# Think Fast! Learning to Control Online Reasoning in Stochastic Environments

Matthew Budd
Stateful Robotics
Oxford, United Kingdom
matt@statefulrobotics.com

Bruno Lacerda
Stateful Robotics
Oxford, United Kingdom
bruno@statefulrobotics.com

Nick Hawes
University of Oxford
Oxford, United Kingdom
nickh@robots.ox.ac.uk

## ABSTRACT

When an autonomous agent's decision-making has resource costs or incurs potential real-world consequences, its performance can be improved by reasoning about its own decision-making process. This is known as *metareasoning*, and is a key capability of rational agents. However, existing metareasoning methods have significant limitations. Most apply only to the *offline* setting, controlling only how long the agent should think before executing its current best solution. Few methods exist for *online* metareasoning, where the agent can interleave thinking and acting, and these make strong simplifying assumptions that limit their performance. It is rarer still for methods to be applicable to stochastic problems, or to consider the effects of the environment on the agent's planning process.

In this work we extend a learning-based metareasoning method for probabilistic planning to the online setting. The framework enables the agent to learn *when*, *where* and *how* to think in order to make better decisions in stochastic environments. We demonstrate our method outperforming several baselines across two domain distributions, each highlighting different benefits of online metareasoning.

## KEYWORDS

Markov Decision Processes; Planning; Reinforcement Learning

**ACM Reference Format:**
Matthew Budd, Bruno Lacerda, and Nick Hawes. 2026. Think Fast! Learning to Control Online Reasoning in Stochastic Environments. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 9 pages. https://doi.org/10.65109/MXNX1909

## 1 INTRODUCTION

Autonomous agents often leverage online computation to surpass the performance attainable through offline pre-training alone [27, 29]. For sequential decision-making, this is implemented using the *decision-time planning* paradigm, where the agent performs planning computation before each action [5, 13]. However, optimally balancing the time spent planning and acting is a challenging problem. In practice, this parameter is usually set to a fixed value by the designer, based on their knowledge of the agent's capabilities and the types of problems the agent will encounter. In this work, we aim to enable agents to learn how best to balance planning and acting, given the current state of the agent and the environment.

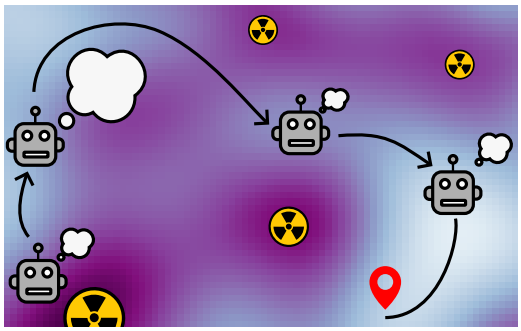


**Figure 1: A mobile robot plans and executes a path across a radioactive environment. Using online metareasoning, it can pause to plan for longer (larger thought bubbles) in safer areas so that it can move continuously through hazardous zones without planning in them.**

We pose the problem of balancing planning and acting as a *metalevel control* problem [15]. In this problem formulation, a *metalevel* agent supervises the *object-level* algorithm carrying out the decision-making process. The metalevel algorithm may reason about when to allow the object-level algorithm to continue planning, and when to act using the current solution. It may also change *how* the object-level algorithm reasons, by selecting hyperparameters or even different algorithms that are better suited to the current problem or situation [6, 8]. This enables trading off rapid initial improvement against thorough long-term exploration based on the current state and available planning budget. *Anytime* [30] object-level algorithms are ideal in this setting, as they can be queried for their current solution at any point in time.

Metalevel control is particularly useful when real-world decision-making is subject to costs and constraints on both acting and planning, often drawing from shared resources. Planning and acting both take time and energy: this is particularly important for mobile robots, where a finite battery reserve must be shared between planning and acting, and computation constraints make planning time a meaningful consideration.

The costs of reasoning may also depend on the agent's state within an environment: Figure 1 illustrates an example setting inspired by Budd et al. [7] where a mobile robot incurs varying rates of radiation exposure while operating in a radioactive environment. In this setting, it can be difficult to specify an optimal planning duration that is long enough for good decision-making but short enough to avoid excessive radiation exposure while planning [7].



*Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus.* © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/MXNX1909

The cost-optimal metalevel behaviour in this setting is to pause to plan only in low radiation areas in order to take advantage of lower planning costs. In other domains, such as underwater robotics in areas with strong water currents, pausing to plan may even cause the agent's state to change due to environmental dynamics.

The simplest metalevel control problem is to decide the length of time to spend planning, before proceeding to fully execute the current best solution. We refer to this as *offline* metareasoning, as no planning takes place during the execution period. Offline metareasoning methods aim to solve the *how to plan* problem, by determining the optimal duration of the planning period and potentially which hyperparameters to use for the planning process [6]. In this work we study the more general case of *online* metareasoning, where the agent interleaves planning and acting. As well as broadening the applicability of metareasoning to controlling a wider class of object-level algorithms, we also wish to demonstrate several metareasoning-specific advantages of online metareasoning.

First, an online metalevel agent can decide *when* best to plan, based on its current knowledge of the environment. Decision-time planning improves performance by focusing search effort on the current state [5], and by adapting behaviour to the actual environment parameters encountered at test time, thereby better addressing epistemic (i.e. model) uncertainty [7]. Similarly, online metareasoning enables adapting *reasoning behaviour* to the environment encountered: rather than planning for all possible situations, the agent can reduce decision-making costs by deferring in-depth planning until it has more information about the environment. For example, consider a robot navigating a building with internal doors it cannot open: if possible, it is more efficient to observe whether the door is open before planning a policy that includes paths passing through the door.

Second, online metareasoning enables optimisation of *where* to plan, rather than being limited to planning at the initial state. Planning may incur more cost at some states than others, as illustrated by the radiation exposure example. In some states planning may be unsafe or not be possible at all: for example, autonomous underwater vehicles often offload their planning to a remote server via satellite, so can only plan while at the water surface.

To the best of our knowledge, the method we present is the first to jointly consider the *when*-, *where*- and *how*-to-plan aspects of metalevel control within a single framework. We achieve this by extending a learning-based metareasoning method for probabilistic planning to the online setting. By learning the behaviour of object-level algorithms from data, the framework applies to any object-level algorithm and problem distribution. We demonstrate our method's performance in domains which highlight the different benefits of online metareasoning. Our method outperforms existing online metalevel controllers, both adaptive and fixed, and a state-of-the-art offline metareasoning method [8].

