PERFORM BETTER AND SPEND LESS: VARIABLE-COMPUTE POLICIES IN RUNTIME-AWARE RL

Anonymous authors
Paper under double-blind review

ABSTRACT

We study control problems where an agent is penalized for the time it takes to compute an action. Consequently, in scenarios where taking more time to compute an action results in a better outcome, a trade-off emerges: how should an agent balance the short-term cost incurred for calculating an action against the long-term total reward that taking that action would result in. We formalize such problems as *runtime aware MDPs (RAMDPs)* that track the compute spent by an agent for determining each action it takes, and discuss some interesting consequences of this formalization. Notably, in some cases, only *variable-compute policies*, i.e., policies that vary their computation time depending on the environment's state, are optimal in RAMDPs. We also conduct experiments on three simulated environments to illustrate how variable-compute policies can maintain performance while spending less compute when compared to *uniform-compute policies*.

1 INTRODUCTION

We study the trade-off between the computation expended by a reinforcement learning (RL) agent and its performance. In many real-world tasks, agents spend a variable amount of time to compute actions in different states, e.g., in a game of chess, players might take more time to calculate their moves for critical board positions (Kahneman, 2011). However, the default paradigm in RL is to model policies as spending a uniform amount of time to compute actions across all states (call these as *uniform-compute policies*). While many policy classes, such as tabular or multi-layer perceptron (MLP) policies, naturally fall within this paradigm, many others, such as transformers with a reasoning loop or test-time planning policies, are better modeled as *variable-compute policies*, i.e., policies that can vary the amount of compute they expend depending on the state. This work investigates variable-compute policies and how to train them in environments that wait for the agent to act.

Variable-compute policies can scale their compute based on how expensive it is to calculate an action and how impactful that action might be, thereby saving the time spent while still retaining good performance. For instance, consider a 3-state Markov decision process (MDP) where it takes one unit of compute to output the optimal action in state one, two units in state two, and three units in state three. Implementing this state to optimal action map using a uniform-compute parameterization (e.g., a fixed layer MLP) will end up using three units of compute for each of the states, thereby wasting resources, when compared to a variable-compute parameterization. Further assume that for this 3-state MDP, there exists an almost-optimal policy that is same as the previously described policy, but only takes one unit of compute in state three. In this case, an agent can save additional compute by adopting this almost-optimal policy, at the cost of losing on some reward. Additionally, for MDPs with unbounded state spaces, uniform-compute policies with an a-priori bound on their compute per state are not feasible, and the agent must scale its compute depending on the size of the state (not state-space) it encounters (see §A for more details and examples).

The MDP formulation does not capture the effect of varying compute to trade off minimizing immediate cost with maximizing long-term value, as it does not track an agent's runtime (Ghugare et al., 2026). Prior work addresses this by considering augmented models of MDPs that reason about computation , e.g., thought MDPs (Hanna & Corrado, 2025), MDPs with imaginary states (Chung et al., 2023), metareasoning MDPs (Lin et al., 2015), iterative-bounded MDPs (Topin et al., 2021), and denoising MDPs (Ren et al., 2025). Now, depending on the computational model, the optimal policy might take different number of compute steps before outputting the optimal environment action. While, one could try to find an optimal computational model that minimizes, say, the number

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

of compute actions, runtime, or feature size, focusing on specific computational models can distract us from deriving general-purpose algorithms that can effectively trade-off performance and computation. We therefore aim to formulate a decision process that is agnostic to the specific model of computation while taking computation into consideration when evaluating an agent's performance.[1]

Related works. There are various ways to implement variable-compute policies, e.g., the adaptive-computation time (ACT) algorithm that enables neural networks to model computation through learning when to halt (Graves, 2016; Banino et al., 2021; Zhu et al., 2026). Other methods halt computation once some criterion, such as a fixed-point equation, is satisfied (Fein-Ashley & Rashidinejad, 2026; Huang et al., 2026). While, chain-of-thought (CoT) methods spend extra compute by extending the sequence being processed with appending tokens (Wei et al., 2022; Wang et al., 2025). We formalize these policies by equipping a fixed-compute function with a computational space.

Many formulations relevant to RAMDPs exist, such as semi-MDPs (Sutton et al., 1999b) that can be instantiated either as options (Precup et al., 1998) or as action-chunking (Zhou et al., 2024; Li et al., 2025). Analyses on semi-MDPs have focused on the benefits of temporal abstractions and open-loop actions (Biedenkapp et al., 2021; Hu & Leung, 2023; Robert et al., 2023; Li et al., 2024). RAMDPs complement these studies since computing the best option or action-chunk can require variable computation in constrained settings. Another relevant formulation is modeling delays in MDPs (Bouteiller et al., 2020; Riemer et al., 2025), say, by assuming that action and observation delays are random variables (r.v.) independent of the policy , or that the policy has the same runtime distribution across states. In contrast, RAMDPs treat runtime as a r.v. that changes over the episode in a highly policy-dependent manner. In LLM applications, inefficient CoT reasoning can make inference costly, which has been addressed by focusing on reasoning with budget constraints (Yu et al., 2025; Liu et al., 2026a; Qi et al., 2025) and predicting inference cost (Manvi et al., 2026). These settings operate in a single-turn reward process where the MDP has deterministic transitions (appending the generated token to the current string), whereas in this paper, we focus on long-term decision-making problems where the generated action leads to a stochastic next state.

Contributions. We formalize the notion of *runtime aware MDPs (RAMDPs)*, which augment an MDP to track the compute taken by the agent's policy to output actions (§2). Additionally, an RAMDP only transitions once the agent has taken an action. This is in contrast to real-time RL (Travnik et al., 2018; Ramstedt & Pal, 2019; Muppidi et al., 2026), where the environment can transition while the agent computes an action. Next, we define variable-compute policies (§2) and reason why using such policies can be beneficial (§3). In particular, we give examples, which show that under computationally-constrained setting, only variable-compute policies are optimal. Since modeling compute time extends the state and action spaces to be countable, learning a policy or a value function in RAMDPs can be challenging. We mitigate this issue slightly by showing that the class of walltime-independent policies are representationally as strong as the class of walltime-dependent policies, and that we can represent walltime-dependent value functions by multiplying a walltime-independent value function with a discounting term (§4). Finally, we demonstrate empirically that variable-compute policies vary their computation based on the state, and can maintain performance to that of the best-performing fixed-compute policy while spending less compute (§5).

2 Problem setting (MDPs, RAMDPs, and variable-time policies)

We begin by recalling the standard MDP definition and then extend it to RAMDPs by including the policy runtime. After that, we give a general formulation for studying variable-compute policies followed by some examples of such policies. (We use standard notation[2] and defer all proofs to §B.)

Markov decision process (MDP) is a 5-tuple $\mathsf{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ represent the environment's state and action sets respectively, $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ its transition function, $r : \mathcal{S} \times \mathcal{A} \to [0, 1]$ its reward function, and $\gamma \in [0, 1)$ its discount factor. An RL agent maintains a policy

[1] This is motivated from complexity theory, where the hardness of a problem is agnostic to the choice of the model of computation (e.g., Turing machine) up to polynomial factors (Arora & Barak, 2009). Also, invariance theorem, from algorithmic information theory, states that the Kolmogorov complexity of a string only differs by an additive constant between Turing-complete computational models (Grünwald & Vitanyi, 2008).

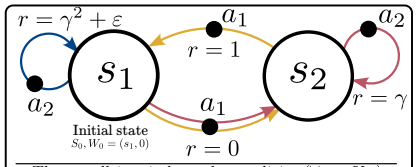
[2] **Notation.** $\mathbb{N} = \{0, 1, \ldots\}$ is the set of natural numbers and $\mathbb{R}$ is the set of reals. Let $\mathbb{N}^\infty := \mathbb{N} \cup \{\infty\}$ and $\mathbb{N}_+^\infty := \mathbb{N}^\infty \setminus \{0\}$. For $n \in \mathbb{N}_+^\infty$, $[n] := \{1, \ldots, n\}$. For a set $\mathcal{E}$, $\Delta(\mathcal{E})$ denotes the set of all probability distributions over $\mathcal{E}$. For $x \in \mathcal{E}$, $\delta_x \in \Delta(\mathcal{E})$ denotes the Dirac-delta distribution centered on x. For a Boolean expression expr, $\mathbb{I}(\mathsf{expr})$ denotes the indicator function evaluated on expr.

$\pi : \mathcal{S} \to \Delta(\mathcal{A})$ that interacts with the MDP to produce a *trajectory* $(\bar{S}_0, A_0, R_1, \bar{S}_1, A_1, \ldots)$, where $\bar{S}_0$ is sampled from some initial-state distribution $\mu_0 \in \Delta(\mathcal{S})$, and for $h \in \mathbb{N}$, $a_h \sim \pi(\cdot|\bar{S}_h)$, $\bar{S}_{h+1} \sim P(\cdot|\bar{S}_h, A_h)$, and $R_{h+1} := r(S_h, A_h)$. We define the trajectory return as $G := \sum_{h=0}^{\infty} \gamma^h R_{h+1}$. Let $\mathbb{P}^\pi$ denote the distribution over trajectories induced by μ_0, π, and P (and let $\mathbb{E}^\pi$ denote the corresponding expectation). We also define the state-value function $V^\pi(s) := \mathbb{E}^\pi[G \,|\, S_0 = s]$ for all $s \in \mathcal{S}$, and the action-value function $Q^\pi(s, a) := \mathbb{E}^\pi[G \,|\, S_0 = s, A_0 = a]$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. The goal in RL is to find a policy π that maximizes the expected return $\mathbb{E}^\pi[G]$. Note that in the MDP formulation, the time taken by an agent to compute an action is not accounted for: an agent that takes a long time to compute its actions is as preferable as an agent that computes its actions faster.

Runtime-aware MDP (RAMDP), denoted by $\overline{\mathsf{M}}$, extends an MDP M by tracking the total compute time spent by the policy from the beginning of the episode to the current timestep. The RAMDP $\overline{\mathsf{M}}$ retains M's transition dynamics, but further discounts the rewards based on the running walltime at that step. In particular, the RAMDP starts in state $S_0 \sim \mu_0$ with the initial walltime W_0 (typically set to zero). An agent interacts with the RAMDP via a *runtime-aware policy* $\bar{\pi} : \mathcal{S} \times \mathbb{N}^\infty \to \Delta(\mathcal{A} \times \mathbb{N}_+^\infty)$, that, given the state and the walltime, outputs a distribution over the environment action and the runtime of the policy. This interaction results in the trajectory $(S_0, W_0, A_0, C_0, R_1, S_1, W_1, \ldots)$, where the action and one-step policy runtime $(A_h, C_h) \sim \bar{\pi}(\cdot, \cdot|S_i, W_i)$, for $h \in \mathbb{N}$. The next state $S_{h+1} \sim P(\cdot|S_h, A_h)$, cumulative walltime $W_{h+1} := W_h + C_h - 1$, and the reward $R_{h+1} := \gamma^{W_h + C_h - 1} r(S_h, A_h)$, for $h \in \mathbb{N}_+$. The trajectory return $G := \sum_{h=0}^{\infty} \gamma^h R_{h+1} = \sum_{h=0}^{\infty} \gamma^{C_0 + \cdots + C_h} r(S_h, A_h)$ and the goal remains to find a policy $\bar{\pi}$ that maximizes the expected return $\mathbb{E}^\pi[G]$. We now give an example to illustrate an RAMDP.

Example 1. Optimal policies with different compute costs in a two state, two action RAMDP.

Consider the MDP (discount factor $\gamma = 0.6$, $\epsilon = 0.01$) and the walltime independent policies, given on the right. Policy π_{11} takes the yellow path and has a runtime of one in both the states; behavior of π_{22} is similar. Whereas, policy π_{21} has a runtime of two in s_1 and one in s_2, and follows the blue path to generate the trajectory: $(s_1, 0), (a_1, 2), 0, (s_2, 2), (a_2, 1), \gamma, (s_2, 3), (a_2, 1), \gamma, \ldots$, and gets a return of $\mathbb{E}^{\pi_{21}}[G] = \gamma^3/(1-\gamma) = 0.54$. The other two policies get returns of $\mathbb{E}^{\pi_{11}}[G] = \gamma/(1-\gamma) \approx 0.94$ and $\mathbb{E}^{\pi_{22}}[G] = \gamma(\gamma^2 + \epsilon)/(1-\gamma^2) \approx 0.35$.
Also note that each policy is optimal in its respective runtime policy class; e.g., π_{21} achieves the highest return amongst all policies that have a runtime of two steps in s_1 and one step in s_2.



Three walltime independent policies ($\forall w \in \mathbb{N}_+$):

$\pi_{11}(s_1, w) = (a_1, 1)$	$\pi_{11}(s_2, w) = (a_1, 1)$
$\pi_{22}(s_1, w) = (a_2, 2)$	$\pi_{22}(s_2, w) = (a_1, 2)$
$\pi_{21}(s_1, w) = (a_1, 2)$	$\pi_{21}(s_2, w) = (a_2, 1)$

This example shows that policies with different runtimes, e.g., exhibit different optimal behaviors. If we increase the runtime of all actions by a constant, the change in value is bounded and depends only on γ, and the upper bound on this change becomes tighter as we decrease the amount—indicating a preference over shorter runtimes—as shown by the following proposition.

Proposition 2 (Runtime-shifted value gap)**.** Let $\bar{\mathcal{S}} = \mathcal{S} \times \mathbb{N}^\infty$ and $\bar{\mathcal{A}} = \mathcal{A} \times \mathbb{N}_+^\infty$. For some $\pi : \bar{\mathcal{S}} \to \bar{\mathcal{A}}$ and $K > 0$, let $\pi^K : \bar{\mathcal{S}} \to \bar{\mathcal{A}} \times \mathbb{R}$ be the policy that outputs $\pi^K(s) = (a^\pi(s), c^\pi(s) + K)$ for all $s \in \bar{\mathcal{S}}$, where $(a^\pi(s), c^\pi(s)) = \pi(s)$. The value gap is then $V^\pi(s) - V^{\pi^K}(s) \in [0, \gamma/(1-\gamma)]$.

Variable-compute policies are modeled by us using a *controller* and a *computational space*; this is inspired from the structure of Turing machines. For parameters $\theta \in \mathbb{R}^d$, a variable-compute policy is defined as $\pi_\theta := (\mathsf{C}_\theta, f_\theta)$. Its computational space is the 3-tuple $\mathsf{C}_\theta = (\mathcal{X}, \mathcal{U}, P_\theta)$, where $\mathcal{X}$ is the computational state set, $\mathcal{U}$ is the computational action set disjoint from the environment's action set (i.e., $\mathcal{A} \cap \mathcal{U} = \emptyset$), and $P_\theta : \mathcal{S} \times \mathcal{X} \times \mathcal{U} \to \Delta(\mathcal{X})$ is the computational transition function (conditioned on the environment state). The policy's controller is a function $f_\theta : \mathcal{S} \times \mathcal{X} \to \Delta(\mathcal{A} \cup \mathcal{U})$ that we assume can be simulated in uniform-TC^0 (Vollmer, 1999). This class of circuits has a fixed depth and the number of gates grows polynomially with respect to the input size—the latter is necessary as the computational state set can be countable. Furthermore, many modern NN architectures can be simulated within uniform-TC^0, e.g. MLPs and CNNs (Ke et al., 2025), Mamba (Chen et al., 2025), and transformers (Chiang, 2025). With an appropriate computational model, these policies can solve problems beyond TC^0 (Merrill & Sabharwal, 2024; Liu et al., 2026b; Xu & Sato, 2026).

We now explain the mechanics of variable-compute policies. Given an initial computational state $x_1 \in \mathcal{X}$ and an environment state $s \in \mathcal{S}$, the controller interacts with the computational space to

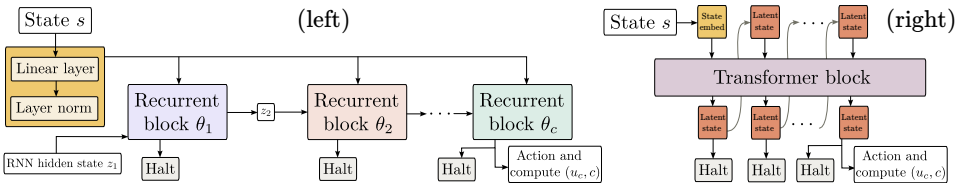


Figure 1: Two variable-compute policy architectures used in this work: **(left)** IRU with untied recurrent blocks (Ghugare et al., 2026) and **(right)** the implicit CoT architecture (Hao et al., 2025). Note that if we fix the number of recurrent blocks in IRU apriori (as done by Ghugare et al., 2026) or the number of thought tokens generated during CoT, the effective policy will be uniform-compute. In contrast, if we let the architecture decide when to "halt", i.e., output an environment action, we will obtain variable-compute policies. In our experiments, we use both these choices (see §5).

produce the computational sequence $(x_1, u_1, \ldots, x_c, u_c)$, where $u_t \sim f_\theta(\cdot \mid x_t, s)$ for $t \in [c]$ and $x_{t+1} \sim P_\theta(\cdot \mid x_t, a_t, s)$ for $t \in [c-1]$. This sequence terminates at step $c := \inf\{t \in \mathbb{N} : u_t \in \mathcal{A}\}$, when the controller outputs an environment action. Upon termination, policy π_θ, given a state s, returns the action $u_c \in \mathcal{A}$.[3] If π_θ were to interact with an RAMDP, then at timestep i, its input would be (S_i, W_i) and its output would be $(A_i, C_i) = (u_c, c)$. If there exists a $c' \in \mathbb{N}$, such that for all states $s \in \mathcal{S}$, $c = c'$ (a.s.), then we call π_θ a *uniform-compute policy*.

