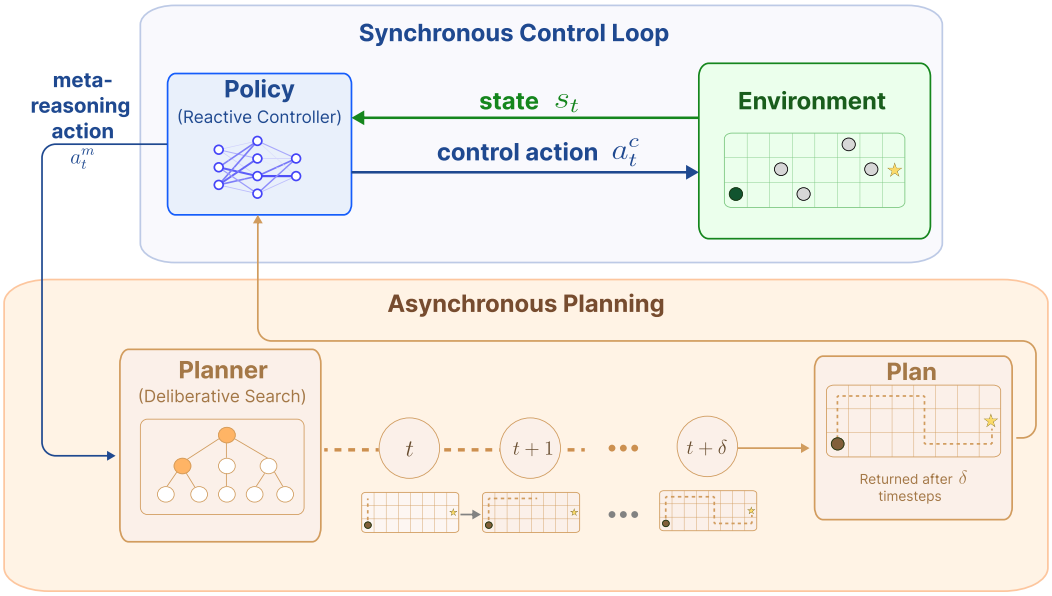


Figure 1: Real-Time Meta-Reasoning MDP visualization. This is the setting in which we instantiate our RL method to enable agent to operate in real-time domains with slow, asynchronous planners.

must correctly decide when to invoke expensive planning to maximize the performance benefit while minimizing the cost. Second, they must act effectively while waiting, as plans take time to arrive. Third, they must correctly implement the plan once it is available.

With these desiderata in mind we ask:

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 formalize a real-time meta-reasoning problem with slow planners as an MDP. We then introduce a method based on this MDP to train RL agents to selectively invoke a computationally expensive planner while interacting with the environment. We evaluate our method in two dynamic environments in which the agent must learn when to request a plan, how to act while waiting for it, and how to follow it once it arrives. Our experiments show that our method trains agents that both accomplish the dynamic tasks successfully and use less computation than fixed heuristics. In summary, we make the following contributions:

1. We formalize the real-time meta-reasoning problem as an MDP that jointly models reactive control, planner invocation, and the delay between requesting and receiving a plan.
2. We introduce an RL-based method which uses curriculum learning to learn when to invoke a slow planner, how to act effectively while waiting for plans, and how to follow the plans.
3. We empirically show our method achieves fewer collisions than heuristic waiting baselines and uses less computation than fixed planning baselines.

2 Background

In this section we describe MDPs, asynchronous planning, and meta-reasoning.

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, $P(s'|s,a)$ defines the transition dynamics, $R(s,a)$ 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 $\pi(a_t \mid s_t)$.

The environment transitions to a new state s_{t+1} 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: $J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$.

Asynchronous Planning. In classical planning settings, it is generally assumed that the environment pauses while the agent computes a plan and that execution begins once planning is complete (Ghallab et al., 2016). However, in real-time environments this assumption does not hold. Instead, time passes as plans are computed, a setting referred to as *asynchronous planning* (Gat, 1992). Formally, when a plan is requested at timestep t, it is not available until timestep $t + \delta$, where $\delta > 0$ is the *planning delay*. During the interval $[t, t + \delta]$, the environment continues to transition according to P. This delay introduces two challenges. First, the agent must act effectively during the delay interval without access to an updated plan. Second, when the plan arrives at timestep $t + \delta$, it was computed based on state s_t, which may no longer reflect the current state $s_{t+\delta}$, making the plan potentially stale upon arrival.