## 2 RELATED WORK

Due to the intractability of exact metareasoning [25], no generally applicable exact algorithm exists. Metalevel control methods therefore differ by what type of object-level algorithm they supervise and the assumptions they make. The difficulty of metareasoning means that algorithms must be cognisant of the overhead of metareasoning [19, 22]. Effort spent on metareasoning should be less than the effort saved as a consequence of using metareasoning, or the net effect is negative.

Our focus is on high-level control over an agent's reasoning process during interaction with a stochastic environment. We therefore take a metalevel control approach, and model an arbitrary object-level algorithm as a parameterised black box. In contrast, some approaches integrate metareasoning components into a specific search algorithm [11, 26], enabling decision-making about which branches of a search tree to expand or execute. The integrated approach is more suitable for problems with timing constraints, where some search branches may become impossible to execute due to deadlines.

*Offline metareasoning.* Callaway et al. [10] carry out approximate metareasoning by defining three estimated Value of Computation (VoC) features, which are designed to be upper or lower bounds on the value of additional reasoning. These VoC features are combined, using weights learned by Bayesian optimisation for the specific problem distribution, to estimate the true value of continuing computation. However, their method incurs significant metareasoning overhead from calculating VoC features and does not scale past the small deterministic MDPs they evaluate their method on. They assume an idealised object-level reasoning process, rather than learning the behaviour of real object-level algorithms, as we do.

One recent work carries out offline metareasoning for probabilistic planning [8], building on a reinforcement learning-based framework that Bhatia et al. [6] designed for metalevel control of deterministic search problems. One key advantage of deep RL-based metalevel control is that the cost of querying the trained metalevel policy is small compared to other methods that require online calculations to determine the benefit of further planning [10, 19]. As these methods are data-driven, they are able to learn the optimal stopping point and hyperparameter control for a given object-level problem distribution and arbitrary object-level algorithm.

*Online metareasoning.* To the best of our knowledge, the only practical online metareasoning algorithm for probabilistic planning is presented by Lin et al. [19]. Their method adds a metalevel control layer to a specific object-level algorithm, Bounded Real-Time Dynamic Programming (BRTDP) [21]. It does not require training on a problem distribution, but its performance is limited by several simplifying assumptions. The most significant of these is the *meta-myopic* assumption, which means that the metalevel agent only considers the immediate benefit of a single additional planning step. Though their formulation supports reasoning about the cost of planning in the agent's current state, the meta-myopic assumption means that the agent is unable to reason about future states potentially having lower costs of planning or more information available. They also do not consider hyperparameter control of the object-level algorithm, so their metalevel agent cannot control *how* the object-level algorithm reasons.

Ho et al. [16] analyse online metareasoning by defining an information processing Bellman objective that jointly optimises both reward and planning cost. They can therefore infer how much planning effort should be spent in each state of the MDP, in a similar manner to our work. As their objective is to study human cognition

rather than a practical metareasoning algorithm, their gradient-based algorithm scales very poorly with the state space size and is impractical for real-world problems.

In common with most prior work on metareasoning for planning [6, 8, 15, 19], we assume the object-level algorithm has access to an accurate model of the environment. Metareasoning under model uncertainty remains largely an open problem: while Godara et al. [14] have recently formulated this setting, the added complexity restricts their approach to small multi-armed bandit problems rather than the sequential decision-making problems we consider.

## 3 PRELIMINARIES

*Object-level SSP MDP.*. The object-level problem we consider is a stochastic shortest path (SSP) Markov Decision Process (MDP) [24], and we also formulate the metalevel control problem as an SSP MDP. An SSP MDP is defined as a 6-tuple $\mathcal{M} = \langle S, init, A, T, C, G\rangle$ where $S$ is a finite set of states; $init : S \to [0, 1]$ is an initial state distribution; $A$ is a finite set of actions; $T : S \times A \times S \to [0, 1]$ is a probabilistic transition function; $C : S \times A \to \mathbb{R}_{\geq 0}$ is a cost function; and $G \subset S$ is a set of absorbing, zero-cost goal states. When referring to the structure of a specific SSP MDP instance $\mathcal{M}$, we add a subscript for clarity, e.g. $T_{\mathcal{M}}$. The two widely studied problems of infinite- and finite-horizon reward maximisation can be compiled into an equivalent SSP MDP [20].

The object-level algorithm aims to produce a policy $\pi$ that minimises the expected cumulative cost of reaching the goal from the initial state. We assume that the policy is stationary and deterministic (i.e., $\pi : S \to A$), which is known to be sufficient for SSP MDPs [20]. A policy is *proper* in state $s$ if, when starting from $s$, it reaches a goal state $s_g \in G$ with probability 1. For an SSP MDP there must exist a policy that is proper in all initial states. All improper policies must have infinite expected cumulative cost from states they are improper in. Under these assumptions, a minimum cost proper policy is known to exist [20].

In the same manner as in [19], the SSP MDP is assumed to include an action NOP which represents the agent planning for a fixed duration. The cost of the NOP action is the cost of planning in that state, and the probabilistic outcomes of the NOP action define possible state transitions that may occur.

## 4 PROBLEM FORMULATION

The object-level algorithm, which is a probabilistic planner, operates on SSP MDP instances $\mathcal{M}$ drawn from a problem distribution $p(\mathcal{M})$ which is a probability distribution over possible MDPs the agent may encounter. This distribution has domain $D_{\mathcal{M}}$, and MDP instances from this domain may have different state spaces, action spaces, transition functions, and cost functions. However, they will likely share some common structure, such as the factorisation of the state space [28]. The internal state of the planner is represented by its configuration $\chi \in X$, where $X$ is the reachable configuration space for the object-level algorithm across all MDPs in $D_{\mathcal{M}}$. The configuration is the full internal state of the planner, including all search trees, value function tables or approximators, and any other internal state variables. A configuration induces a policy $\pi_\chi$ which is the best policy found by the planner so far given the current configuration. The objective of the planner is to find a policy $\pi_\chi$ that minimises the expected cumulative cost of reaching a goal state from the initial state in the object-level MDP $\mathcal{M}$.