We illustrate this notation by some examples (also see Example 3). **(1) Uniform-compute policies**, such as MLPs or CNNs, can be realized by having a singleton computational state set and a controller that always output an environment action; i.e., let $\mathsf{C}_\theta = \big(\{x\}, \{u\}, (s, x, u) \mapsto \delta_x\big)$ for some (x, u) computational state-action pair and $s \in \mathcal{S}$, and $f_\theta : \mathcal{S} \times \{x\} \to \Delta(\mathcal{A})$. Such a policy only produces computational sequences of the form $\{(x, a)\}_{a \in \mathcal{A}}$ and consequently $c = 1$, for all $s \in \mathcal{S}$. Also, f_θ would be exactly equal to the MLP or CNN describing the policy. **(2) Variable-compute policies**, such as transformers with explicit chain-of-thought (CoT, Wei et al., 2022), can be realized by defining the transformer vocabulary to consist of action and thought tokens, which correspond to environment and computational actions respectively. Here, $\mathcal{X}$ will be the set of all possible strings (i.e., token sequences), $\mathcal{U}$ the set of thought tokens, P_θ the token-to-string append operation, and f_θ the transformer block that outputs tokens. If we replace thought tokens with the transformer's last hidden state, we can realize transformers with implicit CoT (Hao et al., 2025). In Figure 1 we show examples of two variable-compute policy architectures used in this paper. **(3) Test-time-planning policies**, such as Monte-Carlo tree search (MCTS, e.g., see Munos, 2014, §2.1; Sutton & Barto, 2018, §8.11) can be realized by having the computational space contain the simulated rollouts and their results, e.g., by setting $\mathcal{X}$ to be the set of rollouts and their value estimates, $\mathcal{U}$ to contain the single action *simulating a rollout*, and P_θ to be the combination of the exploration strategy and the approximate environment model used to simulate a rollout. Note that for MLPs and transformers, P_θ is deterministic and not learned, whereas for test-time planning P_θ is stochastic and learned.

3 When is variable computation helpful?

Variable-compute policies have many benefits over uniform-compute policies: (i) one can increase computational power of a policy by either making the controller class or the computational space more expressive; (ii) the computational space need not be a neural network or differentiable; (iii) many algorithms can be unified using the framework of a controller and computational space (Topin et al., 2021; Chung et al., 2023); and (iv) we can trade off long-term reward for short-term costs in states that require more computation to output an action. This last benefit requires us to incorporate compute cost in the MDP. Thought MDP (Hanna & Corrado, 2025) does this by penalizing computational actions with discounting,[4] and using a deterministic P_θ that is independent of the policy parameters θ. More importantly, Hanna & Corrado (2025, Prop. 4) showed that in a finite-MDP, there exists an optimal uniform-compute policy, thereby arguing that in such MDPs, there is no need for variable-compute policies. While their argument is correct, using variable-compute policies even in finite MDPs can lead to savings. As another example, modern LLM architectures can leverage

[3] It is possible for $c = \infty$, e.g., for a non-halting computational process. A way to avoid this is to set a cutoff time after which the controller outputs, say, an environment action sampled from some default distribution.

[4] One can instead do constrained optimization where we aim to maximize the expected return subject to expected compute cost (Liu et al., 2026a), which can be translated to an additive penalty. If the additive penalty is a strongly convex function, then we can analyze it through regularized MDPs (Geist et al., 2019).

iterative computation (Wei et al., 2022) for increasingly difficult tasks; if they were to instead, use a uniform-compute network, then they would either not be able to solve certain problems, or spend an inordinate amount of compute on simpler problems. To make things more concrete, we now give two examples that show how having a computational space can benefit the agent if the uniform-compute controller f_θ is not powerful enough to represent the optimal policy in a finite-MDP.

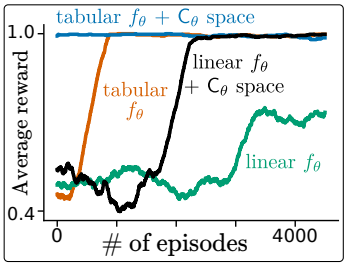
Example 3. Utility of computational space for a restricted controller in the MDP M_{XOR}.

Let M_{XOR} be a one-length episodic MDP based on the XOR problem with $\mathcal{S} = \{0,1\}^2$; $\mathcal{A} = \{0,1\}$; $\mu_0 = P(\cdot|s,a) = \sum_{s\in\mathcal{S}} \delta_s/4$; and for $s = (s_1, s_2) \in \mathcal{S}$, $r(s,a) = \mathbb{I}(a = \mathsf{XOR}(s_1, s_2))$. Let $\Pi := \{s \mapsto \arg\max_i (As)_i - 1 : A \in \mathbb{R}^{2\times 2}\}$, be the class of "linear" policies. Note that $\sup_{\pi\in\Pi} \mathbb{E}^\pi[G] = 0.75$, whereas an optimal policy π^* can get $\mathbb{E}^{\pi^*}[G] = 1$. We'll now construct a computational space and a controller with the same form as Π (an $\arg\max$ operator applied to a linear transform of the inputs) that will allow us to achieve optimality in this MDP.

Let $\theta = [A\ B] \in \mathbb{R}^{4\times 6}$, with $A \in \mathbb{R}^{4\times 2}$ and $B \in \mathbb{R}^{4\times 4}$. Define the computational space $\mathsf{C}_\theta = \{\mathcal{X}, \mathcal{U}, P_\theta\}$, with $\mathcal{X} = \{0,1\}^4$; $\mathcal{U} = \{2,3\}$; for $(s,x,u) \in \mathcal{S}\times\mathcal{U}\times\mathcal{X}$, $P_\theta : (s,x,u) \mapsto \delta_{x'}$ is the deterministic transition function, where $x' = (0, \mathbb{I}(u=2), \mathbb{I}(u=3), r(s,u-2))$; and the deterministic controller $f_\theta(s,x) := \arg\max_i (As + Bx)_i - 1$, for all $(s,x) \in \mathcal{S}\times\mathcal{X}$. Note that P_θ depends on the reward function of the underlying MDP M_{XOR}. Additionally, assume that the computational sequence always starts in $x_1 = (1,0,0,0)$. We claim that there exist parameters θ, such that the variable-compute policy $\pi_\theta = (\mathsf{C}_\theta, f_\theta)$ can solve M_{XOR} optimally.

Indeed, define $\theta^* \in \mathbb{R}^{4\times 6}$ as follows: $\theta^*_{1,4} = \theta^*_{2,5} = 0.5$, $\theta^*_{1,6} = \theta^*_{2,6} = \theta^*_{3,3} = \theta^*_{4,4} = 1$, and $\theta^*_{i,j} = 0$ for all other (i,j) pairs of indices. Let us trace how π_θ will operate in state $s = (0,1)$. By definition, $x_1 = (1,0,0,0)$. Then $u_1 = f_\theta(s,x_1) = 2 \notin \mathcal{A}$, giving $x_2 = (0,1,0,0)$. Next action $u_2 = f_\theta(s,x_2) = 3 \notin \mathcal{A}$, leading to $x_3 = (0,0,1,1)$. Finally, $u_3 = f_\theta(s,x_3) = 1 \in \mathcal{A}$ and the computational sequence terminates. Therefore, $\pi_\theta((0,1)) = u_3 = 1$, which gets a reward of $r((1,0),1) = 1$ and takes three compute steps. Similarly, π_θ can compute the optimal action in three steps in state $(1,0)$, and in two steps in each of states $(0,0)$ and $(1,1)$.

We also ran a Q-learning agent, in the this MDP, with four variations: policy with a tabular or a linear (followed by an $\arg\max$ operator) controller, and each with or without the access to the computational space C_θ. The figure on the right shows the average reward obtained by these four agents on the MDP M_{XOR} over 5000 episodes. We can observe that tabular Q-learning achieves optimality both with and without access to a computational space. However, linear Q-learning seems to perform better after having access to the computational space.



Note that the computational space in the above example can work for any non-linearly separable prediction problem if we assume access to a *verifier*, i.e., a function that can gauge the goodness of an agent's policy while having minimal cost. In our example, P_θ was the verifier, since it depended on the environment's reward function $r(\cdot,\cdot)$, however, the agent did not incur any cost while "accessing" this reward function in the computational space. In practice, some verifiers only provide approximate feedback. We can also view the verifier as a "tool", which is the view taken in the agentic framework where LLM agents have access to external APIs (Schick et al., 2023; Qin et al., 2024) in addition to their own internal reasoning mechanisms. For completeness, we present Example 4 that gives a computational space construction, without requiring a verifier; it essentially transforms the original state space into a linearly separable feature space.

Example 4 (Chan et al., 2026). A verifier-free computational space for solving M_{XOR}.

Recall the MDP M_{XOR} from Example 3 above. Let the parameters $\theta = [A\ B] \in \mathbb{R}^{3\times 3}$, with $A \in \mathbb{R}^{3\times 2}$ and $B \in \mathbb{R}^{3\times 1}$. Define $\mathsf{C}_\theta = \{\mathcal{X}, \mathcal{U}, P_\theta\}$, with $\mathcal{X} = \{0,1\}$; $\mathcal{U} = \{2\}$; $P_\theta(s,x,u) := \delta_1$, for all $(s,x,u) \in \mathcal{S}\times\mathcal{U}\times\mathcal{X}$; and $f_\theta(s,x) := \arg\max_i (As + Bx)_i - 1$, for all $(s,x) \in \mathcal{S}\times\mathcal{X}$. Assume that $x_1 := 0$. Then for θ^* given on top-right, $\pi_\theta = (\mathsf{C}_\theta, f_\theta)$ solves M_{XOR} optimally. In particular, π_θ outputs the optimal action for $s = (0,0)$ in one step, and the optimal actions for $s \in \{(0,1),(1,0),(1,1)\}$ in two steps.

$$\theta^* = \begin{bmatrix} -10 & -10 & 30 \\ -13.5 & -13.5 & 35 \\ 1 & 1 & 0 \end{bmatrix}$$

4 LEARNING POLICIES IN RUNTIME-AWARE MDPS

In this section, we discuss how to develop algorithms for learning variable-compute policies that can trade off computation and performance. Since RAMDP is also an MDP, an appealing way is to use Bellman-based algorithms to find an optimal policy. However, under the computationally-constrained setting (e.g., Example 3), such algorithms can suffer from delusional bias (Lu et al., 2018). Although, under some assumption on the policy class, we may still use a tractable Bellman-based algorithm to obtain an optimal policy (see §D for details). More generally, we illustrate in §E that one can use modifications of policy-class value iteration (Lu et al., 2018) to obtain an optimal policy. Unfortunately, these algorithms, in general, require extra assumptions and a (potentially uncomputable) witness oracle. We instead aim to employ various policy gradient methods (PG, Williams, 1992; Sutton et al., 1999a; Schulman et al., 2017; Haarnoja et al., 2018; Shao et al., 2024). The PG theorem shows that $\nabla_\pi \mathbb{E}_s\left[V^\pi(s)\right] = \mathbb{E}_\tau\big[\sum_{h=0}^{\infty} \nabla_\pi \log \pi(A_h|S_h) Q^\pi(S_h, A_h)\big]$. A naïve application of PG would be to find policies that are functions of both environment state and walltime. This unnecessarily increases the complexity of finding the best policy as the walltime is unbounded. It turns out that time-independent representations can be as strong.

Time-independent representations. Observe that the runtime only influences the return but not the environment state dynamics, we show in Theorem 5 that policies that consider walltime do not provide any benefits. In practice, it is more difficult to take walltime as input to the model as it can grow indefinitely. Let $\Pi^{(\text{wall})} := \{\pi : \mathcal{S} \times \mathbb{N} \to \Delta(\mathcal{A} \times \mathbb{N})\}$ represent the set of all walltime dependent Markov policies. Call a policy $\pi \in \Pi^{(\text{wall})}$ walltime independent, if there exists $\pi^{(\text{ind})} : \mathcal{S} \to \Delta(\mathcal{A} \times \mathbb{N})$ such that $\pi(\cdot, \cdot \mid s, w) = \pi^{(\text{ind})}(\cdot, \cdot \mid s)$, for all $(s, w) \in \mathcal{S} \times \mathbb{N}$; i.e., a walltime independent policy π effectively ignores the walltime when outputting an action.

Theorem 5 (Walltime-independent policies suffice)**.** For any walltime dependent policy $\pi \in \Pi^{(\text{wall})}$, there exists a walltime independent policy π' such that $V^\pi \leq V^{\pi'}$.

Likewise, both the Q-function and the value function are functions of at least one of walltime w and compute time c. As both $w \in \mathbb{N}$ and $c \in \mathbb{N}^+$ are countably infinite, we cannot hope to represent either functions using a table even in finite environment state-action spaces. Fortunately by Lemma 6 we can exploit the structure of RAMDP and write $Q^\pi((s, w), (a, c)) = \gamma^{w+c-1} Q^\pi((s, 0), (a, 1))$ and $V^\pi((s, w)) = \gamma^w V^\pi((s, 0))$. In other words, we can simply keep track of $V^\pi((s, 0))$ and $Q^\pi((s, 0), (a, 1))$ and scale them with appropriate discounting.

Lemma 6 (Runtime factorization)**.** Fix any walltime-independent policy π, then for every $(s, a, c) \in \mathcal{S} \times \mathcal{A} \times \mathbb{N}^+$ and every $w \in \mathbb{N}$, the following relations hold true: (i) $V^\pi((s, w)) = \gamma^w V^\pi((s, 0))$, (ii) $Q^\pi((s, w), (a, c)) = \gamma^w Q^\pi((s, 0), (a, c))$, and (iii) $Q^\pi((s, 0), (a, c)) = \gamma^{c-1} Q^\pi((s, 0), (a, 1))$.

Consequently, under the tabular setting one can solve RAMDPs similarly to standard MDPs when we only consider $c = 1$. On the other hand, under the function-approximation setting, while we can no longer assume that all (a, c) pairs are realizable, the runtime factorization also yields a statistical benefit (see §C). Further, in practice any PG methods (including those with baselines) can simply optimize for a walltime-independent policy and a walltime-independent value function. For example, we can write the TD error as $\delta_t = \gamma^{W_t + C_t - 1} r_t(S_t, A_t) + \gamma^{W_t + C_t} V(S_{t+1}) - \gamma^{W_t} V(S_t) = \gamma^{W_t} \tilde{\delta}_t$, where we define $V(s) := V(s, 0)$ and $\tilde{\delta}_t = \gamma^{C_t - 1} r_t(S_t, A_t) + V(S_{t+1}) - V(S_t)$. We can then compute the generalized-advantage estimate (GAE, Schulman et al., 2015) through backward recursion by setting $\hat{A}_t = \gamma^{W_t} \tilde{A}_t$ where $\tilde{A}_t := \sum_{t' \geq 0} \lambda^{t'} \big(\prod_{k=t}^{t+t'-1} \gamma^{C_k}\big) \tilde{\delta}_{t+t'} = \tilde{\delta}_t + \lambda \gamma^{C_t} \tilde{A}_{t+1}$.

5 EXPERIMENTAL RESULTS

We conducted experiments with the aim to answer the following questions. **(Q1:)** Can we train agents that vary compute based on the state? If so, do these agents maintain performance in terms of success rate? **(Q2:)** Can we train agents that effectively maximize discounted returns under the RAMDP formulation? **(Q3:)** Do the trained agents spend more compute on more difficult instances?