Meta-reasoning. Meta-reasoning is the process of reasoning about when or how to reason (Horvitz, 2013). It is motivated by the idea of *bounded rationality* (Simon, 1990), which states that because decision-making is constrained by time, information, and energy, agents must rely on approximate solutions rather than exact ones. However, bounded rationality does not specify how approximate a solution should be in a given situation, only that some approximation is necessary. Meta-reasoning addresses this challenge by allowing an agent to improve its performance by determining how much computation to allocate to a decision based on the demands of the current situation. In this work, we focus on meta-reasoning for when to invoke expensive computation.

3 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 in which the environment can change while planning occurs. 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) propose both a benchmark for real-time reasoning and an agent combining a faster action generator with a slower reasoner, but fix the amount of reasoning per planning step and do not address the meta-reasoning problem of 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.

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.

4 The Real-Time Meta-Reasoning MDP

We formalize reactive control with asynchronous planning as the *Real-Time Meta-Reasoning MDP* defined by the tuple $(\widehat{\mathcal{S}}, \widehat{\mathcal{A}}, \widehat{P}, \widehat{R}, \gamma)$, where $\widehat{\mathcal{S}}$ is the augmented state space, $\widehat{\mathcal{A}}$ is the augmented action space, $\widehat{P}$ are the augmented transition dynamics, $\widehat{R}$ is the augmented reward function, and γ is the discount factor as in a standard MDP (Section 2).

State Space. The augmented state space $\widehat{\mathcal{S}} = \mathcal{S}_e \times \mathcal{S}_p$ combines the base environment state space $\mathcal{S}_e$ and the plan state space $\mathcal{S}_p$. At each timestep the agent observes $\hat{s}_t := (s_t^e, p_t, w_t) \in \widehat{\mathcal{S}}$, where $s_t^e \in \mathcal{S}_e$ is the current environment state and $(p_t, w_t) \in \mathcal{S}_p$ are the most recent plan and the time remaining until a requested plan is returned. The plan p_t is the most recent plan returned by the planner, with $p_t = \emptyset$ if no plan has been received in the current episode. We leave the exact representation of p_t to be 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 wait counter w_t is the time remaining until a requested plan is returned, with $w_t = 0$ if no plan is currently being computed.

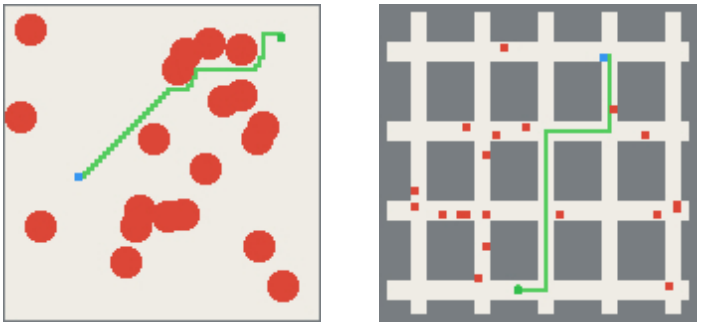
Action Space. The augmented action space $\widehat{\mathcal{A}} = \mathcal{A}_c \times \mathcal{A}_p$ combines the base environment control action space $\mathcal{A}_c$ and a binary plan action space $\mathcal{A}_p = \{0, 1\}$, which indicates whether to invoke the planner. At each timestep t, the agent selects a control action $a_t^c \in \mathcal{A}_c$ and a plan action $a_t^p \in \mathcal{A}_p$ according to a policy $\pi(a_t \mid \hat{s}_t)$, where $a_t = (a_t^c, a_t^p) \in \widehat{\mathcal{A}}$.

Planner and Planning Delay. The plan state transitions according to P_p, determined by the planner f and planning delay δ. In general, δ may be fixed or drawn from a distribution $\delta \sim D$ to model variable planner execution time. When the agent sets $a_t^p = 1$ at timestep t, the planner receives s_t^e and begins computing a plan, and the wait counter is set to $w_{t+1} = \delta$. This rule applies whether or not a plan is already in progress, so a new request made while $w_t > 0$ discards the in-progress computation and restarts the delay using the current state s_t^e. While $a_t^p = 0$ and $w_t > 0$, the wait counter decrements as $w_{t+1} = \max(w_t - 1, 0)$. When the wait counter reaches zero, the plan is returned and $p_{t+1} = f(s_t^e)$, using the environment state at the time the corresponding request was made, otherwise the plan remains unchanged, $p_{t+1} = p_t$. During the interval between a request and its corresponding delay δ, the agent must act without an updated plan. Upon arrival, the plan may be stale, as it was computed using the environment state at request time, which may differ from the environment state at arrival. The full transition dynamics $\widehat{P} = P_e \times P_p$ combine the base environment dynamics P_e with the plan transition dynamics P_p.