The planner is parameterised by hyperparameters $\boldsymbol{\Delta} = \{\Delta_i\}_{i=1}^{N_\Delta}$, where each hyperparameter $\Delta_i$ takes values from either a finite set $\{\delta_1, \delta_2, \ldots, \delta_{k_i}\}$ or a continuous range $[\delta_{\min,i}, \delta_{\max,i}]$. The planner's configuration evolution over time is dependent on the problem instance $\mathcal{M}$, the current configuration $\chi$, and the hyperparameter values or sequence of values it has been run with. To model this evolution, we assume the planner runs in fixed-duration increments of $\tau$, each corresponding to one timestep in the object-level MDP. During $\tau$, the planner may perform many internal iterations or trials. When planning is carried out using hyperparameter values $\boldsymbol{\delta} = \{\delta_i\}_{i=1}^{N_\Delta}$, the configuration transition function is defined as $T_{\mathcal{M}}^{\chi,\delta}(\chi' \mid \chi, \boldsymbol{\delta})$. After an action is executed by the agent, including when the planner runs and the NOP action is therefore executed, the planner configuration is updated to account for the new state $s'$ and the corresponding configuration transition function is defined as $T_{\mathcal{M}}^{\chi,s}(\chi' \mid \chi, s')$. These transition functions may be stochastic if the planner has a stochastic component, such as a sampling component.

We are now ready to define the metalevel MDP, which represents the operation of the object-level planner on a single problem instance $\mathcal{M}$.

Definition 1 (**Metalevel MDP**). *For an SSP MDP instance $\mathcal{M} = \langle S, init, A, T, C, G\rangle$, the metalevel MDP is an SSP MDP $\mathcal{M}^M = \langle S^M, init^M, A^M, T^M, C^M, G^M\rangle$ where:*

- $S^M = S \times X$, *combining object-level states with the planner configuration,*
- $init^M(s, \chi) = init \cdot \mathbb{1}(\chi = \chi_0)$, *where $\mathbb{1}(\cdot)$ is an indicator function and $\chi_0$ is the initial configuration,*
- $A^M = (\{PLAN\} \times \Delta_1 \times \ldots \times \Delta_{N_\Delta}) \cup \{ACT\}$, *i.e. the agent may run the planner for one metalevel timestep $\tau$ using the specified hyperparameter values, or execute a single action (specified by the current policy) in the object-level MDP,*
- $T^M : S^M \times A^M \times S^M \to [0, 1]$ *is defined as:*

$$T^M((s, \chi), a, (s', \chi')) = \begin{cases} T_{\mathcal{M}}^{\chi,\delta}(\tilde{\chi} \mid \chi, \boldsymbol{\delta}) \\ \quad \cdot\ T(s, NOP, s') \cdot\ T_{\mathcal{M}}^{\tilde{\chi},s}(\chi' \mid \tilde{\chi}, s') & \text{if } a = (PLAN, \boldsymbol{\delta}) \\ T(s, \pi_\chi(s), s') \\ \quad \cdot\ T_{\mathcal{M}}^{\chi,s}(\chi' \mid \chi, s') & \text{if } a = ACT \end{cases} \tag{1}$$

*The product terms indicate the transition order: for PLAN, the planner runs for one metalevel timestep $\tau$ with hyperparameters $\boldsymbol{\delta}$ resulting in an intermediate configuration $\tilde{\chi}$, the NOP outcome at the object level is observed, and the planner configuration is updated given the outcome state $s'$. For ACT, the object-level algorithm's policy $\pi_\chi$ selects the object-level action, and the planner configuration is updated given $s'$,*

- $C^M : S^M \times A^M \to \mathbb{R}_{\geq 0}$ *is defined by the object-level cost function, using the object-level action taken:*

$$C^M((s, \chi), a) = \begin{cases} C(s, \pi_\chi(s)) & \text{if } a = ACT \\ C(s, NOP) & \text{otherwise,} \end{cases} \tag{2}$$

- $G^M = G \times X$, *representing goal states in the object-level MDP with any algorithm configuration.*

Given an object-level SSP MDP $\mathcal{M}$, our objective is to derive a metalevel policy $\pi^M : S^M \to A^M$ that is optimal for the metalevel MDP $\mathcal{M}^M$. A cost-optimal solution to the metalevel MDP implies a policy that optimally switches between running the object-level planner and acting on the planner's current policy, given the cost of planning in different states in the MDP and using the optimal hyperparameters at each timestep.

As previously discussed, it is infeasible to solve this metalevel MDP whenever the agent is provided with a new problem instance [8, 19]. Even the simplest object-level algorithms have large configuration spaces, and as the size of the metalevel MDP state space is larger than the object-level MDP state space, metareasoning overhead is prohibitive. Online metareasoning overhead could be avoided by pre-solving the metalevel MDP for every problem instance in $p(\mathcal{M})$'s support, but this is even more infeasible. To build a metalevel agent that can learn to solve metalevel MDPs for problem instances sampled from $p(\mathcal{M})$, we define a new metalevel MDP that is an abstraction of the metalevel MDP for a single problem instance, and contains features that are practical for learning.

## 5 METHOD

In this section, we present our approach to learning a metalevel policy for the metalevel MDP that is generalisable to new problem instances, in a similar way to Bhatia et al. [6] and Budd et al. [8].

### 5.1 Abstracting the Metalevel MDP

Firstly, we reduce the state space of the metalevel MDP by abstracting the configuration space $X$ to a much smaller space of algorithm features $\Omega$. The mapping function $\Phi_X : X \to \Omega$ maps configurations to features. $\Phi_X$ and $\Omega$ are designed specifically for the object-level algorithm, and are chosen to be informative summary statistics of the configuration space. As we will be using deep function approximation to learn value functions, features will be automatically weighted and there is no practical drawback to defining a large number of features. Features are often readily identifiable from an algorithm's description, such as estimated value functions, search tree size/depth, the number of iterations or samples simulated, or the size of the latest update to an estimator.

Secondly, we abstract the metalevel MDP so that it represents the operation of the planner on problems from the problem distribution $p(\mathcal{M})$, not a single problem instance. This consists of augmenting the state with a context vector $\psi \in \Psi$, where $\Phi_D : D_{\mathcal{M}} \to \Psi$. The context vector $\psi = \Phi_D(\mathcal{M})$ is a compact representation of the current problem instance $\mathcal{M}$, and allows for generalisation between problem instances with similar attributes. $\Phi_D$ and $\Psi$ are designed to provide informative summary statistics describing the MDP problem instance, and are problem-dependent. Producing these features should not require extensive analysis of $\mathcal{M}$, in order to avoid metareasoning overhead. As the object-level MDP state representation is also a component of the metalevel MDP state, using a consistent state factor representation across MDPs in the problem distribution will likely also improve generalisation. Note that, as well as encoding a metalevel control problem, the abstract metalevel MDP encodes a meta-RL problem [4] as we represent operating under a distribution of MDPs as a single MDP. Also, as with all MDP state abstraction methods, the transition function may become non-Markovian due to the chosen abstraction of the configuration space [2].