Setup. To address these questions, we conduct experiments on three domains: lightsout (Ghugare et al., 2026), sliding puzzle, and Sokoban (Bonnet et al., 2024). The agent receives sparse rewards for both lightsout and sliding puzzle, where the $+1$ reward is obtained upon success; and the agent receives the default dense rewards for Sokoban, where the agent receives a -0.1 reward per step,

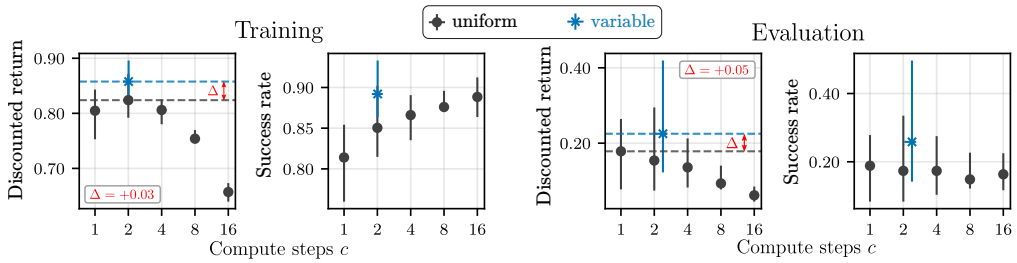


Figure 2: Comparison between uniform budgets and variable budget in `lightsout-3x3` with the IRU architecture. The markers correspond to the interquartile mean (IQM) and the bars correspond to 95% stratified bootstrap confidence interval (CI).

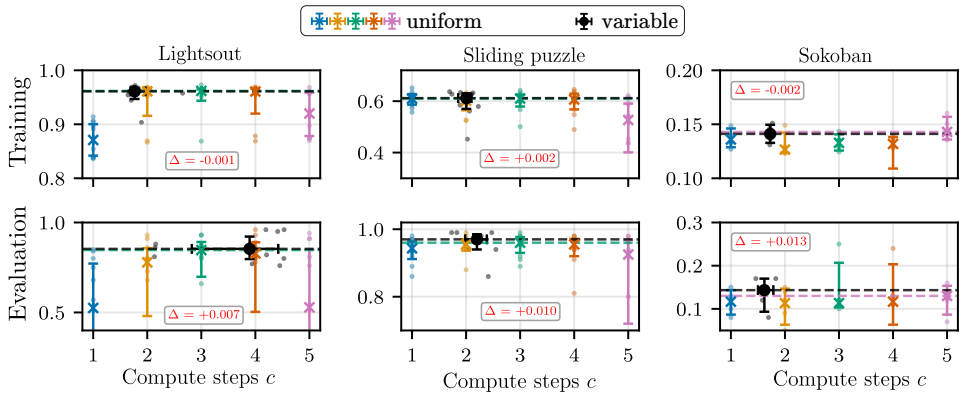


Figure 3: Success-rate comparison between uniform budgets and variable budget in 5×4 lightsout, 3×3 sliding puzzle, and Sokoban. The policies are transformers with implicit chain-of-thought reasoning. The markers correspond to the IQM and the bars correspond to 95% stratified bootstrap CI. The coloured dots correspond to each of ten runs per compute budget for lightsout and sliding puzzle and five runs per compute budget for Sokoban.

a $+1$ reward upon placing a block to the target location, and a $+10$ reward upon solving the level completely. For all experiments, we train each agent using the PPO-clip objective (Schulman et al., 2017) with GAE, and we train each value function by performing expectile regression against λ-returns. To evaluate **Q1** we compare the policies based on the success rate and compute steps; for **Q2** we compare the policies based on the discounted return; for **Q3** we provide correlation analysis between compute time and instance difficulty. We provide the environment details and hyperparameters in §F, and additional analyses in §G.

Simplified setting. We start with a simple experiment on a 3×3 lightsout environment. We model the policy with the IRU architecture (Ghugare et al., 2026) without parameter sharing. For every compute step c the model passes the previous hidden state into a separate learned IRU layer θ_c (Figure 1, left). This ensures that the model becomes more expressive as more compute steps are taken. Here we set the maximum compute steps to be $c_{\text{max}} = 16$ and set $\gamma = 0.99$ and run each variant for ten seeds. Figure 2 shows that fixed-compute policies with more compute steps obtain higher success rate during training. The variable-compute policy, on the other hand, can achieve the highest-performing fixed-compute policy with average compute $c_{\text{mean}} \approx 2$. We also see that it outperforms all fixed-compute policies in discounted return. Perhaps quite surprisingly, having more compute steps can be detrimental to generalization capabilities, as the success rate starts to decrease as fixed-compute policy uses more compute steps. This result supports both **Q1** and **Q2**, but do we get similar results in larger-scale domains?

Larger-scale domains. We conduct experiments on 5×4 lightsout environment, 8-sliding puzzle, and Sokoban. We model the policies using transformers with implicit CoT (Hao et al., 2025)—the architecture shares parameters across compute steps when the input sequence length increases. All policies first project the observation into a feature vector using an encoder, followed by Pre-LN transformer blocks (Xiong et al., 2020). Its last hidden state is then fed into a halting head and an action head, both of which are separate linear functions followed by a sigmoid and a softmax respectively (Figure 1, right). We use RMSNorm (Zhang & Sennrich, 2019) rather than LayerNorm.

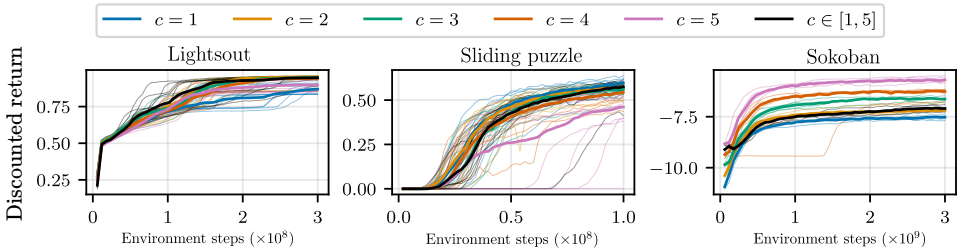


Figure 4: The learning curves of uniform budgets and variable budget in 5×4 lightsout, 3×3 sliding puzzle, and Sokoban. The policies are transformers with implicit chain-of-thought reasoning. We plot both the IOM and the curve for each seed.

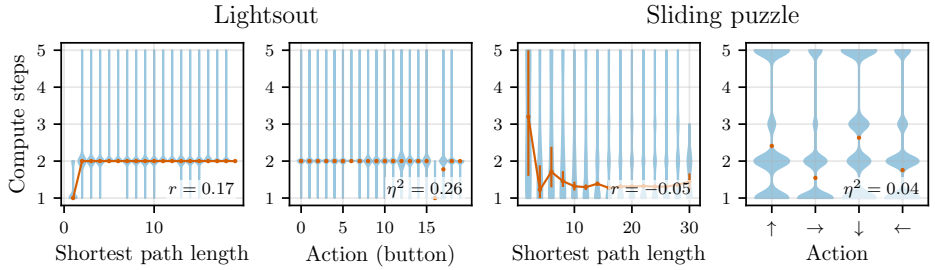


Figure 5: Correlation between compute time and instance difficulty, (action), in lightsout and sliding puzzle environments. We sample 50,000 instances to generate a violin plot and the IQM with 95% stratified bootstrap CI for each analysis.

For all environments we set $c_{\max} = 5$ and set $\gamma = 0.999$. For Sokoban we run each variant with five seeds, and for lightsout and sliding puzzle we run each variant with ten seeds.

Figure 3 shows the success rates and compute times of each agent in the three domains. For 5×4 lightsout environment, the variable-compute policy matches the performance of the best-performing uniform-compute policy while expending less computation during training. During evaluation, while the variable-compute policy maintains the performance on the heldout instances, its average compute steps generally exceed $c = 3$, except for three seeds where compute steps is slightly above 2. For sliding puzzle, the variable-compute policy also matches best-performing uniform-compute policy while using less compute steps during both training and evaluation. Finally, for Sokoban, the variable-compute policy matches the performance to that of the best-performing uniform-policy while using less compute steps in both training and evaluation. This result supports **Q1**—the variable-compute policy can do as well as the best-performing uniform-compute policy, but note that the former policy can use more compute than the latter policy.

Figure 4 shows the learning curves of each agent in the three domains. For lightsout, the variable-compute policy can generally match the best uniform-compute policy in discounted return. For sliding puzzle, the variable-compute policy performs slightly worse than the best policy's discounted return $c = 1$. The learning curves for the variable-compute policy shows that one seed stagnates quite significantly, and also appear to stagnate in three runs. For both lightsout and sliding puzzle, we observe that when $c = 5$ the learning curves seem to vary significantly. For Sokoban, the variable-compute policy also fails to achieve the best policy's discounted return. Since the there is a per-step negative reward penalty, we would expect the variable-compute policy to always spend $c_{\max}$ compute steps. Inspecting the IQM compute time in Figure 3 we can see that the variable-compute policy simply fails to exploit the fact that increasing compute steps decreases the per-step penalty. Furthermore, Figure 10 shows that initially the variable-compute agent uses large compute steps but decreases over training in Sokoban. This result does not fully support **Q2** (lightsout supports it while sliding puzzle and sokoban refute it). However, based on these results we can investigate what causes the variable-compute agent to stagnate in learning and why it fails to optimize for negative rewards (even if in this case it is okay to take more compute steps).

Does compute time correlate with instance difficulty? One might surmise that variable-compute policies always spend more compute time as the instance difficulty increases. For example, obtaining the shortest path between two nodes takes polynomial time w.r.t. the graph size. To test this we evaluate the trained variable-compute policies that achieves near-perfect success rate in 5×4

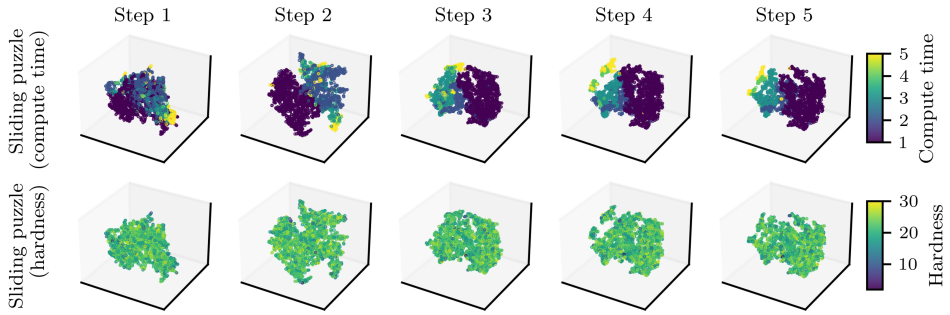


Figure 6: Visualization on the last hidden states generated each step by the transformer with iCoT in the sliding puzzle environment. The colour of each marker corresponds to the compute steps in the top row, and the hardness in the bottom row.

lightsout and 8-puzzle. For the former, we obtain the shortest path between the initial grid and the goal grid by converting this to a linear system (Anderson & Feil, 1998) and solving it using Gauss-Jordan elimination. For the latter, we obtain the shortest path between the initial configuration to the goal configuration in 8-puzzle by generate the shortest paths starting at the goal configuration using breadth-first search. For this analysis we take the best variable-compute policy per environment.

Figure 5 shows the Pearson correlation between compute steps and instance difficulty, and the eta-squared between compute steps and actions. For lightsout, the compute steps and instance difficulty exhibits a weak correlation ($r = 0.17$) and the action explains some variance of the compute time ($\eta^2 = 0.26$). The policy mostly uses $c \approx 1$ on the simplest instances (shortest path length of 1), but otherwise concentrates on $c \approx 2$. For more difficult instances the policy also spends at least 2 compute steps. For sliding puzzle, there is no correlation between the compute steps and the instance difficulty (nor does the action explain much of the compute time's variance). We can see that the policy already allocates more compute on easier instances (shortest paths of length less than 10), and that it is unable to solve more difficult instance even with longer rollout length. It is possible that the underlying transformer is too limited even with $c_{\text{max}} = 5$. To support this point, we further show in Figure 6 a visualization of last hidden states coloured by compute time and problem hardness. We can see that the projected hidden states form clusters based on compute time, but not based on problem hardness as they scatter across the projected space. In general, we can say that the answer to **Q3** is false and it is likely that extra mechanisms are required for stronger correlation.

6 Conclusions

We introduce runtime-aware MDP (RAMDP) to capture the trade-off between compute and performance (§2). We show that variable-compute policies are stronger than fixed-compute policies under constrained setting (§3). This work suggests that if we have computationally-bounded agents we could continually scale the agents or continually learn even in a stationary environment (§A). Under finite state-and-action spaces, we provide some practical policy-gradient algorithms (§§4, C.1) and show that Bellman-based algorithms could be impractical (§§D, E). It will be interesting to provide more analysis and develop algorithms in the countable-state setting, where not all problems can be solved using fixed-compute policies (Murthy et al., 2024).

Limitations and future work. Our experimental result shows that variable-compute policies can match and sometimes outperform fixed-compute policies in different control tasks. While we show that in some cases variable-compute agents can trade off compute and performance through maximizing the discounted return in RAMDPs, learning such policies becomes more challenging as we increase the number of compute steps and the architecture components. We also find a surprising result that the compute step allocated by a variable-compute policy does not necessarily correlate with the instance difficulty nor the output action, which raises the question of how the model's resources are allocated. Our experiments only focused on problems that do not require reactive behaviors (e.g., Atari, Bellemare et al., 2013; or Mujoco, Todorov et al., 2012). It could also be interesting to conduct experiments with LLM reasoning tasks, which are in some ways similar to the control tasks we considered in this paper; we know of prior work (Ayoub et al., 2026) that considered discounting as a means to finetune reasoning performance of LLMs in the single-turn (bandit) setting, but are not aware of any works in the multi-turn (RL) setting.

AI USE STATEMENT

In this work, we used generative AI tools for:

- Assist in the writing of proofs (formulate mathematical claims), specifically the wording in the proofs in §C.
- Implement methods, specifically the code for the algorithms and the boiler-plates to setup new environments.
- Some of the related works in §1.

We have not used generative AI tools for **developing theoretical models or conceptual frameworks, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, assisting with translation, supporting qualitative and thematic data analysis, and interpreting results**; and **generating synthetic data sets, and cleaning and reformatting dataset** are not applicable to this work. Additionally, we used generative AI tools for **creating or modifying figures or images and suggesting experimental parameters**. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide proofs of the theoretical results (§B) and the implementation details for the experiments to ensure reproducibility (§F). The complete source code including the experiment configuration files, implementation, and the scripts to reproduce the figures is available anonymously at https://anonymous.4open.science/r/Stoix-7FF3.

REFERENCES

Marlow Anderson and Todd Feil. Turning lights out with linear algebra. *Mathematics magazine*, 1998. URL https://doi.org/10.1080/0025570X.1998.11996658.

Sanjeev Arora and Boaz Barak. *Computational complexity: A modern approach*. Cambridge University Press, 2009. URL https://theory.cs.princeton.edu/complexity/book.pdf.

Alex Ayoub, Kavosh Asadi, Dale Schuurmans, Csaba Szepesvári, and Karim Bouyarmane. Learning to reason efficiently with discounted reinforcement learning. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/f08223bc8d177df6807811c32f5acfed-Abstract-Conference.html.

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. *ICML Workshop on Automated Machine Learning (AutoML)*, 2021. URL https://openreview.net/forum?id=1EuxRTe0WN.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artifical Intelligence Research (JAIR)*, 2013. URL https://doi.org/10.1613/jair.3912.

André Biedenkapp, Raghu Rajan, Frank Hutter, and Marius Lindauer. TempoRL: Learning when to act. *International Conference on Machine Learning (ICML)*, 2021. URL https://proceedings.mlr.press/v139/biedenkapp21a.html.

Clem Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence Midgley, Elshadai Tegegn, Tristan Kalloniatis, et al. Jumanji: a diverse suite of scalable reinforcement learning environments in jax. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/d6c38e81801a8d10afeabdb0a32fca8a-Abstract-Conference.html.

Yann Bouteiller, Simon Ramstedt, Giovanni Beltrame, Christopher Pal, and Jonathan Binas. Reinforcement learning with random delays. *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=QFYnKlBJYR.

Bryan Chan, Shivam Garg, and Dale Schuurmans. Necessity of computation in decision making, 2026. URL https://chanb.github.io/blogs/necessity_of_computation.

Yifang Chen, Xiaoyu Li, Yingyu Liang, Zhenmei Shi, and Zhao Song. The computational limits of state-space models and mamba via the lens of circuit complexity. *Conference on Parsimony and Learning (CPAL)*, 2025. URL https://proceedings.mlr.press/v280/chen25b.

David Chiang. Transformers in uniform TC^0. *Transactions on Machine Learning Research (TMLR)*, 2025. URL https://openreview.net/forum?id=ZA7D4nQuQF.

Stephen Chung, Ivan Anokhin, and David Krueger. Thinker: Learning to plan and act. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/4761fab863f0900d90cf601fce6d5155-Abstract-Conference.html.