Reward. The augmented reward $\widehat{R}(\hat{s}_t, a_t) = R_e(s_t^e, a_t^c) - r^p \cdot a_t^p$ combines the base environment reward with a planning cost, where $r^p > 0$ is a fixed penalty incurred whenever the planner is invoked.

5 Reinforcement Learning with Delayed Computation

In this section, we describe our instantiation of a Real-Time Meta-Reasoning MDP and how we train agents to operate in domains with slow, asynchronous planners. We consider two domains



(a) Evade (b) Warehouse

Figure 2: Visual representation of the environments used in experiments. In all environments, the green square represents the agent, blue square the goal and green line is the plan produced by the planner. The red objects are all obstacles which cause failure upon contact, and the gray represents walls which the agent cannot pass through.

which are both discrete, vision-based environments. As such, we instantiate this MDP for discrete, vision-based environments, in which the plan is represented in the agent's image observation.

The joint action space $\widehat{\mathcal{A}}$ contains ten actions, combining five movement actions with two plan actions.

$$\widehat{\mathcal{A}} = \mathcal{A}_c \times \mathcal{A}_p = \{\text{Up, Down, Right, Left, Stay}\} \times \{0, 1\}$$

This action space allows the agent to move and invoke the planner within the same timestep.

We instantiate the observation space using a vision-based setup in which the agent receives a top-down view of its local surroundings as a multi-channel image of size $N \times N$, where $N < M$ and M is the size of the full environment. Each channel encodes a different aspect of the environment.

1. *Obstacles*: The positions of dynamic obstacles within the view area at the current timestep t.
2. *Previous Obstacles*: The positions of dynamic obstacles at the previous timestep $t - 1$, allowing the agent to understand obstacle movement direction.
3. *Goal*: The position of the goal if it falls within the observation area.
4. *Walls*: The positions of static walls in the environment.

To instantiate the plan and wait state $(p_t, w_t) \in \mathcal{S}_p$, we augment the base observation with three additional components. First, the plan p_t is provided as an image channel that renders the plan by interpolating the waypoints returned by the planner. Second, the wait counter is given as a normalized value $\tilde{w}_t = w_t/\delta \in [0, 1]$. Third, a binary indicator $\mathbb{1}[w_t > 0]$ is included to let the agent distinguish between a plan having just been requested, when $\tilde{w}_t$ is close to one, versus no plan having been requested at all, when $\tilde{w}_t$ is undefined or zero by convention.

Training an agent to simultaneously invoke the planner, follow returned plans, and act effectively during the planning delay is difficult from scratch, as it is challenging to learn the benefit of calling the planner when it has a high delay. The core challenge is that learning to invoke the planner and follow its output requires the agent to experience the benefit of planning. Under a large delay, however, the agent is unlikely to successfully invoke the planner, wait, and then follow the returned plan to reach a long-distance goal. To address this, we use a three-stage curriculum in which the planning delay is progressively increased. In the first stage, the agent is trained with a short delay, allowing it to learn to invoke the planner and follow plans without the added difficulty of acting during a long delay. In the second stage, the delay is increased to an intermediate value, requiring the agent to act before the plan arrives. In the final stage, the delay is increased to the target value δ^*, enabling the agent to learn the full waiting behavior required for dynamic environments. Specific delay values for each stage are provided in Section 6.

Experiment	Success Rate ↑	Collision Rate ↓	Plans/Ep. ↓
Freeze While Planning	$68.86 \pm 1.16\%$	$31.98 \pm 1.18\%$	2.69 ± 0.09
Avoid Obstacles While Waiting	$78.99 \pm 2.77\%$	$21.30 \pm 2.38\%$	4.20 ± 0.38
Fixed Frequency (10)	$90.02 \pm 1.11\%$	$11.71 \pm 1.14\%$	4.09 ± 0.06
Fixed Frequency (20)	$90.26 \pm 0.57\%$	$11.42 \pm 0.63\%$	3.24 ± 0.04
Fixed Frequency (40)	$89.61 \pm 0.61\%$	$12.25 \pm 0.58\%$	3.31 ± 0.13
Fixed Frequency (80)	$88.26 \pm 1.09\%$	$13.46 \pm 1.16\%$	$\mathbf{2.06 \pm 0.03}$
Full Method	$\mathbf{92.36 \pm 0.84\%}$	$\mathbf{8.87 \pm 0.68\%}$	2.41 ± 0.10

Table 1: Results on the **Warehouse** environment. ↑ indicates higher is better, ↓ lower is better. Success rate is the percentage of runs the goal is reached. Collision Rate is the percentage of runs the agent collides with an obstacle and Plans/Ep. is the average number of plans called in an episode. All experiments run for 20 seeds and evaluated on 2048 episodes post training. Confidence intervals are 95% CIs computed using the Student's *t*-distribution.