Definition 2 (**Abstract Metalevel MDP**). *For a problem distribution $p(\mathcal{M})$ over problem instances $\mathcal{M}$, the abstract metalevel MDP is an SSP MDP $\hat{\mathcal{M}}^M = \langle \hat{S}^M, \hat{init}^M, \hat{A}^M, \hat{T}^M, \hat{C}^M, \hat{G}^M \rangle$ where:*

- $\hat{S}^M = S^M \times \Omega \times \Psi$, *i.e. states are* $(s, \omega, \psi)$,
- $\hat{init}^M : \hat{S}^M \to [0, 1]$ *is defined as:*

$$\hat{init}^M(s, \omega, \psi) = \int_{\{\mathcal{M} | \Phi_D(\mathcal{M}) = \psi\}} init_{\mathcal{M}}(s)\, p(\mathcal{M})\, \mathrm{d}\mathcal{M} \times \mathbb{1}(\omega = \Phi_X(\chi_0)),$$

  *where $init_{\mathcal{M}}$ is the initial state distribution of problem instance $\mathcal{M}$, and $\Phi_X(\chi_0)$ is the initial algorithm features,*
- $\hat{A}^M = A^M$, *i.e. the action space is unchanged,*
- $\hat{T}^M : \hat{S}^M \times \hat{A}^M \times \hat{S}^M \to [0, 1]$ *is composed of four components $T_\psi$, $T^\delta_\omega$, $T^s_\omega$, and $\pi_\omega$, which are abstracted versions of those in Equation 1:*

$$\hat{T}^M\big((s, \omega, \psi), a, (s', \omega', \psi)\big) = \begin{cases} T^\delta_\omega(\tilde{\omega} \mid \omega, \boldsymbol{\delta}, \psi) \\ \quad \cdot\ T_\psi(s, \textsc{nop}, s') \cdot\ T^s_\omega(\omega' \mid \tilde{\omega}, s', \psi) & \text{if } a = (\textsc{plan}, \boldsymbol{\delta}) \\ T_\psi(s, \pi_\omega(s), s') \\ \quad \cdot\ T^s_\omega(\omega' \mid \omega, s', \psi) & \text{if } a = \textsc{act} \end{cases}$$

  *where $T_\psi$ represents object-level state transitions and $T^\delta_\omega / T^s_\omega$ represent algorithm feature transitions caused by planning or updating the object-level algorithm with the new object-level state, as in Definition 1.*
  *By abstracting the object-level MDP instance $\mathcal{M}$ to a context vector $\psi$, we no longer have a specific transition function $T_{\mathcal{M}}$. Given the set of MDPs where $\Phi_D(\mathcal{M}) = \psi$, we can construct $T_\psi$ by marginalisation over the distribution $p(\mathcal{M})$: $T_\psi = \int_{\{\mathcal{M} | \Phi_D(\mathcal{M}) = \psi\}} T_{\mathcal{M}}\, p(\mathcal{M})\, d\mathcal{M}$. Similarly, $T^\delta_\omega$ and $T^s_\omega$ are constructed by marginalisation over the object-level algorithm configuration space and the problem distribution. Finally, $\pi_\omega$ is the policy induced by the features $\omega$, so is an abstraction of $\pi_\chi$ calculated by marginalising over the configuration space $X$. Note that the context vector $\psi$ is fixed within an episode, as a single object-level problem instance is being solved,*
- $\hat{C}^M$ *and* $\hat{G}^M$ *are defined as in Definition 1, but with $\pi_\omega$ replacing $\pi_\chi$ and $\Omega$ replacing $X$.*

### 5.2 Performance Bounds for Abstract Metalevel Policies

We now analyse the approximation error introduced by our abstract metalevel MDP formulation. We derive a bound on the suboptimality of policies learned in the abstract metalevel MDP when executed in a ground (non-abstracted) metalevel MDP. The ground metalevel MDP extends Definition 1 from a single problem instance $\mathcal{M}$ to operating over the full problem distribution $p(\mathcal{M})$, similarly to the abstract metalevel MDP in Definition 2. However, unlike the abstract metalevel MDP, the ground metalevel MDP does not apply abstraction to the configuration space $X$ or the problem instance space $D_{\mathcal{M}}$. Our analysis is an extension of an existing result on

state abstraction in MDPs [1], to the metalevel control setting with SSP MDPs.

Definition 3 (**Ground Metalevel MDP**). *For a problem distribution $p(\mathcal{M})$, the ground metalevel MDP is defined as $\mathcal{M}^G = \langle S^G, init^G, A^G, T^G, C^G, G^G \rangle$ where:*

- $S^G = S \times X \times D_{\mathcal{M}}$,
- $init^G((s, \chi, \mathcal{M})) = p(\mathcal{M}) \cdot init_{\mathcal{M}}(s) \cdot \mathbb{1}(\chi = \chi_0)$,
- $A^G = A^M$,
- $T^G((s, \chi, \mathcal{M}), a, (s', \chi', \mathcal{M}')) = T^M_{\mathcal{M}}((s, \chi), a, (s', \chi')) \cdot \mathbb{1}(\mathcal{M}' = \mathcal{M})$, *i.e. the transition function is the transition function of the metalevel MDP for problem instance $\mathcal{M}$,*
- $C^G((s, \chi, \mathcal{M}), a) = C^M((s, \chi), a)$, *i.e. the cost function is the cost function of the metalevel MDP for problem instance $\mathcal{M}$, and $G^G = G \times X \times D_{\mathcal{M}}$.*

The abstract metalevel MDP (Definition 2) is then the state abstraction of $\mathcal{M}_G$ under $\Phi((s, \chi, \mathcal{M})) = (s, \Phi_X(\chi), \Phi_D(\mathcal{M}))$. Following the standard state abstraction formulation [18], Definition 2 obtains $T_\psi$ and $T_\omega$ by summing $T_G$ over ground states $(s, \chi, \mathcal{M})$ that map to each abstract state $(s, \omega, \psi)$. The two MDPs are equivalent when $\Phi_X$ and $\Phi_D$ are identity functions.

We now state the approximation conditions required for our performance bound. Our analysis relies on two standard requirements for SSP MDPs and an approximation condition on the abstraction functions.

**Requirement 1 (Bounded Costs).** The object-level SSP MDP cost functions in $D_{\mathcal{M}}$ are bounded: there exists $C_{\max} < \infty$ such that $C_{\mathcal{M}}(s, a) \leq C_{\max}$ for all $\mathcal{M} \in D_{\mathcal{M}}$, $s \in S$, and $a \in A$. In this case, the cost functions of the metalevel MDPs are also bounded by $C_{\max}$, because they are defined by the object-level cost function (see Definition 2).