Jacob Fein-Ashley and Paria Rashidinejad. Solve the loop: Attractor models for language and reasoning. *arXiv preprint*, 2026. URL https://arxiv.org/abs/2605.12466.

Marc Finzi, Shikai Qiu, Yiding Jiang, Pavel Izmailov, J Zico Kolter, and Andrew Gordon Wilson. From entropy to epiplexity: Rethinking information for computationally bounded intelligence. *arXiv preprint*, 2026. URL https://arxiv.org/abs/2601.03220.

Matthieu Geist, Bruno Scherrer, and Olivier Pietquin. A theory of regularized markov decision processes. In *International conference on machine learning (ICML)*, 2019. URL https://proceedings.mlr.press/v97/geist19a.html.

Raj Ghugare, Michał Bortkiewicz, Alicja Ziarko, and Benjamin Eysenbach. On computation and reinforcement learning. In *International conference on machine learning (ICML)*, 2026. URL https://openreview.net/pdf?id=HKzMnBX5B2.

E. Mark Gold. Language identification in the limit. *Information and Control*, 1967. URL https://doi.org/10.1016/S0019-9958(67)91165-5.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint*, 2016. URL https://arxiv.org/abs/1603.08983.

Peter D. Grünwald and Paul M. B. Vitanyi. Algorithmic information theory. *arXiv preprint*, 2008. URL https://arxiv.org/abs/0809.2754.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning (ICML)*, 2018. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

Josiah Hanna and Nicholas Corrado. When can model-free reinforcement learning be enough for thinking? *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://papers.nips.cc/paper_files/paper/2025/hash/2a4179ef39846557e99f6bfac580ea2e-Abstract-Conference.html.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *Conference on Language Models (COLM)*, 2025. URL https://openreview.net/pdf?id=Itxz7S4Ip3.

Xiaoyan Hu and Ho-fung Leung. Provably (more) sample-efficient offline RL with options. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/d91b532a76ea98ac1ef5226b862bfc49-Abstract-Conference.html.

Benhao Huang, Zhengyang Geng, and Zico Kolter. Equilibrium reasoners: Learning attractors enables scalable reasoning. *International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=lh95PnOlpM.

Daniel Kahneman. *Thinking, fast and slow*. Farrar, Straus, and Giroux, 2011. URL https://en.wikipedia.org/wiki/Thinking,_Fast_and_Slow.

Yekun Ke, Xiaoyu Li, Yingyu Liang, Zhenmei Shi, and Zhao Song. Circuit complexity bounds for visual autoregressive model. *arXiv preprint*, 2025. URL https://arxiv.org/abs/2501.04299.

Jon Kleinberg and Sendhil Mullainathan. Language generation in the limit. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://papers.nips.cc/paper_files/paper/2024/hash/7988e9b3876ad689e921ce05d711442f-Abstract-Conference.html.

Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in Neural Information Processing Systems (NeurIPS)*, 1999. URL https://papers.nips.cc/paper_files/paper/1999/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html.

Qiyang Li, Zhiyuan Zhou, and Sergey Levine. Reinforcement learning with action chunking. *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://papers.nips.cc/paper_files/paper/2025/hash/50348e8f9aef984abe0ea1ec2a326f78-Abstract-Conference.html.

Zhening Li, Gabriel Poesia, and Armando Solar-Lezama. When do skills help reinforcement learning? A theoretical analysis of temporal abstractions. *International Conference on Machine Learning (ICML)*, 2024. URL https://proceedings.mlr.press/v235/li24be.html.

Christopher H Lin, Andrey Kolobov, Ece Kamar, and Eric Horvitz. Metareasoning for planning under uncertainty. *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2015. URL https://www.ijcai.org/Proceedings/15/Papers/229.pdf.

Chang Liu, Yiran Zhao, Lawrence Liu, Yaoqi Ye, Csaba Szepesvári, and Lin F. Yang. LACONIC: Length-aware constrained reinforcement learning for LLM. *arXiv preprint*, 2026a. URL https://arxiv.org/abs/2602.14468.

Yuxi Liu, Konpat Preechakul, Kananart Kuwarananchaoen, and Yutong Bai. The serial scaling hypothesis. *International Conference on Learning Representations (ICLR)*, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/ae82cea8f98b0e5960b84077c975db29-Abstract-Conference.html.

Tyler Lu, Dale Schuurmans, and Craig Boutilier. Non-delusional Q-learning and value-iteration. *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. URL https://papers.nips.cc/paper_files/paper/2018/hash/5fd0245f6c9ddbdf3eff0f505975b6a7-Abstract.html.

Rohin Manvi, Joey Hong, Tim Seyde, Maxime Labonne, Mathias Lechner, and Sergey Levine. Zero-overhead introspection for adaptive test-time compute. *International Conference on Learning Representations (ICLR)*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/b22c5b6a87d259769fd186b016412f6a-Abstract-Conference.html.

Jincheng Mei, Wesley Chung, Valentin Thomas, Bo Dai, Csaba Szepesvári, and Dale Schuurmans. The role of baselines in policy gradient optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://papers.nips.cc/paper_files/paper/2022/hash/718d02a76d69686a36eccc8cde3e6a41-Abstract-Conference.html.

William Merrill and Ashish Sabharwal. A logic for expressing log-precision transformers. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/a48e5877c7bf86a513950ab23b360498-Abstract-Conference.html.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. *International Conference on Learning Representations (ICLR)*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/1f59721c106ea80f613299039112f651-Abstract-Conference.html.

Rémi Munos. From bandits to Monte-Carlo tree search: The optimistic principle applied to optimization and planning. *Foundations and Trends in Machine Learning*, 2014. URL https://doi.org/10.1561/2200000038.

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

Yashaswini Murthy, Isaac Grosof, Siva Theja Maguluri, and R. Srikant. Performance of NPG in countable state-space average-cost RL. *arXiv preprint*, 2024. URL https://arxiv.org/abs/2405.20467.

Doina Precup, Richard S. Sutton, and Satinder Singh. Theoretical results on reinforcement learning with temporally abstract options. *European Conference on Machine Learning (ECML)*, 1998. URL https://link.springer.com/chapter/10.1007/BFb0026709.

Penghui Qi, Zichen Liu, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Optimizing anytime reasoning via budget relative policy optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://papers.nips.cc/paper_files/paper/2025/hash/21efcb6a9fe5c90d660d3f560bb6094d-Abstract-Conference.html.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *International Conference on Learning Representations (ICLR)*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/28e50ee5b72e90b50e7196fde8ea260e-Abstract-Conference.html.

Simon Ramstedt and Chris Pal. Real-time reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. URL https://papers.nips.cc/paper_files/paper/2019/hash/54e36c5ff5f6a1802925ca009f3ebb68-Abstract.html.

Allen Ren, Justin Lidard, Lars Ankile, Anthony Simeonov, Pulkit Agrawal, Anirudha Majumdar, Benjamin Burchfiel, Hongkai Dai, and Max Simchowitz. Diffusion policy policy optimization. *International Conference on Learning Representations (ICLR)*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/c0749c39aaff9e9e4c91f7118bf21b1e-Abstract-Conference.html.

Matthew Riemer, Gopeshh Subbaraj, Glen Berseth, and Irina Rish. Enabling real-time reinforcement learning at scale with staggered asynchronous inference. *International Conference on Learning Representations (ICLR)*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/61715c5379f9061abc9d8efe3f1723e4-Abstract-Conference.html.

Arnaud Robert, Ciara Pike-Burke, and Aldo A. Faisal. Sample complexity of goal-conditioned hierarchical reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/c5ed2c8acda8c3716b1b6f9c6c713aaa-Abstract-Conference.html.

Stuart Russell and Eric Wefald. Principles of metareasoning. *Artificial Intelligence*, 1991. URL https://www.sciencedirect.com/science/article/pii/000437029190015C.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint*, 2015. URL https://arxiv.org/abs/1506.02438.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint*, 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint*, 2024. URL https://arxiv.org/abs/2402.03300.

Hava T. Siegelmann. *Neural networks and analog computation: Beyond the Turing limit*. Birkhäuser-Verlag, 1999. URL https://link.springer.com/book/10.1007/978-1-4612-0707-8.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (2nd ed.)*. MIT Press, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems (NeurIPS)*, 1999a. URL https://papers.nips.cc/paper_files/paper/1999/hash/464d828b85b0bed98e80ade0a5c43b0f-Abstract.html.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 1999b. URL https://www.sciencedirect.com/science/article/pii/S0004370299000521.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012. URL https://doi.org/10.1109/IROS.2012.6386109.

Edan Toledo. Stoix: Distributed single-agent reinforcement learning end-to-end in jax, 2024. URL https://github.com/EdanToledo/Stoix.

Nicholay Topin, Stephanie Milani, Fei Fang, and Manuela Veloso. Iterative bounding MDPs: Learning interpretable policies via non-interpretable methods. *AAAI Conference on Artificial Intelligence (AAAI)*, 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/17192.

Mark Towers, Ariel Kwiatkowski, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, J. K. Terry, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments. *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://papers.nips.cc/paper_files/paper/2025/hash/d7ff1795e8527f6443371c3933bdb52b-Abstract-Datasets_and_Benchmarks_Track.html.

Jaden B. Travnik, Kory W. Mathewson, Richard S. Sutton, and Patrick M. Pilarski. Reactive reinforcement learning in asynchronous environments. *Frontiers in Robotics and AI*, 2018. URL https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2018.00079/full.

Heribert Vollmer. *Introduction to circuit complexity: A uniform approach*. Springer-Verlag, 1999. URL https://link.springer.com/book/10.1007/978-3-662-03927-4.

Xiaoqiang Wang, Suyuchen Wang, Yun Zhu, and Bang Liu. System-1.5 reasoning: Traversal in language and latent spaces with dynamic shortcuts. *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://papers.nips.cc/paper_files/paper/2025/hash/5e66b9f710a8e38351da690f3806cba9-Abstract-Conference.html.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992. URL https://link.springer.com/article/10.1007/BF00992696.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. *International Conference on Machine Learning (ICML)*, 2020. URL https://proceedings.mlr.press/v119/xiong20b.html.

Kevin Xu and Issei Sato. A formal comparison between chain of thought and latent thought. *International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=d6BqyajwMj.

Zishun Yu, Tengyu Xu, Di Jin, Karthik Abinav Sankararaman, Yun He, Wenxuan Zhou, Zhouhao Zeng, Eryk Helenowski, Chen Zhu, Sinong Wang, Hao Ma, and Han Fang. Think smarter not harder: Adaptive reasoning with inference aware optimization. *International Conference on Machine Learning (ICML)*, 2025. URL https://proceedings.mlr.press/v267/yu25s.html.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. URL https://papers.nips.cc/paper_files/paper/2019/hash/1e8a19426224ca89e83cef47f1e7f53b-Abstract.html.

Helen Zhou, Audrey Huang, Kamyar Azizzadenesheli, David Childers, and Zachary Lipton. Timing as an action: Learning when to observe and act. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2024. URL https://proceedings.mlr.press/v238/zhou24c.html.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason Eshraghian. Scaling latent reasoning via looped language models. *arXiv preprint*, 2026. URL https://arxiv.org/abs/2510.25741.

A Countable state space

Computable policies. It is common to treat policies as fixed-budget state-to-action mappings in RL. In practice, these policies are implemented as programs, or more dominantly as neural networks with fixed precision, thus they have inherit limitations in computational budget with a finite-length description (Siegelmann, 1999; Merrill & Sabharwal, 2023). Consequently we can describe these policies using any Turing-complete computation models—for some theoretical results we will represent policies with Turing machines (TM)—we defer the definition of a Turing machine to Arora & Barak (2009). We now introduce (deterministic) computable and time-bounded policies—note that one can extend these definitions to stochastic policies (Finzi et al., 2026).

Definition 7 (Time-bounded TMs)**.** Let M be a TM that halts on every input $s \in \{0,1\}^*$ and $t : \mathbb{N} \to \mathbb{N}$ be a function. M is t-bounded if it decides all inputs s within $t(|s|)$ steps.

Definition 8 (Computable policies and time-bounded policies)**.** Consider a deterministic policy $\pi : \{0,1\}^* \to \{0,1\}$. We have the following:

1. π is said to be computable if there exists a TM such that $\pi(s) = 1$ if $M(s)$ halts and accepts and $\pi(s) = 0$ if $M(s)$ halts and rejects for all $s \in \{0,1\}^*$.
2. π is said to be t-bounded if there exists a t-bounded TM such that $\pi(s) = 1$ if $M(s)$ accepts and $\pi(s) = 0$ if $M(s)$ rejects for all $s \in \{0,1\}^*$.

Here, notice that our definition of a computable policy needs not halt on every possible input, whereas a t-bounded policy must halt on every input. This is to capture policy classes, such as those with variable computation time (Graves, 2016, ACT) or chain-of-thought reasoning (Wei et al., 2022, CoT), where there is no guarantee that a policy will ever halt.

When the environment state space is countable, say $s = \mathbb{N}$, Ghugare et al. (2026) showed that policies with more computational budget can solve broader range of problems:

Theorem 9 (Policy Hierarchy Theorem (Ghugare et al., 2026))**.** Let $g(n)$ and $t(n)$ be time-constructible functions such that $g(n) \in o(t(n)/\log t(n))$. Let Π_t to be the set of t-bounded deterministic policies. Then there exists an MDP with an initial-state distribution $\mu \in \Delta(\mathbb{N})$ and a function $f(n) \in O(t(n))$ such that:

1. The optimal policy π^* belongs to the policy class Π_f.
2. For any policy $\pi \in \Pi_g$, we have that $\mathbb{E}_{s\sim\mu_0} V^{\pi}(s) < \mathbb{E}_{s\sim\mu_0} V^{\pi^*}(s)$.

However, this setting still assumes that computation is a free resource—the agent can spend unbounded amount of compute to obtain the optimal action. In practice, the time (or energy) spent computing such action also increases (Russell & Wefald, 1991), therefore there is a trade-off between performance and computation cost.

In addition to Theorem 9, there are MDPs with countable state space where the corresponding optimal policy is not computable or runtime unrealizable, henceforth we call these oracle policies.

Definition 10 (Runtime unrealizability)**.** A policy π is said to be runtime unrealizable if for every function $f : \mathbb{N} \to \mathbb{N}$ such that $f(n) \in O(1)$, we have that $\pi \notin \Pi_f$.

It is important to differentiate oracle policies and optimal policies under RAMDP—an oracle policy is unobtainable while optimal policies are the best obtainable policies.

Theorem 11. There exists MDPs such that the oracle policy is (1) not computable, or is (2) runtime unrealizable.

Proof. To prove this theorem we consider countable state-space MDPs. Without loss of generality we assume that $\mathcal{S} = \mathbb{N} \cup \{s_{\mathsf{abs}}\}$. In both cases, the action space is binary, i.e., $\mathcal{A} = \{0,1\}$. Upon taking any action, the environment transitions from the current state s to the absorbing state s_{abs} with $+0$ reward. In other words, these MDPs are effective one-step problems where the policy can only obtain a positive reward at the first timestep.

For (1), we will map the halting problem to a corresponding MDP. Suppose we are given a TM M, and without loss of generality we assume that $a = 1$ corresponds to M halting on its input, and that $a = 0$ corresponds to M looping on its input. Consider a MDP with the reward function is $r(s,a) = +1$ when $a \in \mathcal{A}$ corresponds to whether running M on the input s halts, and $r(s,a) = 0$ otherwise. By definition, the oracle policy π^* must halt and correctly predicts whether M halts on input s for any $s \in \mathcal{S}$. Suppose now π^* is computable, then there exists a TM M' such that $\pi^*(s) = 1$ if $M'(s)$ halts and accepts and $\pi^* = 0$ if $M'(s)$ halts and rejects. Since the halting problem is undecidable, there is no such a TM (Arora & Barak, 2009), a contradiction.