6 Experiments

In this section we detail our experimental setup and the empirical results.

6.1 Environments

We evaluate our method on two environments designed to test different aspects of asynchronous planning. First, the frequency with which replanning is required, second, the degree to which plans become stale, and third, the difficulty of acting effectively during the planning delay. Both environments test all three components, but do so in different ways. For specific environment parameters see Table 4.

1. **Warehouse**: The agent must navigate a warehouse-like environment shared with other agents performing independent tasks. The agent must follow its plan while avoiding collisions with the obstacle agents. The goal moves randomly during the episode, requiring the agent to recognize when its current plan is no longer valid and replan. This environment primarily tests reactive behavior, as many other agents move nearby while the agent waits for an updated plan.
2. **Evade**: The agent navigates a large open area toward a moving goal while avoiding large obstacles that move randomly. The planner computes paths around obstacles, but their movement can invalidate a plan during execution. The agent also must evade obstacles while waiting for updated plans. This environment primarily tests plan staleness, as obstacle movement may force replanning.

6.2 Empirical Setup

We train all agents using PPO (Schulman et al., 2017) with the Brax training library (Freeman et al., 2021). Hyperparameters are provided in Table 3. We train all agents using the delay curriculum described in Section 5, with planning delays of $\delta \in \{2, 10, 20\}$ trained for $\{10^7, 10^7, 3 \times 10^7\}$ steps respectively. All agents are evaluated at $\delta = \delta^* = 20$.

6.3 Baselines

We compare our agents against two types of baselines. First, we fix the agent's waiting behavior using heuristic waiting policies, which control how the agent acts during the planning delay. Second, we fix the agent's meta-reasoning behavior using heuristic replanning policies, which request plans at fixed intervals. All baselines are trained with the same curriculum as our method to isolate the contribution of learning each behavior.

Experiment	Success Rate ↑	Collision Rate ↓	Plans/Ep. ↓
Freeze While Planning	$74.03 \pm 2.13\%$	$25.57 \pm 2.10\%$	2.45 ± 0.22
Avoid Obstacles While Waiting	$84.87 \pm 1.26\%$	$13.88 \pm 1.07\%$	4.29 ± 0.33
Fixed Frequency (10)	$88.78 \pm 2.91\%$	$10.17 \pm 2.81\%$	4.35 ± 0.11
Fixed Frequency (20)	$\mathbf{89.57 \pm 0.81\%}$	$9.15 \pm 0.84\%$	3.50 ± 0.07
Fixed Frequency (40)	$87.89 \pm 5.71\%$	$11.11 \pm 5.80\%$	3.38 ± 0.12
Fixed Frequency (80)	$87.94 \pm 0.75\%$	$9.57 \pm 0.79\%$	$\mathbf{2.18 \pm 0.05}$
Full Method	$87.92 \pm 1.70\%$	$\mathbf{8.77 \pm 0.82\%}$	2.34 ± 0.08

Table 2: Results on the **Evade** environment. ↑ indicates higher is better, ↓ lower is better. Success rate is the percentage of runs the goal is reached. Collision Rate is the percentage of runs the agent collides with an obstacle and Plans/Ep. is the average number of plans called in an episode. All experiments run for 20 seeds and evaluated on 2048 episodes post training. Confidence intervals are 95% CIs computed using the Student's *t*-distribution.

Waiting policy baselines. We evaluate two heuristic waiting policies to assess the effectiveness of the learned waiting behavior. The *Freeze While Waiting* baseline executes a *Stay* action for the duration of the planning delay, so the agent does not move while the plan is computed. The *Avoid Obstacles While Waiting* baseline moves directly away from the nearest obstacle within its view range.

Meta-reasoning baselines. We evaluate a set of *Fixed Frequency* baselines to assess whether learned meta-reasoning reduces compute usage while maintaining task performance. Specifically, we test fixed replanning intervals of $N \in \{10, 20, 40, 80\}$ timesteps.