**Requirement 2 (Finite Effective Horizon).** There exists a finite effective horizon $H$, which is an upper bound on the expected number of timesteps to reach a goal state in the ground metalevel MDP. We define $H = \sup_{\mathcal{M} \in D_{\mathcal{M}}, \pi \in \Pi} \mathbb{E}_\pi[\eta \mid (s_0, \chi_0, \mathcal{M}) \sim init^G]$, where $\eta$ is the number of metalevel timesteps to reach a goal state and $\Pi$ is the set of all proper policies in the ground metalevel MDP. This requirement is satisfied in practice when the planning budget per episode is bounded and any configuration $\chi$ reachable after the maximum planning budget induces a proper policy $\pi_\chi$ with bounded expected time to reach a goal state. Any metalevel policy that executes PLAN with positive probability will eventually exhaust the budget and must then ACT with such a configuration, guaranteeing goal reachability. This is the case in our experiments (Section 6).

Definition 4 ($\epsilon$-Approximate Q-Abstraction). *Following the $Q^*$-irrelevance notion of Abel et al. [1], we say that an abstraction function $\Phi$ is an $\epsilon$-approximate Q-abstraction if ground states mapped to the same abstract state have optimal Q-values differing by at most $\epsilon$. Formally, for all $(s_1, \chi_1, \mathcal{M}_1), (s_2, \chi_2, \mathcal{M}_2) \in S^G$ where $\Phi((s_1, \chi_1, \mathcal{M}_1)) = \Phi((s_2, \chi_2, \mathcal{M}_2))$:*

$$|Q^{*G}((s_1, \chi_1, \mathcal{M}_1), a) - Q^{*G}((s_2, \chi_2, \mathcal{M}_2), a)| \leq \epsilon \tag{3}$$

*for all $a \in A^G$, where $Q^{*G}$ is the optimal Q-function in $\mathcal{M}^G$.*

Proposition 1 (Abstraction Performance Bound). *Suppose $\Phi$ is an $\epsilon$-approximate Q-abstraction (Definition 4). Let $\pi^{*G}$ be the optimal policy in $\mathcal{M}^G$, $\pi^{*M}$ be the optimal policy in $\hat{\mathcal{M}}^M$, and define the induced policy $\pi^{*MG}$ in the ground metalevel MDP by $\pi^{*MG}((s, \chi, \mathcal{M})) = \pi^{*M}((s, \Phi_X(\chi), \Phi_D(\mathcal{M})))$. Then:*

$$V^{*G}((s, \chi, \mathcal{M})) - V^{*MG}((s, \chi, \mathcal{M})) \leq 2\epsilon C_{\max} H^2 \tag{4}$$

*for all $(s, \chi, \mathcal{M})$ reachable from $init_G$ under proper policies.*

Proof. Since $\hat{\mathcal{M}}^M$ is the state abstraction of $\mathcal{M}^G$ under $\Phi$ (as shown above), and $\Phi$ satisfies the $\epsilon$-approximate Q-abstraction condition, we apply Theorem 5.1 of Abel et al. [1], which bounds the value difference for $\epsilon$-approximate Q-abstractions in discounted MDPs. Their bound for discount factor $\gamma$ is $\frac{2\epsilon C_{\max}}{(1-\gamma)^2}$. Using the standard SSP-to-discounted correspondence [17], where the effective horizon $H$ replaces $\frac{1}{1-\gamma}$, we obtain the stated bound of $2\epsilon C_{\max} H^2$. □

Metalevel policies derived from the abstract MDP, such as $\pi^{*MG}$ in Proposition 1, are also proper in the ground MDP when they satisfy the same conditions as Requirement 2. Under the bounded planning budget, executing PLAN with positive probability ensures eventual execution with a proper configuration, guaranteeing goal properness and therefore inclusion in $\Pi$.

Estimating $\epsilon$ for a given abstraction is challenging, as abstraction functions are typically hand-designed features extracted from algorithm state and problem descriptions. However, Proposition 1 provides guidance for abstraction design: effective algorithm features and problem context features should preserve distinctions between states with different optimal action values. Our empirical results (Section 6) demonstrate that our chosen features, described in detail in the supplementary material[1], achieve strong performance across the tested domains.

## 5.3 Learning to Act in the Abstract Metalevel MDP

Algorithm 1 outlines our method for training a deep RL-based policy for the abstract metalevel MDP. We present the approach with a DQN-like structure [23] for clarity, but the metalevel MDP construction and interaction is independent of the learning algorithm used, and any deep RL algorithm could be used to learn a metalevel policy. The RL policy is trained in-the-loop by sampling one object-level problem instance from $p(\mathcal{M})$ at the start of each episode. Additional implementation details are given in the supplementary material, but we highlight one key design choice here:

Budd et al. [8] provide their RL-based offline metalevel controller with an observation of the offline planning cost per timestep, so that it can learn how to act under different planning costs. Along similar lines, we provide the agent with the Q-value estimate $Q_\chi(s, \text{NOP})$. This is the estimated value of executing the NOP action in the current state $s$ *according to the object-level algorithm's current value estimate*, so is not an assumption of knowledge of the true optimal value function. Compared to only providing the immediate cost of planning, the Q-value captures both immediate planning cost and the estimated value of possible successor states after taking the NOP action. As it is dependent on the object-level algorithm configuration $\chi$, it is incorporated into the features $\omega$ in Line 20.

[1] Supplementary material is available at https://doi.org/10.5281/zenodo.18614866 [9].

**Algorithm 1** General deep RL on the abstract metalevel MDP

**Input**: Problem distribution $p(\mathcal{M})$, object-level algorithm, number of training episodes $M_e$, object-level planning time per metalevel timestep $\tau$