We will now prove (2) use the time-hierarchy theorem (THT) (Arora & Barak, 2009). By THT there exists a decision problem L in $\mathsf{DTIME}(O(n))$ but not in $\mathsf{DTIME}(O(1))$. Choose such a problem L, and without loss of generality we assume that $a = 1$ means that the input string is in L and $a = 0$ otherwise. Consider now a MDP with the reward function $r(s,a) = +1$ when $a \in \mathcal{A}$ corresponds to whether $s \in \mathsf{L}$, and $r(s,a) = 0$ otherwise. By definition, the oracle policy π^* must halt and

correctly predicts whether $s \in \mathsf{L}$ for all $s \in \mathcal{S}$. Since L is a decidable problem, there exists a TM M such that it halts on all inputs and has outputs equivalent to that of the oracle policy. However, since $\mathsf{L} \in \mathsf{DTIME}(O(n))$, there is a sufficiently large input length N such that countably-many input strings $s \in \mathcal{S}$ of length at least N cannot be decided within $\mathsf{DTIME}(O(1))$ steps (otherwise contradicting with the fact that $\mathsf{L} \in \mathsf{DTIME}(O(n))$). Thus, while π^* is computable, it is runtime unrealizable. □

Naturally, when we extend the MDPs to consider runtime, i.e., RAMDPs, there is a value gap between oracle and optimal policies—for example in the case where a policy class Π contains computable policies, some actions $a \in \mathcal{A}$ is not computable at a particular state $s \in \mathcal{S}$, thus the oracle action a^{ALL} can never be picked. In the following let $\bar{\mathcal{S}} = \mathcal{S} \times \mathbb{N}^{\infty}$ and $\bar{\mathcal{A}} = \mathcal{A} \times \mathbb{N}^{\infty}_{+}$. For a class of deterministic policies Π, we define the action-restricted Bellman operator to be $(\mathcal{T}^{\Pi} f)(\bar{s}) := \max_{\bar{a} \in \bar{\mathcal{A}}^{\Pi}(\bar{s})} r(\bar{s}, \bar{a}) + \gamma \mathbb{E}_{\bar{s}'}\left[f(\bar{s}')\right]$, where $\mathcal{A}^{\Pi}(\bar{s}) = \{\bar{a} \in \bar{\mathcal{A}} | \exists \pi \in \Pi, \pi(\bar{s}) = (a, c) =: \bar{a}\}$ is the set of actions that can be generated by some policy $\pi \in \Pi$ at state $\bar{s} \in \bar{\mathcal{S}}$. Recall that c is the runtime required to compute an action using the policy π at state $\bar{s}$. In §D we show that this Bellman operator has a unique fixed point, i.e., $V^{\Pi}(\bar{s}) = (\mathcal{T}^{\Pi} V^{\Pi})(\bar{s})$.

Proposition 12 (V^{ALL} optimality)**.** Let V^{ALL} be the value of the oracle policy and V^{Π} be the value of the optimal policy. For any policy class Π, we have that $V^{\mathsf{ALL}}(\bar{s}) \geq V^{\Pi}(\bar{s})$ for any state $\bar{s} \in \bar{\mathcal{S}}$.

Proof. For any $\bar{s} \in \bar{\mathcal{S}}$, we have that

$$
\begin{aligned}
V^{\Pi}(\bar{s}) &= (\mathcal{T}^{\Pi} V^{\Pi})(\bar{s}) \\
&= \max_{a \in \mathcal{A}^{\Pi}(\bar{s})} r(\bar{s}, \bar{a}) + \gamma \mathbb{E}_{\bar{s}'}\left[V^{\Pi}(\bar{s}')\right] \\
&\leq \max_{\bar{a} \in \bar{\mathcal{A}}^{\mathsf{ALL}}(\bar{s})} r(\bar{s}, \bar{a}) + \gamma \mathbb{E}_{\bar{s}'}\left[V^{\Pi}(\bar{s}')\right] \\
&\leq \max_{\bar{a} \in \bar{\mathcal{A}}^{\mathsf{ALL}}(\bar{s})} r(\bar{s}, \bar{a}) + \gamma \mathbb{E}_{\bar{s}'}\left[\max_{\bar{a}' \in \bar{\mathcal{A}}^{\mathsf{ALL}}(\bar{s}')} r(\bar{s}', \bar{a}') + \gamma \mathbb{E}_{\bar{s}''}\left[V^{\Pi}(\bar{s}'')\right]\right] \\
&\vdots \\
&\leq V^{\mathsf{ALL}}(\bar{s}),
\end{aligned}
$$

where $\mathcal{A}^{\Pi}(\bar{s}) \subseteq \mathcal{A}^{\mathsf{ALL}}(\bar{s})$ as ALL contains the set of all possible policies. □

The interesting case is where all actions are computable. Here, if the value gap between Π^{ALL} and Π is non-zero, the oracle action must require $c > 1$ for all policy $\pi \in \Pi$.

Proposition 13. Consider any policy class Π. Let $\pi^{\mathsf{ALL}}, \bar{\pi}$ be the policies realizing the fixed points of $V^{\mathsf{ALL}}, V^{\Pi}$ respectively. For any state $\bar{s} \in \bar{\mathcal{S}}$, let $a^{\mathsf{ALL}}(\bar{s})$ be the oracle action and $a^{\Pi}(\bar{s})$ be the best-in-class action. Assume that for all states there is a unique oracle action, and assume all oracle actions are realizable by some policy in Π, i.e., $\forall \bar{s} \in \bar{\mathcal{S}}, \exists \pi \in \Pi, a^{\mathsf{ALL}}(\bar{s}) = \pi(\bar{s})$. If there is a state $\bar{s} \in \bar{\mathcal{S}}$ such that $V^{\mathsf{ALL}}(\bar{s}) - V^{\Pi}(\bar{s}) > 0$, then there is a state $\bar{s}' \in \bar{\mathcal{S}}$ such that $c^{\bar{\pi}}(\bar{s}') > c^{\pi^{\mathsf{ALL}}}(\bar{s}') = 1$.

Proof. Suppose by contradiction that there is some $\bar{s} \in \bar{\mathcal{S}}$ such that $V^{\mathsf{ALL}}(\bar{s}) - V^{\Pi}(\bar{s}) > 0$. Suppose for every state $\bar{s}' \in \bar{\mathcal{S}}$ we have that $c^{\bar{\pi}}(\bar{s}') = 1$. Then, we have that

$$
\begin{aligned}
V^{\Pi}(\bar{s}) &= (\mathcal{T}^{\Pi} V^{\Pi})(\bar{s}) \\
&= \max_{\bar{a} \in \mathcal{A}^{\Pi}(\bar{s})} r(\bar{s}, \bar{a}) + \gamma \mathbb{E}_{\bar{s}'}\left[V^{\Pi}(\bar{s}')\right] \\
&\geq r(\bar{s}, \bar{a}^{\mathsf{ALL}}(\bar{s})) + \gamma \mathbb{E}_{\bar{s}'}\left[V^{\Pi}(\bar{s}')\right] \\
&\geq r(\bar{s}, \bar{a}^{\mathsf{ALL}}(\bar{s})) + \gamma \mathbb{E}_{\bar{s}'}\left[r(\bar{s}', \bar{a}^{\mathsf{ALL}}(\bar{s}')) + \gamma \mathbb{E}_{\bar{s}''}\left[V^{\Pi}(\bar{s}'')\right]\right] \\
&\vdots \\
&= V^{\mathsf{ALL}}(\bar{s}),
\end{aligned}
$$

where the inequality comes from the fact $a^{\mathsf{ALL}}(\bar{s}) = (\bar{a}^{\mathsf{ALL}}(\bar{s}), 1) \in \mathcal{A}^{\Pi}(\bar{s})$, but this contradicts $V^{\mathsf{ALL}}(\bar{s}) - V^{\Pi}(\bar{s}) > 0$. □

B Detailed proofs

B.1 Proof of Proposition 2

Proposition 2 (Runtime-shifted value gap)**.** Let $\bar{\mathcal{S}} = \mathcal{S} \times \mathbb{N}^{\infty}$ and $\bar{\mathcal{A}} = \mathcal{A} \times \mathbb{N}_{+}^{\infty}$. For some $\pi : \bar{\mathcal{S}} \to \bar{\mathcal{A}}$ and $K > 0$, let $\pi^K : \bar{\mathcal{S}} \to \bar{\mathcal{A}} \times \mathbb{R}$ be the policy that outputs $\pi^K(s) = (a^\pi(s), c^\pi(s) + K)$ for all $s \in \bar{\mathcal{S}}$, where $(a^\pi(s), c^\pi(s)) = \pi(s)$. The value gap is then $V^\pi(s) - V^{\pi^K}(s) \in [0, \gamma/(1-\gamma)]$.

Proof. For any $\bar{s} \in \bar{\mathcal{S}}$, the value gap between π and π^K is

$$
\begin{aligned}
V^\pi(\bar{s}) - V^{\pi^K}(\bar{s}) &= \mathbb{E}^\pi\left[\sum_{h=0}^{\infty} \gamma^{\sum_{h'=0}^{h} C_{h'}} r(S_h, A_h) | \bar{S}_0 = \bar{s}\right] - \mathbb{E}^{\pi^K}\left[\sum_{h=0}^{\infty} \gamma^{\sum_{h'=0}^{h} C_{h'}} r(S_h, A_h) | \bar{S}_0 = \bar{s}\right] \\
&= \mathbb{E}^\pi\left[\sum_{h=0}^{\infty} \gamma^{\sum_{h'=0}^{h} C_{h'}} r(S_h, A_h) | \bar{S}_0 = \bar{s}\right] - \mathbb{E}^{\pi}\left[\sum_{h=0}^{\infty} \gamma^{\sum_{h'=0}^{h} (C_{h'}+K)} r(S_h, A_h) | \bar{S}_0 = \bar{s}\right] \\
&= \mathbb{E}^\pi\left[\sum_{h=0}^{\infty} (\gamma^{\sum_{h'=0}^{h} C_{h'}} - \gamma^{\sum_{h'=0}^{h} (C_{h'}+K)}) r(S_h, A_h) | \bar{S}_0 = \bar{s}\right] \\
&\le \mathbb{E}^\pi\left[\sum_{h=0}^{\infty} \gamma^{\sum_{h'=0}^{h} C_{h'}} (1 - \gamma^{(h+1)K}) | \bar{S}_0 = \bar{s}\right] \\
&\le \mathbb{E}^\pi\left[\sum_{h=0}^{\infty} \gamma^{h+1} (1 - \gamma^{(h+1)K}) | \bar{S}_0 = \bar{s}\right] \\
&\le \frac{\gamma}{1-\gamma}.
\end{aligned}
$$

□

We also note that $V^\pi(\bar{s}) - V^{\pi^K}(\bar{s}) \ge 0$ as $\gamma^{\sum_{h'=0}^{h} C_{h'}} - \gamma^{\sum_{h'=0}^{h} (C_{h'}+K)} \ge 0$.

B.2 Proof of Theorem 5

Theorem 5 (Walltime-independent policies suffice)**.** For any walltime dependent policy $\pi \in \Pi^{(\text{wall})}$, there exists a walltime independent policy π' such that $V^\pi \le V^{\pi'}$.

Proof. Let $\Pi^{(\text{wall})} := \{\pi : \mathcal{S} \times \mathbb{N} \to \Delta(\mathcal{A} \times \mathbb{N})\}$ represent the set of all walltime dependent Markov policies. Call a policy $\pi \in \Pi^{(\text{wall})}$ walltime independent, if there exists $\pi^{(\text{ind})} : \mathcal{S} \to \Delta(\mathcal{A} \times \mathbb{N})$ such that $\pi(\cdot, \cdot \mid s, w) = \pi^{(\text{ind})}(\cdot, \cdot \mid s)$, for all $(s, w) \in \mathcal{S} \times \mathbb{N}$; i.e., a walltime independent policy π effectively ignores the walltime when outputting an action.

Consider any walltime-dependent policy $\pi^{(\text{wall})} \in \Pi^{(\text{wall})}$ and let $V^{(\text{wall})}((s, w))$ and $Q^{(\text{wall})}((s, w), (a, c))$ be its corresponding value function and Q-function respectively. For any $\bar{s} = (s, w) \in \bar{\mathcal{S}}$, we can find a walltime-independent policy $\pi^{(\text{ind})}$ such that

$$
\pi^{(\text{ind})}(s) = \operatorname*{arg\,max}_{a,c \in \bar{A} \times \mathbb{N}_{+}^{\infty}} \max_{w \in \mathbb{N}^{\infty}} Q^{(\text{wall})}((s, w), (a, c)).
$$

We show that this is a maximum rather than a supremum. Suppose the first walltime and runtime are respectively $W_0 = w$ and $C_0 = c$. By definition we must have $R_t \le \gamma^{w+c-1}$. Therefore we have that $0 \le G \le \frac{\gamma^{w+c-1}}{1-\gamma}$. Taking the expectation on the random variable G conditioning on $(s, w), (a, c)$ we have that $0 \le Q^{(\text{wall})}((s, w), (a, c)) \le \frac{\gamma^{w+c-1}}{1-\gamma}$. Here we also have $\gamma^\infty = 0$.

Fix any (a, c), the inner max is attainable. Indeed, suppose $M := \sup_w Q(w) \in [0, \frac{1}{1-\gamma}]$. If $M = 0$ then we are done. If $M > 0$, we want to choose a $w_0 \in \mathbb{N}$ such that

$$\frac{\gamma^{w_0+c-1}}{1-\gamma} \leq \frac{M}{2} \iff \gamma^{w_0+c-1} \leq \frac{(1-\gamma)M}{2} =: \varepsilon.$$

For sufficiently large w_0 we must have $\varepsilon > w_0$. Let $H_1 = \{0, \dots, w_0 - 1\}$ and $H_2 = \{w_0, w_0 + 1, \dots\}$. We can then consider $\sup_w Q(w) = \max \sup_{w \in H_1} Q(w), \sup_{w \in H_2} Q(w)$. By construction $\sup_{w \in H_2} Q(w) \leq M/2 < M$, so we must have that $\sup_{w \in H_1} Q(w) = M$. The set H_1 is finite (and non-empty) so its supremum is also the maximum, which is attainable.

Call the walltime that maximizes the inner maximization when fixing the (a, c) pair $w^*(a, c)$. We can then use a similar argument for the outer argmax, notably applying the above argument on c. Since $\mathcal{A}$ is finite, we can fix each $a \in \mathcal{A}$ and find the corresponding c_0 that maximizes $\sup_{c \in \mathbb{N}_+^\infty} Q((s, w^*(a, c)), (a, c))$. This becomes finding the argmax over a finite set. □

B.3 Proof of Lemma 6

Lemma 6 (Runtime factorization)**.** Fix any walltime-independent policy π, then for every $(s, a, c) \in \mathcal{S} \times \mathcal{A} \times \mathbb{N}^+$ and every $w \in \mathbb{N}$, the following relations hold true: (i) $V^\pi((s, w)) = \gamma^w V^\pi((s, 0))$, (ii) $Q^\pi((s, w), (a, c)) = \gamma^w Q^\pi((s, 0), (a, c))$, and (iii) $Q^\pi((s, 0), (a, c)) = \gamma^{c-1} Q^\pi((s, 0), (a, 1))$.