6.4 Evaluation Metrics

We evaluate all policies on three metrics. *Success rate*: the percentage of episodes in which the agent reaches the goal before truncation, without having triggered a collision. *Collision rate*: the percentage of episodes in which the agent contacts an obstacle, terminating the episode. *Plans per episode*: the number of planner calls per episode.

6.5 Empirical Analysis

In this section, we analyze the effectiveness of our method across both environments. Results are shown in Table 1 for the Warehouse environment and Table 2 for the Evade environment.

In the Warehouse environment, our method achieves a higher success rate and lower collision rate than both waiting policy baselines, demonstrating that the learned waiting behavior enables the agent to navigate effectively while waiting for plans. Our method also uses fewer plans per episode than all fixed-frequency baselines except *Fixed Frequency (80)*. While *Fixed Frequency (80)* uses fewer plans per episode than our method and the other fixed-frequency baselines, it achieves a lower success rate than our method. In contrast, our method learns to selectively invoke the planner at a low rate while maintaining the highest success rate among all methods evaluated. This result suggests that jointly learning meta-reasoning and reactive control is beneficial for both improved acting and plan-calling. Interestingly, *Avoid Obstacles While Waiting* invokes the planner at nearly double the rate of our method, yet it still achieves a significantly worse collision rate. This result suggests that while avoiding nearby obstacles may keep the agent safe during the waiting period, the lack of long-horizon reasoning can leave the policy in difficult situations once it regains control.

In the Evade environment, results are more mixed. Compared to the waiting policy baselines, our method achieves a significantly lower collision rate. Its success rate is comparable to *Avoid Obstacles While Waiting*, suggesting that the agent becomes cautious and prioritizes obstacle avoidance

at the cost of task completion, which causes episodes to time out rather than fail via collision. This may reflect a higher degree of plan staleness in this environment. Because adversaries can block planned paths and the goal can move substantially during the delay, the agent may learn to wait for updated plans rather than act on stale ones, resulting in overly conservative behavior. Compared to the fixed-frequency baselines, our method performs similarly across success rate and collision rate. While our method uses fewer plans per episode than all fixed-frequency baselines except *Fixed Frequency (80)*, it does not perform substantially better on success rate or collision rate in this environment. This result suggests our method is still able to jointly perform meta-reasoning and control successfully in Evade, though improving its performance relative to fixed-frequency baselines in this environment is left to future work.

7 Discussion and Future Work

Our empirical results demonstrate that RL agents can learn to perform both meta-reasoning and reactive control using slow planners in real-time domains. Here we identify limitations of this work and outline directions for future research.

First, our method's performance varies across environments. While it performs strongly in the Warehouse domain, achieving the highest success rate and lowest collision rate, in the Evade environment it becomes overly cautious by prioritizing avoiding collisions rather than task competion. Future work will explore method adjustments to better balance safety and task completion across domains.

Second, we evaluate on only two environments. While these environments are designed to test distinct aspects of the asynchronous planning problem, they are limited in scope. Both are gridworld-based, which isolates the core challenges but does not capture the nuance and complexity present in many real-world tasks. We plan to expand the evaluation to a larger and more diverse set of environments, including more realistic domains such as autonomous driving and robotic navigation, and to incorporate realistic slow planners such as diffusion-based motion planners or reasoning LLMs to evaluate our method beyond the gridworld setting.

Third, beyond verifying the effectiveness of the curriculum against fixed baselines, we do not run ablations on our method. Future work will investigate the contribution of individual components and the sensitivity of performance to each.

Fourth, we do not analyze the behavior learned by the policy. Future work should include behavioral analyses examining, for example, when and how often the agent chooses to replan throughout an episode, and how this varies with environment dynamics.

8 Conclusion

In this work, we presented a method for training agents that jointly perform meta-reasoning over slow planners and reactive control in real-time domains. We framed this problem as a Real-Time Meta-Reasoning MDP, and used this formalism to train RL agents that outperform heuristic baselines fixing either waiting behavior or replanning frequency. As agents are deployed in dynamic, real-world settings where computation is both slow and costly, enabling them to flexibly allocate computation while effectively handling a changing environment will enable more general and useful agents.

Acknowledgments

This work took place in the Prediction and Action Lab (PAL) at the University of Wisconsin–Madison. PAL research is supported by NSF (IIS-2410981) and the Wisconsin Alumni Research Foundation. Adam Labiosa is supported by an NSF Research Traineeship award, No. 2152163.