**Output**: Trained policy $\pi_\theta^M$.

Initialise $\pi_\theta^M$ randomly
**for** episode = $1, \ldots, M_e$ **do**
  Sample $\mathcal{M}$ from $D$ according to $p(\mathcal{M})$
  $\psi \leftarrow \Phi_D(\mathcal{M})$ {calculate context for problem instance $\mathcal{M}$}
  $\chi \leftarrow \chi_0$; {instantiate planner}
  $\omega \leftarrow \Phi_X(\chi)$ {calculate initial features}
  $s \sim init_{\mathcal{M}}$ {sample initial object-level state}
  $\hat{s}^M \leftarrow (s, \omega, \psi)$ {construct metalevel state}
  **repeat** {run episode}
    $\hat{a}^M \leftarrow$ select action using $\pi_\theta^M(\hat{s}^M)$ {or e.g. $\epsilon$-greedy}
    **if** $\hat{a}^M = (\text{PLAN}, \boldsymbol{\delta})$ **then**
      $\chi \sim T_{\mathcal{M}}^{\chi,\boldsymbol{\delta}}(\cdot \mid \chi, \boldsymbol{\delta})$ {run planner for time $\tau$ using hyperparameters $\boldsymbol{\delta}$}
      $a \leftarrow$ NOP
    **else** {$\hat{a}^M = \text{ACT}$}
      $a \leftarrow \pi_\chi(s)$ {select action using object-level policy induced by $\chi$}
    **end if**
    $\hat{c}^M \leftarrow C_{\mathcal{M}}(s, a)$ {incur object-level cost from planning or acting in instance $\mathcal{M}$}
    $s \sim T_{\mathcal{M}}(s, a, \cdot)$ {object-level state outcome}
    $\chi \sim T_{\mathcal{M}}^{\chi,s}(\cdot \mid \chi, s)$ {update planner with new current object-level state}
    $\omega \leftarrow \Phi_X(\chi)$ {update features}
    $\hat{s}^{M\prime} \leftarrow (s, \omega, \psi)$
    Store $(\hat{s}^M, \hat{a}^M, \hat{c}^M, \hat{s}^{M\prime})$ {e.g. replay buffer}
    $\hat{s}^M \leftarrow \hat{s}^{M\prime}$
  **until** $s \in G_{\mathcal{M}}$
  Update parameters of $\pi_\theta^M$ using stored episode data
**end for**

A similar Q-value is also used by the myopic metalevel controller in Lin et al. [19].

# 6 EXPERIMENTS

## 6.1 Baseline Algorithms

We compare to three baseline metalevel control methods. The most basic is a fixed decision-time planner (*DTP*), parameterised by the ratio of planning to acting time. The optimal value of this parameter is problem-dependent, and so we evaluate a range of values for each problem instance to find the best-performing ratio. The second baseline, *Bounds*, is the correlated metareasoner proposed by Lin et al. [19]. It uses a meta-myopic approximation based on how the object-level algorithm's value bounds changed over the previous computation step. The third baseline is the offline learned metareasoner from Budd et al. [8] (*OffLearn*), which learns from reasoning data and uses hyperparameter control to optimise the object-level algorithm's performance as our method does, but only supports planning at the initial state. This method relies on access to a default policy to complete its policy outputs, effectively granting it privileged information not available to our approach. We call our method the *OnLearn* metareasoner.

## 6.2 Object-level Algorithm: Weighted Bounded RTDP (WBRTDP).

*Bounds* is built on the Bounded Real-Time Dynamic Programming (BRTDP) algorithm [21]. For fair comparison between metareasoning agents, we therefore use a BRTDP-based object-level algorithm for both learning methods. As the learning-based methods support hyperparameter control, we specifically use Weighted BRTDP (WBRTDP) [8] as the object-level algorithm. The weight parameter allows the learning-based algorithms to trade off the quality of the solution and the speed of computing that solution. Each metalevel MDP timestep corresponds to a fixed number of state Bellman backup operations in the object-level algorithm. This is a more consistent time unit than walltime, which is dependent on hardware capability and load, or BRTDP trials, which vary in length. Additional details, including discussion of the algorithm features abstraction function, are given in the supplementary material.

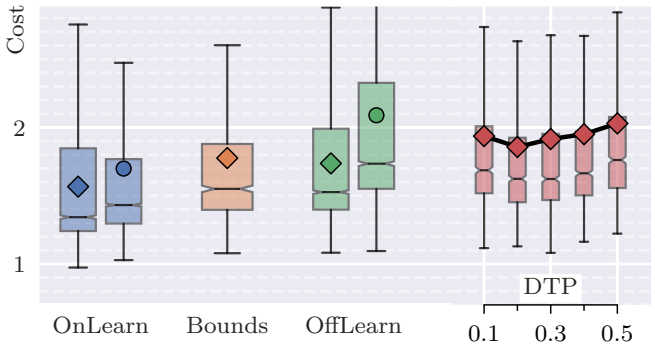
## 6.3 Deep RL Algorithm

We use DQN [23] as the deep RL algorithm for the metalevel controller. We are able to use DQN as the metalevel controller because the hyperparameter values are discrete, leading to a purely discrete action space. A different deep RL algorithm would be required if the hyperparameters were continuous, which would lead to a mixed discrete-continuous action space [12]. Other learning algorithms are evaluated in the supplementary material. For RL-based metalevel controllers, we evaluate across 5 DQN agents trained with different random seeds. Details on training and compute resources are given in the supplementary material.

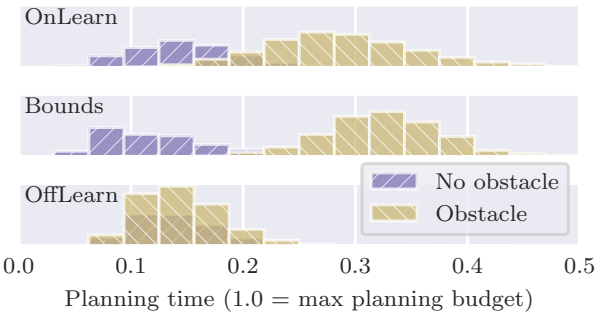
## 6.4 Experiment Domains

The **ObstacleRacetrack** domain is based on the racetrack domain used by Budd et al. [8] and originally introduced by Barto et al. [3]. It is a 2D gridworld where the agent state is the 4-tuple $x, y, v_x, v_y$, and actions are to accelerate in an 8-connected direction. Actions have a failure probability that varies by problem instance. The agent's goal is to reach one of two finish lines. However, one finish line is potentially blocked by an obstacle: whether it is blocked or not is unknown to the agent *a priori*. This is represented in the object-level MDP by a state factor with values $\{-1, 0, 1\}$, where $-1$ indicates the obstacle state is unknown. The agent observes the state of the obstacle by colliding with it if it is present, or passing through that area if it is not. The finish line behind the possible obstacle is closer, so incurs less cost to reach and requires less intensive planning to determine how to reach. This domain is designed to demonstrate the benefits of online metareasoning under epistemic uncertainty: the optimal metalevel behaviour is to generate a simple policy to reach the obstacle, and then to plan a detailed policy to reach the further away goal *only if the obstacle is present*.

The maximum size of an ObstacleRacetrack problem instance is $\sim 11\times$ larger and $\sim 2.5\times$ larger than those used to evaluate the previous state-of-the-art methods for online metareasoning [19], and offline metareasoning [8] respectively.