Proof. Fix any $\pi \in Pi$ and any $z \in \mathbb{N}$. We first focus on V^π. Consider a trajectory $((S_0, W_0), (A_0, C_0), (S_1, W_1), (A_1, C_1) \dots)$ with $(S_0, W_0) = (s, w)$. Notice that by definition, the walltime satisfies the recurrence $W_{h+1} = W_h + C_h - 1$ with $W_0 = w$. We can write $W_h + C_h - 1 = w + M_h$ with $M_h := \sum_{k=0}^{h} (C_k - 1) \in \mathbb{N}^\infty$. Furthermore, since $(A_h, C_h) \sim \pi(S_h)$ and $S_{h+1} \sim P(\cdot | S_h, A_h)$, the process $(S_h, A_h, C_h)_{h \geq 0}$ given $S_0 = s$ does not depend on w. Now, consider the sequence $(s_h, a_h, c_h)_{h \geq 0}$. Using this sequence we construct two trajectories:

$$\tau := ((S_h = s_h, W_h = w + M_h), (A_h = a_h, C_h = c_h))_{h \geq 0}$$
$$\tau^0 := ((S_h^0 = s_h, W_h^0 = M_h), (A_h^0 = a_h, C_h^0 = c_h))_{h \geq 0}.$$

We claim that $R(\bar{S}_h, \bar{A}_h) = \gamma^w R(\bar{S}_h^0, \bar{A}_h^0)$ term-wise. If $W_h + C_h = \infty$, then since w is finite this holds if and only if $M_h = \infty$, so both rewards are 0 and the claim holds trivially. Otherwise, we have that $M_h < \infty$ and

$$\begin{aligned} R(\bar{S}_h, \bar{A}_h) &= \gamma^{W_h + C_h - 1} r(S_h, A_h) \\ &= \gamma^{w + M_h} r(S_h, A_h) \\ &= \gamma^w \cdot \gamma^{W_h^0 + C_h - 1} r(S_h, A_h) \\ &= \gamma^w R(\bar{S}_h^0, \bar{A}_h^0). \end{aligned}$$

By the definition of the return, we have

$$G(\tau) = \sum_{h=0}^{\infty} \gamma^h R(\bar{S}_h, \bar{A}_h) = \gamma^w \sum_{h=0}^{\infty} \gamma^h R(\bar{S}_h^0, \bar{A}_h^0) = \gamma^w G(\tau^0).$$

Since $G(\tau)$ is well-defined for any trajectory τ, we have that

$$V^\pi((s, w)) = \mathbb{E}^\pi \left[G(\tau) | \bar{S}_0 = (s, w) \right] = \gamma^w \mathbb{E}^\pi \left[G(\tau^0) | \bar{S}_0 = (s, 0) \right] = \gamma^w V^\pi((s, 0)).$$

We now focus on Q^π. Splitting the return at $h = 0$ and conditioning on the first transition gives the standard recurrence

$$Q^\pi(\bar{s}, \bar{a}) = r(\bar{s}, \bar{a}) + \gamma \mathbb{E}_{\bar{s}' \sim P(\cdot | \bar{s}, \bar{a})} \left[V^\pi(\bar{s}') \right].$$

Take $\bar{s} = (s, w)$ and $\bar{a} = (a, c)$ with $w \in \mathbb{N}$ and $c \in \mathbb{N}^+$ both finite. By definition, the next state is $\bar{s}' = (s', w + c - 1)$ with $s' \sim P(\cdot | s, a)$, and $R(s, a) = \gamma^{w+c-1} r(\bar{s}, \bar{a})$. Since $w + c - 1 \in \mathbb{N}$, the

previous result applies to the next state and gives

$$\begin{aligned}
Q^\pi((s,w),(a,c)) &= \gamma^{w+c-1} r(s,a) + \gamma \mathbb{E}_{s'}\left[V^\pi((s', w+c-1))\right] \\
&= \gamma^{w+c-1} r(s,a) + \gamma^{w+c} \mathbb{E}_{s'}\left[V^\pi((s',0))\right] \\
&= \gamma^{w}\left(\gamma^{c-1} r(s,a) + \gamma^{c} \mathbb{E}_{s'}\left[V^\pi((s',0))\right]\right) \\
&= \gamma^{w}\left(\gamma^{c-1} r(s,a) + \gamma \mathbb{E}_{s'}\left[V^\pi((s',c-1))\right]\right) \\
&= \gamma^{w} Q^\pi((s,0),(a,c)).
\end{aligned}$$

For the second statement, we have that

$$\begin{aligned}
Q^\pi((s,0),(a,c)) &= \gamma^{c-1} r(s,a) + \gamma \mathbb{E}_{s'}\left[V^\pi((s', c-1))\right] \\
&= \gamma^{c-1} r(s,a) + \gamma^{c} \mathbb{E}_{s'}\left[V^\pi((s',0))\right] \\
&= \gamma^{c-1}\left(r(s,a) + \gamma \mathbb{E}_{s'}\left[V^\pi((s',0))\right]\right) \\
&= \gamma^{c-1} Q^\pi((s,0),(a,1)).
\end{aligned}$$

□

C STATISTICAL BENEFIT OF RUNTIME FACTORIZATION

In this section we show that the runtime factorization (Lemma 6) yields a statistical benefit: every sample informs the Q-value at every runtime. Concretely, fix a pair $(s, \bar{a})$ and define $Q_c := Q^\pi((s,0),(\bar{a},c))$. Suppose we are given N independent on-policy rollouts: rollout i starts at $\bar{S}_0 = (s,0)$, takes first action $\bar{A}_0 = (a, C_i)$ with runtime $C_i \sim \nu$ drawn i.i.d. from a sampling distribution ν over $\mathbb{N}^+$ (e.g., the runtime distribution of a stochastic policy at s), follows π thereafter, and records the return $G_i := \sum_{h=0}^{\infty} \gamma^h R(\bar{S}_h, \bar{A}_h)$, so that $\mathbb{E}\left[G_i | C_i = c\right] = Q_c$. We compare the separate (per-runtime) estimator against the factorized estimator that pools all samples through $Q_c = \gamma^{c-1} Q_1$:

$$\hat{Q}_c^{\mathrm{sep}} := \frac{1}{N_c} \sum_{i: C_i = c} G_i, \qquad \hat{Q}_c^{\mathrm{fac}} := \gamma^{c-1} \cdot \frac{1}{N} \sum_{i=1}^{N} \gamma^{-(C_i-1)} G_i,$$

where $N_c := |\{i : C_i = c\}|$, so that $\sum_c N_c = N$.

Theorem 14. Fix $\delta \in (0,1)$, the following statements hold:

1. (Factorized.) With probability at least $1-\delta$, simultaneously for every $c \in \mathbb{N}^+$,
$$\left|\hat{Q}_c^{\mathrm{fac}} - Q_c\right| \le \frac{\gamma^{c-1}}{1-\gamma} \sqrt{\frac{\ln(2/\delta)}{2N}}.$$
2. (Separate.) Let $\mathcal{C} \subseteq \mathbb{N}^+$ be any finite set of runtimes with $|\mathcal{C}| = C$. With probability at least $1-\delta$, for every $c \in \mathcal{C}$ with $N_c \ge 1$,
$$\left|\hat{Q}_c^{\mathrm{sep}} - Q_c\right| \le \frac{\gamma^{c-1}}{1-\gamma} \sqrt{\frac{\ln(2C/\delta)}{2N_c}}.$$

Proof. We first record a pathwise bound on the return. From the proof of the runtime factorization lemma, the discount exponent of the h-th reward along rollout i is $Z_h + C_h - 1 = (C_i - 1) + \sum_{k=1}^{h} (C_k - 1) \ge C_i - 1$, since every runtime satisfies $C_k \ge 1$. As $R \in [0,1]$, it follows that

$$0 \le G_i \le \gamma^{C_i - 1} \sum_{h=0}^{\infty} \gamma^h = \frac{\gamma^{C_i-1}}{1-\gamma}.$$

We first prove (1). Define the renormalized returns $X_i := \gamma^{-(C_i-1)} G_i$. The pairs (C_i, G_i) are i.i.d., hence so are the X_i, and the pathwise bound gives $X_i \in [0, \frac{1}{1-\gamma}]$. By the tower property and the runtime factorization lemma,

$$\mathbb{E}\left[X_i\right] = \mathbb{E}\left[\gamma^{-(C_i-1)} \mathbb{E}\left[G_i | C_i\right]\right] = \mathbb{E}\left[\gamma^{-(C_i-1)} Q_{C_i}\right] = \mathbb{E}\left[\gamma^{-(C_i-1)} \gamma^{C_i-1} Q_1\right] = Q_1.$$

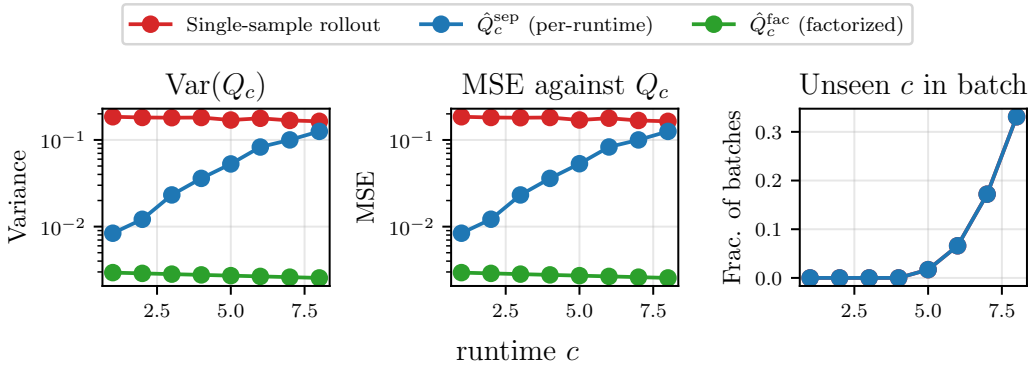


Figure 7: Experiment result on comparing different critic estimators.

Let $\hat{Q}_1 := \frac{1}{N}\sum_{i=1}^{N} X_i$. By Hoeffding's inequality, with probability at least $1-\delta$,

$$\left|\hat{Q}_1 - Q_1\right| \le \frac{1}{1-\gamma}\sqrt{\frac{\ln(2/\delta)}{2N}}.$$

On this single event, for every $c \in \mathbb{N}^+$ simultaneously—no union bound is needed, as the estimates for all runtimes are deterministic functions of $\hat{Q}_1$—this gives

$$\left|\hat{Q}_c^{\text{fac}} - Q_c\right| = \left|\gamma^{c-1}\hat{Q}_1 - \gamma^{c-1}Q_1\right| = \gamma^{c-1}\left|\hat{Q}_1 - Q_1\right| \le \frac{\gamma^{c-1}}{1-\gamma}\sqrt{\frac{\ln(2/\delta)}{2N}}.$$

We now prove (2). Condition on the runtime draws $(C_1, \ldots, C_N)$, which fixes the counts N_c. Conditionally, the returns $\{G_i : C_i = c\}$ are independent with mean Q_c and range $\left[0, \frac{\gamma^{c-1}}{1-\gamma}\right]$ by the pathwise bound. For each $c \in \mathcal{C}$ with $N_c \ge 1$, Hoeffding's inequality with failure probability δ/C gives

$$\mathbb{P}\left(\left|\hat{Q}_c^{\text{sep}} - Q_c\right| > \frac{\gamma^{c-1}}{1-\gamma}\sqrt{\frac{\ln(2C/\delta)}{2N_c}}\middle| C_{1:N}\right) \le \frac{\delta}{C},$$

and a union bound over $c \in \mathcal{C}$ yields conditional failure probability at most δ. Since this holds for every realization of $C_{1:N}$, it holds unconditionally. □

In other words, the separate estimator provides no estimate whatsoever for any runtime outside the empirical support $\{C_1, \ldots, C_N\}$—with countably many runtimes, any finite batch leaves infinitely many Q_c unestimated—whereas the factorized estimator covers the entire curve $c \mapsto Q_c$, including runtimes that were never executed. In §C.1 we further show that this runtime-factorized Q-parameterization for actor-critic algorithms also enjoys a better variance reduction than that without the factorization.

In Figure 7, we compare the squared error and the variance of $\hat{Q}^{\text{fac}}$ and $\hat{Q}^{\text{sep}}$ in the FrozenLake environment (Towers et al., 2025). Both Q-functions are tabular and are trained for M iterations using EMA, with $\eta = 0.1$, over N Monte-Carlo returns of a trained policy. The policy architecture is similar to PonderNet (Banino et al., 2021) where it loops through a linear layer followed by a tanh function whenever the halt predictor does not halt. The policy is trained using actor critic where the score is the advantage $\hat{A} = \hat{Q}^{\text{fac}} - \hat{V}$. In Figure 7, right, we show the proportional of batches where there are no samples for a particular runtime c, thus we expect $\hat{Q}^{\text{sep}}$ to perform worse on said c when compared to $\hat{Q}^{\text{fac}}$. Here, we can see that $\hat{Q}^{\text{fac}}$ has significantly lower squared error and variance compared to $\hat{Q}^{\text{sep}}$. Furthermore, notice that for larger c both quantities are even lower—this is expected as for larger runtime c the γ discounting exponentially shrinks the expected return, approaching zero.

C.1 VARIANCE REDUCTION WITH RUNTIME-FACTORIZED ADVANTAGE

In practice, researchers apply actor-critic methods (Konda & Tsitsiklis, 1999) or REINFORCE with baseline (Mei et al., 2022), which can reduce variance and the aggressiveness of the updates. We now show that a runtime-factorized version of the advantage can further reduce the variance of the estimator when comparing to that of the standard advantage function.

Fix any state $s \in \mathcal{S}$. Consider the single-sample REINFORCE estimators, with samples $(A, C) \sim \pi(\cdot|s)$, drawn independently of the critic batches $\mathcal{D} := (C_i, G_i)_{i \le N}$, a baseline $b = V^\pi(s)$, and advantages $\hat{Q} - b$:

$$\hat{g}_{\text{fac}} := u(A, C) \left(\gamma^{C-1} \hat{Q}^{\text{fac}}(A, 1) - b \right), \qquad \hat{g}_{\text{sep}} := u(A, C) \left(\hat{Q}^{\text{sep}}(A, C) - b \right),$$

where $u(A, C) = \nabla_\pi \log \pi(A, C|s) \in \mathbb{R}^d$. There are two sources of randomness here: (1) the action sample $\bar{A} := (A, C) \sim \pi(\cdot|s)$, the pair whose score is weighted in the gradient estimate; (2) the critic data $\mathcal{D} := (\bar{A}_i, G_i)_{i \le N}$, where the N i.i.d. pairs are drawn from μ. Here, μ generates a pair as follows: μ first generates a trajectory using π starting at state $(s_0, 0)$, it then sets $\bar{A}$ to be first action and G to be the cumulative return of the generated trajectory.

Since the Q-function $\hat{Q}$ is learned using $\mathcal{D}$, we can characterize its error when compared to Q as

$$\varepsilon(a, c\,; \mathcal{D}) := \hat{Q}(a, c\,; \mathcal{D}) - Q(a, c).$$

For the factorized Q-estimate and the separate Q-estimate, their corresponding errors are respectively

$$\varepsilon_{\text{fac}}(a, c\,; \mathcal{D}) = \gamma^{c-1} \left(\hat{Q}_1(a\,; \mathcal{D}_a) - Q(a, 1) \right), \qquad \varepsilon_{\text{sep}}(a, c\,; \mathcal{D}) = \hat{Q}_c^{\text{sep}}(a\,; \mathcal{D}_a) - Q(a, c),$$

where $\mathcal{D}_a$ denotes the samples where the first environment action is $A_0 = a$. We first show that the variance of both estimators can be decomposed similarly if they are unbiased.

Lemma 15 (Variance decomposition of policy gradients)**.** Fix a state $s \in \bar{\mathcal{S}}$ and a policy $\pi \in \Pi$. Assuming that $\mathbb{E}_{\mathcal{D} \sim \mu} [\varepsilon(a, c\,; \mathcal{D})] = 0$, then the policy gradient estimator $\hat{g} := u(A, C) \left(\hat{Q}(A, C\,; \mathcal{D}) - b \right)$ satisfies:

1. $\mathbb{E}_{\pi \otimes \mu} [\hat{g}] = g(s) := \mathbb{E}_{(a,c) \sim \pi(\cdot|s)} [u(a, c) (Q(a, c) - b)]$
2. $\text{Var}_{\pi \otimes \mu} (\hat{g}) = \text{Var}_{(a,c) \sim \pi(\cdot|s)} \Big(u(a, c) (Q(a, c) - b) \Big) + \mathbb{E}_{(a,c) \sim \pi(\cdot|s)} \left[\|u(a, c)\|^2 v(a, c) \right]$,

where $v(a, c) := \text{Var}_{\mathcal{D} \sim \mu} \left(\hat{Q}(a, c\,; \mathcal{D}) \right)$ and $\text{Var}_{\pi \otimes \mu} (\hat{g}) = \mathbb{E}_{\pi \otimes \mu} \left[\|\hat{g} - \mathbb{E}[\hat{g}]\|^2 \right]$.

Proof. We first add and subtract $Q(A, C)$ to $\hat{g}$ to decompose the estimator into a term depending only on the action sample and a term depending on both sources of randomness:

$$\hat{g} = \underbrace{u(A, C) (Q(A, C) - b)}_{=: T_1} + \underbrace{u(A, C) \varepsilon(A, C\,; \mathcal{D})}_{=: T_2}.$$

We first show unbiasedness of $\hat{g}$. Inspecting $\mathbb{E}_{\pi \otimes \mu} [T_2]$, we have that

$$\mathbb{E}_{\pi \otimes \mu} [T_2] = \mathbb{E}_{(a,c) \sim \pi(\cdot|s)} \Big[u(a, c) \underbrace{\mathbb{E}_{\mathcal{D} \sim \mu} [\varepsilon(a, c\,; \mathcal{D})]}_{= 0 \text{ by assumption}} \Big] = \mathbf{0}_d,$$

hence $\mathbb{E}_{\pi \otimes \mu} [\hat{g}] = \mathbb{E}_{\pi \otimes \mu} [T_1] = \mathbb{E}_{(a,c) \sim \pi(\cdot|s)} [u(a, c)(Q(a, c) - b)] = g(s)$.