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.

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.

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].

Erann Gat. Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots. 1992.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach. In *Advances in Neural Information Processing Systems*, volume 38, pp. 41340–41391. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/3b01972cf31e6fa0fe29e4b8b5c2a0a1-Abstract-Conference.html.

Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning and acting*. Cambridge University Press, 2016.

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.

Eric J. Horvitz. Reasoning About Beliefs and Actions Under Computational Resource Constraints, March 2013. URL https://arxiv.org/abs/1304.2759v1.

Khurram Javed and Richard S Sutton. The Big World Hypothesis and its Ramifications for Artificial Intelligence. 2024.

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].

Thibault Kruse, Amit Kumar Pandey, Rachid Alami, and Alexandra Kirsch. Human-aware robot navigation: A survey. *Robotics and Autonomous Systems*, 61(12):1726–1743, December 2013. ISSN 0921-8890. DOI: 10.1016/j.robot.2013.05.007. URL https://www.sciencedirect.com/science/article/pii/S0921889013001048.

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.

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.

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.

OpenAI, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich,

Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever, Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñonero Candela, Joe Palermo, Joel Parish, Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus, Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk, Lukasz Kaiser, Luke Metz, Madelaine Boyd, Maja Trebacz, Manas Joglekar, Mark Chen, Marko Tintor, Mason Meyer, Matt Jones, Matt Kaufer, Max Schwarzer, Meghan Shah, Mehmet Yatbaz, Melody Y. Guan, Mengyuan Xu, Mengyuan Yan, Mia Glaese, Mianna Chen, Michael Lampe, Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang, Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowdhury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne, Pavel Izmailov, Peter Zhokhov, Rachel Dias, Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan Greene, Saachi Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng, Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou, Wes McCabe, Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai, Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. OpenAI o1 System Card, December 2024. URL http://arxiv.org/abs/2412.16720. arXiv:2412.16720 [cs].

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.

Herbert A. Simon. Bounded Rationality. In *Utility and Probability*, pp. 15–18. Palgrave Macmillan, London, 1990. ISBN 978-1-349-20568-4. DOI: 10.1007/978-1-349-20568-4_5. URL https://link.springer.com/chapter/âĂŒ10.1007/978-1-349-20568-4_5.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning. *International Conference on Learning Representations*, 2025:10131–10165, May 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b623663fd9b874366f3ce019fdfdd44-Abstract-Conference.html.

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].

Wenda Xu, Junqing Wei, John M. Dolan, Huijing Zhao, and Hongbin Zha. A real-time motion planner with trajectory optimization for autonomous vehicles. In *2012 IEEE International Conference on Robotics and Automation*, pp. 2061–2067, St Paul, MN, USA, May 2012. IEEE. ISBN 978-1-4673-1405-3 978-1-4673-1403-9 978-1-4673-1578-4 978-1-4673-1404-6. DOI: 10.1109/ICRA.2012.6225063. URL http://ieeexplore.ieee.org/document/6225063/.

Supplementary Materials

The following content was not necessarily subject to peer review.

9 Appendix

Hyperparameter	Value used	Brax default
Number of parallel environments	512	1
Learning rate	3×10^{-4}	1×10^{-4}
Discount factor γ	0.99	0.9
Entropy coefficient	0.0	1×10^{-4}
Rollout length	32	10
Batch size	256	32
Number of minibatches	8	16
Updates per batch	4	2
PPO clipping parameter ϵ	0.2	0.3
Number of evaluations	10	1
Deterministic evaluation	true	false

Table 3: PPO hyperparameters which differ from Brax PPO defaults.

Parameter	**Warehouse**	**Evade**
Episode horizon	500 steps	500 steps
World size	160×160	160×160
Agent speed	2 units/step	2 units/step
Moving obstacle type	Other robots	Circular blobs
Number of obstacles	20 robots	20 blobs
Obstacle size	Agent-sized robots	Radius 6.0
Obstacle speed	1 unit/step	1 unit/step
Obstacle movement start rate	Continuous	$p = 0.25$ per idle step
Obstacle movement duration	N/A	5–20 steps
Goal movement start rate	$p = 0.25$ per idle step	$p = 0.25$ per idle step
Goal movement duration	10–20 steps	5–10 steps
Goal movement speed	1 unit/step	1 unit/step
Goal size	4×4	4×4

Table 4: Environment parameters for the agent, the obstacles and the goal.