**(a) Normalised cost-to-goal for the method and baselines. Markers (diamond marker for full method, circle for disabled hyperparameter control) show the mean cost.**




**(b) Distribution of planning time for the method and baselines, separated by the ground-truth obstacle state.**

**Figure 2: Cost performance and planning time distributions in the ObstacleRacetrack domain.**

The **RadWorld** domain is based on the example in Figure 1. The agent exists in a gridworld with randomly generated obstacles and radiation level distributions. It aims to reach a goal on the opposite side of the map while minimising the total radiation exposure from planning and acting. Actions are to move one cell in an 8-connected manner, with costs geometrically adjusted for diagonal movement. Actions have a radiation-level dependent failure probability, ranging linearly from 0 to 0.75 for the lowest and highest radiation levels respectively. This domain is designed to demonstrate the benefits of online metareasoning where different states have substantially different costs of planning.

Problem distributions are defined implicitly by procedural generation of object-level MDPs for the agent to act in. Learning methods are trained on 10K object-level MDPs drawn from $p(\mathcal{M})$. All algorithms are evaluated on 1K object-level MDPs also drawn from $p(\mathcal{M})$, which form a held-out evaluation set. Each evaluation problem instance is seeded by its ID, ensuring all algorithms face identical stochastic outcomes (such as ground-truth obstacle presence in ObstacleRacetrack).

Domain visualisations, problem context functions and further details are given in the supplementary material.

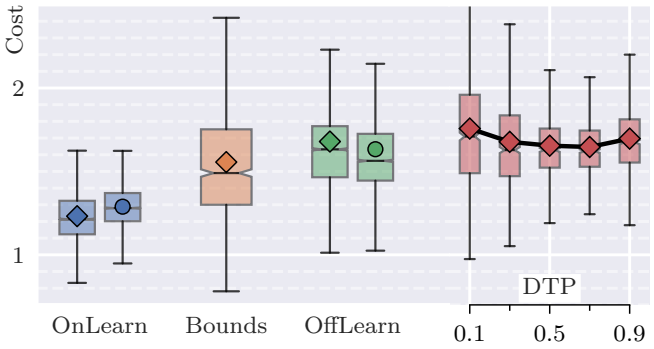
## 6.5 Quantitative and Qualitative Results

Cost results are normalised by the expected cost of the optimal policy for that problem instance. It is not possible to achieve normalised cost of 1 in expectation when reasoning has costs, so a normalised cost of 1 is a lower bound on cost (and therefore an upper bound on performance) for all metareasoning methods. As domains are stochastic, it is still sometimes possible for a few runs to achieve normalised cost $< 1$. When comparing algorithms in the text, we report how much excess cost they incur over the lower bound of 1, rather than the difference between the normalised costs of the algorithms. Statistical significance is calculated using the one-sided Mann-Whitney U test, with significance threshold $p = 0.01$. For the learning-based metalevel controllers, we also train and evaluate in a case where hyperparameter control is disabled, giving the learned controllers access to the same interface as *Bounds*.

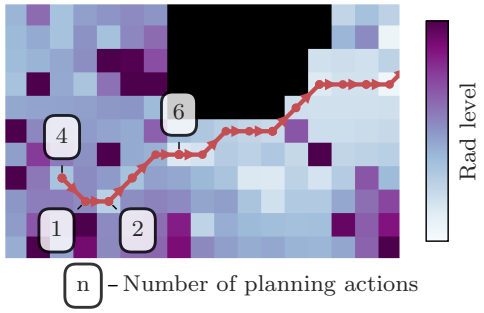
In the ObstacleRacetrack domain, *OnLearn* achieves significantly lower normalised cost than both *Bounds* and *OffLearn*, as shown in Figure 2a. Relative to *OnLearn*, *Bounds* incurs a 37% mean excess-cost increase (61% median increase), while *OffLearn* incurs a 30% mean increase (54% median increase). All parameterisations of *DTP* perform worse than the other methods: the best-performing parameterisation (0.2) incurs a 51% mean excess cost (82% median) relative to *OnLearn*. The ablated version of *OnLearn* performs 18% worse than *OnLearn*, but is still significantly better than *Bounds*. The ablated version of *OffLearn* performs significantly worse than all online methods.

Figure 2b shows the distribution of planning time used by three of the metalevel controllers, with the distributions separated by whether the obstacle ground-truth state was present or not present in that evaluation MDP. The two online metalevel controllers (*OnLearn* and *Bounds*) both use significantly more planning time when the obstacle is present, as this is the state where the agent needs to plan a more complex policy. Analysis of planning states in MDPs where the obstacle was present shows planning taking place after the obstacle is reached. Despite varying its planning time, and displaying a larger variation in planning times than *OnLearn*, there is no significant cost performance difference between *OffLearn* and *Bounds*. *OffLearn* has the benefit of hyperparameter control and being provided with a default policy, although the default policy does not check the obstacle state.

A wider cost performance gap is observed in the RadWorld domain, as shown in Figure 3a. For all methods the mean and median performance is similar, indicating that the distribution of costs is more consistent across runs than in ObstacleRacetrack. *OnLearn* achieves significantly lower normalised cost than all baselines. Compared to *OnLearn*, *Bounds* incurs 2.4× as much excess cost and *OffLearn* incurs 2.9× as much excess cost. *Bounds* performs significantly better than *OffLearn* in this domain, and *OffLearn* does not significantly outperform the best *DTP* parameterisation. *Bounds* is able to make myopic metalevel decisions about the cost of planning in its current state, but is not able to reason about the cost of planning being lower in potential future states. *OffLearn* is able to control how long to reason based on the radiation level of the




**(a) Normalised cost-to-goal for the method and baselines. Markers show the mean cost. Note that the value range of DTP planning ratio differs from Figure 2a.**




**(b) The beginning of one evaluation trajectory for the online learning method, showing the agent's path (red) and how much time it chooses to spend planning at each state.**

**Figure 3: Cost performance and an example evaluation trajectory for the RadWorld domain.**

start state, but has no ability to reason at other states. The ablated version of *OnLearn* performs very similarly to *OnLearn*, showing that most of the performance gain is due to non-myopic metalevel reasoning rather than hyperparameter control. Interestingly, in this domain, the ablated version of *OffLearn* performs slightly better than *OffLearn*.

Figure 3b shows the qualitative reasoning behaviour of *OnLearn* in a RadWorld MDP, showing only the states near the start state. It spends 4 metalevel timesteps planning at the start state, but waits until it reaches a low radiation level state to plan for longer. After this, it only performs one more planning step during the 80-length trajectory to the goal.