We now analyze the variance of this estimator. We have that

$$\begin{aligned} \mathbb{E}_{\pi \otimes \mu} \left[\|\hat{g} - \mathbb{E}[\hat{g}]\|^2 \right] &= \mathbb{E}_{\pi \otimes \mu} \Big[\|T_1 + T_2 - \mathbb{E}[T_1] - \underbrace{\mathbb{E}[T_2]}_{=0} \|^2 \Big] \\ &= \mathbb{E}_{\pi \otimes \mu} \Big[\|T_1 - \mathbb{E}[T_1]\|^2 \Big] + \mathbb{E}_{\pi \otimes \mu} \Big[\|T_2\|^2 \Big] - 2 \mathbb{E}_{\pi \otimes \mu} \Big[(T_1 - \mathbb{E}[T_1])^\top T_2 \Big] \end{aligned}$$

Consider the cross term, we have that

$$\begin{aligned}\mathbb{E}_{\pi\otimes\mu}\Big[\left(T_1-\mathbb{E}\left[T_1\right]\right)^\top T_2\Big] &= \mathbb{E}_{\pi\otimes\mu}\left[T_1^\top T_2\right]-\mathbb{E}\left[T_1\right]\underbrace{\mathbb{E}\left[T_2\right]}_{=0}\\ &= \mathbb{E}_{\pi\otimes\mu}\Big[\left(Q(A,C)-b\right)u(A,C)^\top u(A,C)\,\varepsilon(A,C\,;\mathcal{D})\Big]\\ &= \mathbb{E}_{\pi\otimes\mu}\left[\left(Q(A,C)-b\right)\|u(A,C)\|^2\,\varepsilon(A,C\,;\mathcal{D})\right]\\ &= \mathbb{E}_{(a,c)\sim\pi(\cdot|s)}\Big[\left(Q(a,c)-b\right)\|u(a,c)\|^2\,\underbrace{\mathbb{E}_{\mathcal{D}\sim\mu}\left[\varepsilon(a,c\,;\mathcal{D})\right]}_{=0\text{ by assumption}}\Big]\\ &= 0.\end{aligned}$$

For the T_1 term, we have that

$$\begin{aligned}\mathbb{E}_{\pi\otimes\mu}\Big[\|T_1-\mathbb{E}\left[T_1\right]\|^2\Big] &= \mathrm{Var}(T_1)\\ &= \mathrm{Var}_{(a,c)\sim\pi(\cdot|s)}\Big(u(a,c)\left(Q(a,c)-b\right)\Big),\end{aligned}$$

which is the REINFORCE-with-baseline variance where we use the true Q-function.

For the T_2 term, since $\mathbb{E}_{\pi\otimes\mu}[T_2]=0$, we have that

$$\begin{aligned}\mathbb{E}_{\pi\otimes\mu}\left[\|T_2\|^2\right] &= \mathrm{Var}(T_2)\\ &= \mathbb{E}_{(a,c)\sim\pi(\cdot|s)}\Big[\|u(a,c)\|^2\,\underbrace{\mathbb{E}_{\mathcal{D}\sim\mu}\left[\varepsilon(a,c\,;\mathcal{D})^2\right]}_{=v(a,c)\text{ by assumption}}\Big].\end{aligned}$$

Putting everything together we obtain the desired result. □

Note that using a similar argument in Lemma 6, we have the following:

Lemma 16. Fix a walltime-independent policy π and $(s,a)\in\bar{\mathcal{S}}\times\bar{\mathcal{A}}$. Let $G^{(c)}$ denote the return of a rollout with $\bar{S}_0=(s,0)$ and $A_0=(a,c)$ that follows π thereafter. Then for every finite $c\in\mathbb{N}^+$, we have that

$$G^{(c)}\overset{d}{=}\gamma^{c-1}G^{(1)}.$$

In particular, the renormalized return $X:=\gamma^{-(c-1)}G^{(c)}$ satisfies $\mathbb{E}[X]=Q^\pi((s,0),(a,1))$ and $\mathrm{Var}(X)=\sigma_1^2(s,a):=\mathrm{Var}(G^{(1)})$, independently of c.

Proof. The proof uses a similar idea as in Lemma 6. Specifically, the process $(S_h,A_h,C_h)_{h\geq1}$ given $(S_0,A_0)=(s,a)$ does not depend on the first runtime C_0 as the environment kernel P depends only on (s,a) and π is walltime-independent. Coupling the two rollouts on the same realization of this process, the discount exponent of the h-th reward is $(C_0-1)+\sum_{h'=1}^{h}(C_{h'}-1)$, so every reward under $C_0=c$ equals γ^{c-1} times the corresponding reward under $C_0=1$ (both being 0 on the event $\sum_{h'=1}^{h}(C_{h'}-1)=\infty$). Hence $G^{(c)}=\gamma^{c-1}G^{(1)}$ pathwise under the coupling, which implies equality in distribution. The moment identities follow immediately. □

Consequently we can obtain the exact expression for the two critics' variances. For a batch of N rollouts at a fixed $(s,a)\in\mathcal{S}\times\mathcal{A}$, when conditioning on the sampled runtime $C_{1:N}$, each $X_i=\gamma^{-(C_i-1)}G_i$ is distributionally equal to $G^{(1)}$ regardless of C_i, and each G_i with $C_i=c$ is distributionally equal to $\gamma^{c-1}G^{(1)}$. Hence, for every $c\in\mathbb{N}^\infty$, we have that

$$\mathrm{Var}\left(\gamma^{c-1}\hat{Q}_1^{\mathrm{fac}}|C_{1:N}\right)=\gamma^{2(c-1)}\mathrm{Var}(\hat{Q}_1\mid C_{1:N})=\frac{\gamma^{2(c-1)}}{N^2}\sum_{i=1}^{N}\mathrm{Var}(X_i\mid C_i)=\frac{\gamma^{2(c-1)}\sigma_1^2(s,a)}{N},$$

$$\mathrm{Var}\left(\hat{Q}_c^{\mathrm{sep}}|C_{1:N}\right)=\frac{\gamma^{2(c-1)}\sigma_1^2(s,a)}{N_c}=\frac{N}{N_c}\mathrm{Var}\left(\gamma^{c-1}\hat{Q}_1^{\mathrm{fac}}|C_{1:N}\right),$$

where $\hat{Q}_1^{\text{fac}} = \frac{1}{N}\sum_{i=1}^{N}\gamma^{-(C_i-1)}G_i$ and $\hat{Q}_c^{\text{sep}} = \frac{1}{N_c}\sum_{i:C_i=c}G_i$. The per-runtime critic is thus exactly $N/N_c \geq 1$ times noisier at every runtime (and this eventually propagates directly into the policy gradient). Taking $N \to \infty$, the empirical counts is replaced by the runtime sampling distribution ρ through the almost-sure convergence $N_c/N \to \rho(c)$. We then obtain the following result:

Lemma 17. Fix a $c \in \text{supp}(\rho)$. As $N \to \infty$,

$$\sqrt{N}\left(\hat{Q}_c^{\text{fac}} - Q(c)\right) \xrightarrow{d} \mathcal{N}\left(0, \gamma^{2(c-1)}\sigma_1^2(a)\right),$$
$$\sqrt{N}\left(\hat{Q}_c^{\text{sep}} - Q(c)\right) \xrightarrow{d} \mathcal{N}\left(0, \frac{\gamma^{2(c-1)}\sigma_1^2(a)}{\rho(c)}\right).$$

Proof. We first focus on the factorized estimator. The renormalized returns $X_i := \gamma^{-(C_i-1)}G_i$ are i.i.d., and by Lemma 16, X_i given C_i is distributionally equal to $G^{(1)}$ regardless of C_i. Consequently $\mathbb{E}[X_i] = Q(1)$ and, by the law of total variance,

$$\text{Var}(X_i) = \mathbb{E}\left[\text{Var}(X_i|C_i)\right] + \text{Var}\left(\mathbb{E}[X_i|C_i]\right) = \sigma_1^2(a) + 0,$$

where the second term vanishes since $\mathbb{E}[X_i|C_i] = Q(1)$ is constant in C_i. By classical CLT, we have that $\sqrt{N}(\hat{Q}_1^{\text{fac}} - Q(1)) \xrightarrow{d} \mathcal{N}(0, \sigma_1^2(a))$. By Lemma 6, multiplying by the constant γ^{c-1} we get $Q(c) = \gamma^{c-1}Q(1)$. Combining both then gives us

$$\sqrt{N}\left(\hat{Q}_c^{\text{fac}} - Q(c)\right) \xrightarrow{d} \mathcal{N}\left(0, \gamma^{2(c-1)}\sigma_1^2(a)\right).$$

We now focus on the separate estimator. Write

$$\begin{aligned}\sqrt{N}\left(\hat{Q}_c^{\text{sep}} - Q(c)\right) &= \sqrt{N}\left(\sum_{i=1}^{N}\frac{\mathbb{1}\{C_i = c\}\left(G_i - Q(c)\right)}{N_c}\right) \\ &= \frac{\frac{1}{\sqrt{N}}\sum_{i=1}^{N}\mathbb{1}\{C_i = c\}\left(G_i - Q(c)\right)}{N_c/N}.\end{aligned}$$

The summands in the numerator are i.i.d. with mean zero and variance

$$\begin{aligned}\mathbb{E}\left[\mathbb{1}\{C_i = c\}^2\left(G_i - Q(c)\right)^2\right] &= \mathbb{E}_{c'\sim\rho}\left[\mathbb{E}\left[\mathbb{1}\{C_i = c\}(G_i - Q(c))^2|C_i = c'\right]\right] \\ &= \mathbb{E}_{c'\sim\rho}\left[\mathbb{1}\{c' = c\}\mathbb{E}\left[(G_i - Q(c))^2|C_i = c'\right]\right] \\ &= \mathbb{E}_{c'\sim\rho}\left[\mathbb{1}\{c' = c\}\text{Var}(G_i - Q(c)|C_i = c')\right] \\ &= \sum_{c'}\rho(c')\mathbb{1}\{c' = c\}\text{Var}(G_i - Q(c)|C_i = c') \\ &= \rho(c)\text{Var}(G_i - Q(c)|C_i = c) \\ &= \rho(c)\gamma^{2(c-1)}\sigma_1^2(a).\end{aligned}$$

Thus, by the CLT the numerator converges in distribution to $\mathcal{N}(0, \rho(c)\gamma^{2(c-1)}\sigma_1^2(a))$. By the strong law of large numbers, we have that $N_c/N \to \rho(c) > 0$ almost surely. Applying Slutsky's theorem gives us

$$\sqrt{N}\left(\hat{Q}_c^{\text{sep}} - Q(c)\right) \xrightarrow{d} \mathcal{N}\left(0, \frac{\gamma^{2(c-1)}\sigma_1^2(a)}{\rho(c)}\right),$$

with asymptotic variance $\rho(c)\gamma^{2(c-1)}\sigma_1^2(a)/\rho(c)^2 = \gamma^{2(c-1)}\sigma_1^2(a)/\rho(c)$. □

In other words, the asymptotic relative efficiency of the separate critic with respect to the factorized critic is $\rho(c) \leq 1$: the separate critic is asymptotically $1/\rho(c)$ times noisier at every runtime, uniformly in c. Consequently we obtain the following variance reduction result, where the factorized advantage generally has smaller variance than the standard advantage.

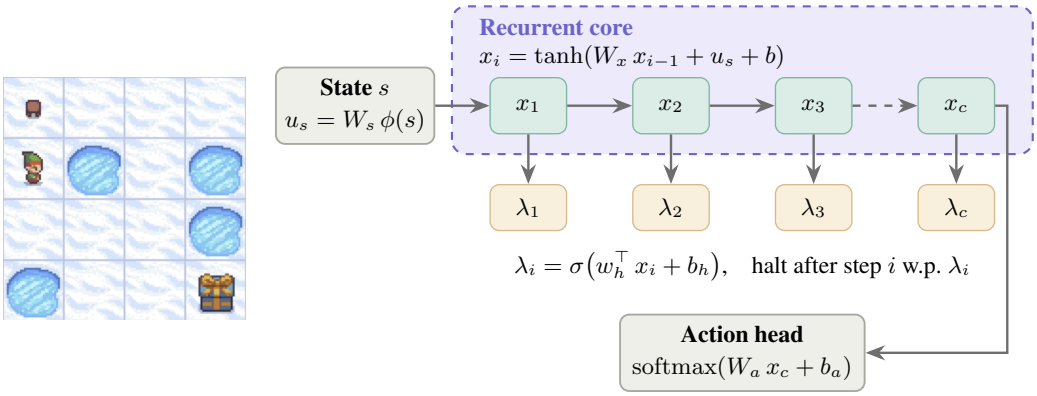


Figure 8: **(Left)** The FrozenLake environment. **(Right)** A policy with a PonderNet architecture.

Theorem 18 (Variance reduction of the factorized advantage)**.** Suppose $\mathrm{supp}(\pi(\cdot|s)) \subseteq \{(a,c) : c \in \mathrm{supp}(\rho_a)\}$. Then, we have that

$$v_{\mathrm{sep}} - v_{\mathrm{fac}} = \mathbb{E}_{(a,c)\sim\pi}\left[\|u(a,c)\|^2\, \gamma^{2(c-1)}\, \sigma_1^2(a)\, \frac{1-\rho_a(c)}{\rho_a(c)}\right] \geq 0,$$

with strict inequality if and only if there exists (a,c) in the support of $\pi(\cdot|s)$ with $u(a,c) \neq 0$, $\sigma_1^2(a) > 0$, and $\rho_a(c) < 1$. Consequently, $\mathrm{Var}_{\pi\otimes\mu}\left(\hat{g}_{\mathrm{sep}}\right) \geq \mathrm{Var}_{\pi\otimes\mu}\left(\hat{g}_{\mathrm{fac}}\right)$.

Proof. By Lemma 17, $v_{\mathrm{fac}}(a,c) = \gamma^{2(c-1)}\sigma_1^2(a)$ and $v_{\mathrm{sep}}(a,c) = \gamma^{2(c-1)}\sigma_1^2(a)/\rho_a(c)$. Subtracting inside the expectation,

$$v_{\mathrm{sep}}(a,c) - v_{\mathrm{fac}}(a,c) = \gamma^{2(c-1)}\sigma_1^2(a)\left(\frac{1}{\rho_a(c)} - 1\right) = \gamma^{2(c-1)}\sigma_1^2(a)\,\frac{1-\rho_a(c)}{\rho_a(c)} \geq 0,$$

since $\rho_a(c) \leq 1$. Each summand is nonnegative, and a summand is strictly positive exactly when $u(a,c) \neq 0$, $\sigma_1^2(a) > 0$, and $\rho_a(c) < 1$; the expectation is strictly positive if and only if multiple c's have positive mass for π taking action a at state s. Plugging this result into statement (2) of Lemma 15 we obtain the desired variance reduction result. □

Toy experiments. We illustrate in the FrozenLake environment (Towers et al., 2025) that the runtime-factorized advantage enables the agent to learn more efficiently compared to that with other advantage variants. This environment is a grid-world reaching task with a slipping mechanism that shifts the agent to adjacency cell with some probability upon taking any action (Figure 8, left). In our setting, the environment state is a scalar $s \in \{0, \ldots, 15\}$ corresponding to the s'th cell of the grid, counting from left to right then from top to bottom; and the environment action is the four cardinal directions. The agent always starts at state $s = 0$ (the top-left cell) and its goal is to reach $s = 15$ (the bottom-right cell). Upon reaching the goal, the episode terminates and the environment emits a $+1$ reward. There are a subset of states $s \notin \{0, 15\}$ such that the episode terminates immediately (the hole cells) with a $+0$ reward. We limit the agent to interact for only 100 timesteps per episode and set $\gamma = 0.99$. Note that in this scenario, a linear policy cannot achieve optimal performance as the optimal paths are not linearly separable (Chan et al., 2026).

We compare four RL agents: (1) advantage actor critic (A2C) with $\hat{Q}^{\mathrm{fac}}$, (2) A2C with $\hat{Q}^{\mathrm{sep}}$, (3) REINFORCE with Monte-carlo return, and (4) A2C with fixed computation $c = 1$. Note that we do not restrict c for all other variants. For each RL agent we train a PonderNet policy (Figure 8, right) with a recurrent core consisting a linear layer followed by a tanh function. Upon receiving a halting signal, the model projects the latent vector to logits followed by a softmax function. All Q-functions and value functions use a tabular representation. All agents are trained with 10 seeds and we show the mean and standard error in Figure 9.

We can see that all variants with pondering outperforms the variant without pondering in both discounted and undiscounted return, suggesting that pondering is necessary for an improved perfor-

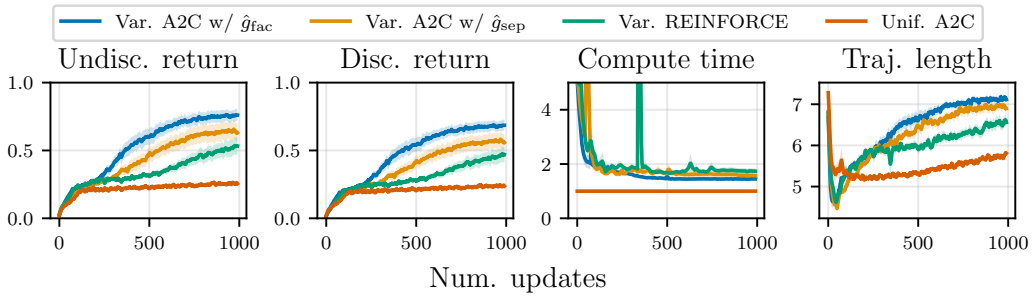


Figure 9: Training curve on policy gradient with different score functions in the `FrozenLake` environment. Each line is the mean over 10 seeds and the shaded area is the 95% confidence interval.

mance. As expected, even though all estimators are unbiased, the performance improves as the Q-estimator variance decreases—the runtime factorization comes out on top of other variants (recall in Figure 7 that it has the lowest variance). Furthermore, we can see that both REINFORCE with Monte-carlo return and A2C with $\hat{Q}^{\text{sep}}$ have sudden spikes in the runtime per decision over training. We hypothesize that this is due to the fact that runtime factorization has generalization across runtime, while these two variants do not. Specifically with $\hat{Q}^{\text{fac}}$ correctly identifies that the Q-value decreases with increasing c, while $\hat{Q}^{\text{sep}}$ might overestimate the Q-value for increasing but unseen c.