We carried out an ablation of *OnLearn* by replacing the NOP action Q-value estimate observation (Section 5.3) with the immediate cost of the NOP action in the current state, in a similar manner to Budd et al. [8]. We found that this led to a statistically significant 10% increase in mean excess cost in the ObstacleRacetrack domain, and no significant difference in the RadWorld domain. The lack of effect in RadWorld is likely because the agent remains in the same state while planning, whereas in ObstacleRacetrack the agent may transition state depending on its current speed. As the RL agent has an observation of the object-level algorithm's value estimate of the current state, it is straightforward to reconstruct $Q_X(s, \text{NOP})$ from this and the immediate cost observation.

## 7 CONCLUSION

We have presented a learning-based metalevel control framework for online metareasoning in probabilistic planning, and shown it to outperform several baselines in two novel environments that are well-suited to evaluating metareasoning methods.

*Limitations.* As a deep RL-based method, the trade-off for minimal metareasoning overhead is the requirement to train on a representative problem distribution, which we assume is available. The metalevel control formalisation itself, with separate object-level and metalevel agents, also introduces one notable limitation in the online setting: one can construct problems where optimal planning and acting decisions are not aligned, leading to suboptimal metalevel performance. For example, if there exists a state with a low cost of planning which does not contribute to solving the object-level problem, the metalevel agent has no way to instruct the planner to visit that state. This limitation could be addressed by tighter integration of the metalevel controller and planner, with the drawback of losing the generality of the metalevel controller.

Designing the abstraction functions $\Phi_X$ and $\Phi_D$ requires domain knowledge. However, as demonstrated in our experiments (with abstraction functions detailed in the supplementary material), effective abstractions can be constructed from basic problem characteristics and algorithm statistics, without requiring extensive theoretical analysis. Finally, although we consider a simple case of epistemic uncertainty in the environment, we assume the agent has a perfect model of the environment.

## ACKNOWLEDGMENTS

This work received EPSRC funding via the "From Sensing to Collaboration" programme grant [EP/V000748/1], and was supported by a gift from Amazon Web Services.

## REFERENCES

[1] David Abel, David Hershkowitz, and Michael Littman. 2016. Near optimal behavior via approximate state abstraction. In *International Conference on Machine Learning*. PMLR, 2915–2923.
[2] Aijun Bai, Siddharth Srivastava, and Stuart Russell. 2016. Markovian State and Action Abstractions for MDPs via Hierarchical MCTS.. In *Proceedings of the 25th International Conference on Artificial Intelligence (IJCAI)*. 3029–3039.
[3] Andrew G Barto, Steven J Bradtke, and Satinder P Singh. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72, 1-2 (1995), 81–138.
[4] Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. 2023. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028* (2023).
[5] Dimitri Bertsekas. 2022. *Lessons from AlphaZero for optimal, model predictive, and adaptive control*. Athena Scientific.
[6] Abhinav Bhatia, Justin Svegliato, Samer B Nashed, and Shlomo Zilberstein. 2022. Tuning the hyperparameters of anytime planning: A metareasoning approach with deep reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, Vol. 32. 556–564.
[7] Matthew Budd, Paul Duckworth, Nick Hawes, and Bruno Lacerda. 2023. Bayesian reinforcement learning for single-episode missions in partially unknown environments. In *Conference on Robot Learning (CoRL)*. PMLR, 1189–1198.
[8] Matthew Budd, Bruno Lacerda, and Nick Hawes. 2024. Stop! Planner Time: Metareasoning for Probabilistic Planning Using Learned Performance Profiles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 20053–20060.

[9] Matthew Budd, Bruno Lacerda, and Nick Hawes. 2026. Think Fast! Learning to Control Online Reasoning in Stochastic Environments, Supplementary Material. https://doi.org/10.5281/zenodo.18614866

[10] F Callaway, S Gul, P Krueger, TL Griffiths, and F Lieder. 2018. Learning to select computations. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI)*. 776–785.

[11] Bence Cserna, Wheeler Ruml, and Jeremy Frank. 2017. Planning time to think: Metareasoning for on-line planning with durative actions. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, Vol. 27. 56–60.

[12] Olivier Delalleau, Maxim Peter, Eloi Alonso, and Adrien Logut. 2019. Discrete and continuous action representation for practical RL in video games. *arXiv preprint arXiv:1912.11077* (2019).

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

[14] Prakhar Godara, Tilman Diego Aléman, and Angela J Yu. 2024. Metareasoning in uncertain environments: a meta-BAMDP framework. *arXiv preprint arXiv:2408.01253* (2024).

[15] Nicholas Hay, Stuart Russell, David Tolpin, and Solomon Eyal Shimony. 2012. Selecting computations: theory and applications. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence (UAI)*. 346–355.

[16] Mark K Ho, David Abel, Jonathan D Cohen, Michael L Littman, and Thomas L Griffiths. 2020. The efficiency of human cognition reflects planned information processing. In *Proceedings of the 34th AAAI conference on artificial intelligence*.

[17] Michael Kearns and Satinder Singh. 2002. Near-optimal reinforcement learning in polynomial time. *Machine learning* 49, 2 (2002), 209–232.

[18] Lihong Li, Thomas J Walsh, and Michael L Littman. 2006. Towards a unified theory of state abstraction for MDPs. *AI&M* 1, 2 (2006), 3.

[19] Christopher H Lin, Andrey Kolobov, Ece Kamar, and Eric Horvitz. 2015. Metareasoning for Planning Under Uncertainty. In *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI)*. 1601–1609.

[20] Mausam and Andrey Kolobov. 2012. *Planning with Markov decision processes: An AI perspective*. Morgan & Claypool Publishers.

[21] H Brendan McMahan, Maxim Likhachev, and Geoffrey J Gordon. 2005. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*. 569–576.

[22] Smitha Milli, Falk Lieder, and Thomas Griffiths. 2017. When does bounded-optimal metareasoning favor few cognitive systems?. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[24] Martin L Puterman. 1994. Markov Decision Processes: Discrete Stochastic Dynamic Programming.

[25] Stuart Russell and Eric Wefald. 1991. Principles of metareasoning. *Artificial Intelligence* 49, 1-3 (1991), 361–395.

[26] Eren Sezener and Peter Dayan. 2020. Static and dynamic values of computation in MCTS. In *Proceedings of the Thirty-Sixth Conference on Uncertainty in Artificial Intelligence (UAI)*. 31–40.

[27] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of Go without human knowledge. *nature* 550, 7676 (2017), 354–359.

[28] Alexander L Strehl, Carlos Diuk, and Michael L Littman. 2007. Efficient structure learning in factored-state MDPs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 7. 645–650.

[29] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems (NeurIPS)* 35 (2022), 24824–24837.

[30] Shlomo Zilberstein and Stuart Russell. 1995. Approximate reasoning using anytime algorithms. In *Imprecise and approximate computation*. Springer, 43–62.