D Runtime-aware Bellman operator

In this section we will focus on deterministic policies Π to illustrate that naively extending the Bellman operator is challenging. Consider a class of deterministic policies Π, we define the action-restricted Bellman operator to be $(\mathcal{T}^{\Pi} f)(\bar{s}) := \max_{\bar{a} \in \bar{\mathcal{A}}^{\Pi}(\bar{s})} r(\bar{s}, \bar{a}) + \gamma \mathbb{E}_{\bar{s}'}[f(\bar{s}')]$, where $\bar{\mathcal{A}}^{\Pi}(\bar{s}) = \{\bar{a} \in \bar{\mathcal{A}} | \exists \pi \in \Pi, \pi(\bar{s}) = (a, c) =: \bar{a}\}$ is the set of actions that can be generated by some policy $\pi \in \Pi$ at state $\bar{s} \in \bar{\mathcal{S}}$. Recall that c is the runtime required to compute an action using the policy π at state $\bar{s}$. The Bellman operator $\mathcal{T}^{\Pi}$ then satisfies similar properties to that of the Bellman operator defined in standard MDPs. For example, $\mathcal{T}^{\Pi}$ is a γ-contraction in the supremum norm, thus there is a unique fixed point, call it V^{Π}. Other useful properties including monotonicity and improvement follow as well, all of which can be proved using similar techniques in the standard MDP setting.

Lemma 19 (γ-contraction of $\mathcal{T}^{\Pi}$)**.** Let $f, g : \bar{\mathcal{S}} \to \mathbb{R}$. We have that $\|\mathcal{T}^{\Pi} f - \mathcal{T}^{\Pi} g\|_{\infty} \leq \gamma \|f - g\|_{\infty}$.

Proof. For any state $\bar{s} \in \bar{\mathcal{S}}$, we have that

$$
\begin{aligned}
&|(\mathcal{T}^{\Pi} f - \mathcal{T}^{\Pi} g)(\bar{s})| \\
=&\left| \left(\max_{a \in \mathcal{A}^{\Pi}(\bar{s})} r(\bar{s}, a) + \gamma \mathbb{E}_{\bar{s}'}[f(\bar{s}')] \right) - \left(\max_{a \in \mathcal{A}^{\Pi}(\bar{s})} r(\bar{s}, a) + \gamma \mathbb{E}_{\bar{s}'}[g(\bar{s}')] \right) \right| \\
\leq& \max_{a \in \mathcal{A}^{\Pi}(\bar{s})} \gamma |\mathbb{E}_{\bar{s}'}[f(\bar{s}') - g(\bar{s}')]|,
\end{aligned}
$$

where we use the inequality $|\max_{a_1} f_1(a_1) - \max_{a_2} f_2(a_2)| \leq \max_a |f_1(a) - f_2(a)|$. Let $a^{\Pi}(\bar{s}) = \arg\max_{a \in \mathcal{A}^{\Pi}(\bar{s})} \gamma |\mathbb{E}_{\bar{s}'}[f(\bar{s}') - g(\bar{s}')]|$, we then have

$$
\begin{aligned}
&\gamma \left| \sum_{\bar{s}' \in \bar{\mathcal{S}}} P(\bar{s}' | \bar{s}, a^{\Pi}(\bar{s}))(f(\bar{s}') - g(\bar{s}')) \right| \\
\leq& \gamma \sum_{\bar{s}' \in \bar{\mathcal{S}}} P(\bar{s}' | \bar{s}, a^{\Pi}(\bar{s}))|(f(\bar{s}') - g(\bar{s}'))| \\
\leq& \gamma \sum_{\bar{s}' \in \bar{\mathcal{S}}} P(\bar{s}' | \bar{s}, a^{\Pi}(\bar{s}))\|u - v\|_{\infty} \\
\leq& \gamma \|u - v\|_{\infty},
\end{aligned}
$$

where the first inequality comes from Hölder's inequality. □

Lemma 20 (Monotonicity of $\mathcal{T}^{\Pi}$)**.** Let $f, g : \bar{\mathcal{S}} \to \mathbb{R}$ such that for all $\bar{s} \in \bar{\mathcal{S}}$ we have $f(\bar{s}) \leq g(\bar{s})$. Then for any $\bar{s} \in \bar{\mathcal{S}}$ we have $(\mathcal{T}^{\Pi} f)(\bar{s}) \leq (\mathcal{T}^{\Pi} g)(\bar{s})$.

Proof. Since $f(\bar{s}) \leq g(\bar{s})$ for all $\bar{s} \in \bar{\mathcal{S}}$, taking the expectation over the same distribution preserves the inequality, i.e., $\mathbb{E}_s[f(\bar{s})] \leq \mathbb{E}_s[g(\bar{s})]$. Thus for any action $a \in \mathcal{A}$ we have

$$r(\bar{s}, a) + \mathbb{E}_{\bar{s}'}[f(\bar{s}')] \leq r(\bar{s}, a) + \mathbb{E}_{\bar{s}'}[g(\bar{s}')].$$

Taking the supermum over $\mathcal{A}^{\Pi}(\bar{s})$ we obtain the result. □

Lemma 21. Let $\hat{V}^{\Pi}(\bar{s}) = \sup_{\pi \in \Pi} V^{\pi}(\bar{s})$ for all $\bar{s} \in \bar{\mathcal{S}}$. Then for any $\bar{s} \in \bar{\mathcal{S}}$ we have that $(\mathcal{T}^{\Pi} \hat{V}^{\Pi})(\bar{s}) \geq \hat{V}^{\Pi}(\bar{s})$.

Proof. Consider $\pi \in \Pi$. First, we have that $V^{\pi}(\bar{s}) \leq \hat{V}^{\Pi}(\bar{s})$ by definition. Applying $\mathcal{T}^{\pi}$ on both sides we get $V^{\pi}(\bar{s}) \leq \mathcal{T}^{\pi} \hat{V}^{\Pi}(\bar{s}) \leq (\mathcal{T}^{\Pi} \hat{V}^{\Pi})(\bar{s})$. Taking supremum over $\pi \in \Pi$ we get $\hat{V}^{\Pi}(\bar{s}) \leq (\mathcal{T}^{\Pi} \hat{V}^{\Pi})(\bar{s})$. □

If Π is such that the greedy policy w.r.t. $\hat{V}^{\Pi}$ is realizable we can show that $\hat{V}^{\Pi}(\bar{s}) = (\mathcal{T}^{\Pi} \hat{V}^{\Pi})(\bar{s})$. Generally, the policy that realizes V^{Π} may lie outside of Π, i.e., $V^{\Pi}(\bar{s}) \geq \sup_{\pi \in \Pi} V^{\pi}(\bar{s})$, due to the fact that for all $\pi \in \Pi$, $\bar{\mathcal{A}}^{\{\pi\}}(\bar{s}) \subseteq \bar{\mathcal{A}}^{\Pi}(\bar{s})$. This is similar to the delusional bias where there is no policy that can realize the Bellman backup step in the Bellman operator (Lu et al., 2018)—in our scenario there might not exist a single policy that can uniformly output the best action and runtime for every state. One way to ensure equality is by assuming *runtime closure*:

Definition 22 (Runtime closure)**.** Define $(a^{\pi}(\bar{s}), c^{\pi}(\bar{s})) = \pi(\bar{s})$ to be the action a computed by π with runtime c at state $\bar{s}$. A policy class Π satisfies the runtime closure property if there is a runtime function $\tau : \bar{\mathcal{S}} \times \bar{\mathcal{A}} \to \mathbb{N}^{\infty}$ such that:

1. For any policy $\pi \in \Pi$, its runtime is lower bounded by τ, i.e., for every state $\bar{s} \in \bar{\mathcal{S}}$, $c^{\pi}(\bar{s}) \geq \tau(\bar{s}, a^{\pi}(\bar{s}))$.
2. For any mapping $f : \bar{\mathcal{S}} \to \bar{\mathcal{A}}$, there is a policy $\pi \in \Pi$ such that $f(\bar{s}) = a^{\pi}(\bar{s})$ and $c^{\pi}(\bar{s}) = \tau(\bar{s}, a^{\pi}(\bar{s}))$ for all $\bar{s} \in \bar{\mathcal{S}}$.

Theorem 23. Suppose Π satisfies runtime closure, then for any $\bar{s} \in \bar{\mathcal{S}}$ we have that $V^{\Pi}(\bar{s}) = \hat{V}^{\Pi}(\bar{s})$.

Proof. Define the greedy action and the greedy policy of V^{Π} at state $\bar{s}$ to respectively be

$$(a^{\Pi}(\bar{s}), c^{\Pi}(\bar{s})) = \arg\max_{\bar{a} \in \bar{\mathcal{A}}^{\Pi}} r(\bar{s}, a) + \gamma \mathbb{E}_{\bar{s}'}\left[V^{\Pi}(\bar{s}')\right], \qquad f^{\Pi}(\bar{s}) = a^{\Pi}(\bar{s}).$$

By the runtime closure property, there is an $o \in \Pi$ such that for all $\bar{s} \in \bar{\mathcal{S}}$, we have $a^{o}(\bar{s}) = f^{\Pi}(\bar{s})$ and $c^{o}(\bar{s}) = \tau(\bar{s}, f^{\Pi}(\bar{s}))$, where $c^{o}(\bar{s})$ is the minimal runtime within the policy class. Furthermore, increasing runtime $c^{o}(\bar{s})$ by any positive K decreases the expected return by at least γ^{K}. Thus, we have that $(\mathcal{T}^{o} V^{\Pi})(\bar{s}) = (\mathcal{T}^{\Pi} V^{\Pi})(\bar{s}) = V^{\Pi}(\bar{s})$, where $\mathcal{T}^{o}$ is the expected Bellman operator of π^{o}. Note that both V^{Π} and V^{o} are the unique fixed points for the respective Bellman operators due to Lemma 19 and the γ-contraction property of $\mathcal{T}^{o}$. Since $V^{o}(\bar{s}) = (\mathcal{T}^{o} V^{o})(\bar{s})$ and $(\mathcal{T}^{o} V^{\Pi})(\bar{s}) = V^{\Pi}(\bar{s})$, by the uniqueness of fixed point theorem we must have $V^{o} = V^{\Pi}$. Finally, we have $\hat{V}^{\Pi}(\bar{s}) = \sup_{\pi \in \Pi} V^{\pi}(\bar{s}) \geq V^{o}(\bar{s}) = V^{\Pi}(\bar{s})$. By Lemmas 20 and 21, we have $\hat{V}^{\Pi}(\bar{s}) \leq (\mathcal{T}^{\Pi} \hat{V}^{\Pi})(\bar{s}) \leq (\mathcal{T}^{\Pi^2} \hat{V}^{\Pi})(\bar{s}) \leq \cdots \leq (\mathcal{T}^{\Pi^{\infty}} \hat{V}^{\Pi})(\bar{s}) = V^{\Pi}(\bar{s})$, so it must be that $V^{\Pi}(\bar{s}) = \hat{V}^{\Pi}(\bar{s})$. □

Unfortunately, the runtime closure property can easily be violated if the considered policy class Π has bounded description length (Finzi et al., 2026), time complexity (Ghugare et al., 2026), and circuit complexity (Merrill & Sabharwal, 2023). Indeed, this is a direct consequence of Theorem 11.

With finite state space, a simple counting argument shows that if the policy class is too restrictive then the runtime closure property is still easily violated. For example, consider a program alphabet Σ and fixed description length ℓ, then there are at most $N = |\Sigma|^\ell$ possible programs—if $N < |\bar{\mathcal{S}}||\bar{\mathcal{A}}|$ the policy class containing length-ℓ programs cannot represent all possible state-to-action mappings.

E RUNTIME-AWARE DYNAMIC PROGRAMMING

In this section we explore a dynamic-programming (planning) algorithm that can obtains a best-in-class policy. The idea here is to extend policy-class value iteration (PCVI) (Lu et al., 2018). We only focus on walltime-independent policies. Formally, let Π be the policy class and $[s \mapsto (a, c)] = \{\pi \in \Pi | \pi(s) = (a, c)\}$. The information set $X \subseteq \Pi$ is a set of policies that is consistent with a particular Q-value q to some (s, a, c) tuple. We paritition the set of policies into non-empty subsets $\mathcal{X} = \{X_1, \ldots, X_k\}$ where $X_1 \cup \cdots \cup X_k = \Pi$ and $X_i \cap X_j = \emptyset$ for $i \neq j$, and we denote 2^Π to be the set of all finite partitions of Π. A partition function $h : \mathcal{X} \to \mathbb{R}$ assigns values to all partitions of $\mathcal{X}$. Let $\mathcal{H} = \{h : \mathcal{X} \to \mathbb{R} | \mathcal{X} \in 2^\Pi\}$ denote the set of all partition functions. For $h_1, h_2 \in \mathcal{H}$, the intersection sum is

$$(h_1 \oplus h_2)(X_1 \cap X_2) = h_1(X_1) + h_2(X_2), \forall X_1 \in \mathsf{dom}(h_1), X_2 \in \mathsf{dom}(h_2), X_1 \cap X_2 \neq \emptyset.$$

The intersection sum satisfies the following properties that will enable finding the best-in-class policy.

Proposition 24 (Proposition 6 of Lu et al. (2018))**.** For any $h, h_1, h_2, h_3 \in \mathcal{H}$, the following properties hold for the intersection sum.

- (Identity.) Let $e = X \mapsto 0$, then $h \oplus e = e \oplus h = h$.
- (Refinement.) We have (1) $h_1 \oplus h_2$ is a refinement of h_1 and of h_2, and (2) if h_1' and h_2' is a refinement of h_1 and h_2 respectively, then $h_1' \oplus h_2'$ is a refinement of $h_1 \oplus h_2$.
- (Closure.) $h_1 \oplus h_2 \in \mathcal{H}$.
- (Commutative.) $h_1 \oplus h_2 = h_2 \oplus h_1$.
- (Associative.) $(h_1 \oplus h_2) \oplus h_3 = h_1 \oplus (h_2 \oplus h_3)$.

Proof. See Appendix A.6 of Lu et al. (2018) for the complete proof. □

The goal is to keep track of the policy-consistent backups which is achieved through information sets. With an assumption to the witness oracle we outline the original PCVI algorithm in Algorithm 1.

Assumption 1 (Witness oracle)**.** We have access to an oracle $\mathsf{Witness}$ where for any $X \subseteq \Pi$ defined by a conjunction of a collection of state-to-action constraints, outputs $\emptyset$ if X is an inconsistent set of constraints, and outputs any witness $\pi \in X$ otherwise.

Algorithm 1 PCVI

Require: $\mathcal{S}$, $\mathcal{A}$, P, r, γ, Π, initial state s_0.

$\mathtt{Q}[s, a] \leftarrow$ initialize to mapping $\mathcal{A}^\mathcal{S} \to 0$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$
$\mathtt{ConQ}[s, a] \leftarrow$ initialize to mapping $[s \mapsto a] \to 0$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$
repeat
  for $(s, a) \in \mathcal{S} \times \mathcal{A})$ **do**
    $\mathtt{Q}[s, a] \leftarrow R(s, a) + \gamma \oplus_{s' \in \mathcal{S}} P(s'|s, a)\mathtt{ConQ}[s']$
    $\mathtt{ConQ}[s, a](Z) \leftarrow \mathtt{Q}[s, a](X)$ for all X s.t. $Z = X \cap [s \mapsto a] \neq \emptyset$
    Update $\mathtt{ConQ}[s]$
  end for
until $\mathtt{Q}$ converges
$X^* = \arg\max_X \mathtt{ConQ}[\bar{s}_0](X)$
$q^* = \mathtt{ConQ}[\bar{s}_0](X^*)$
$\pi^* = \mathsf{Witness}(X^*)$
return π^*

Theorem 25 (PCVI Theorem (Lu et al., 2018))**.** Algorithm 1 has the following guarantees:

1. (Convergence and correctness). The function $\mathtt{Q}$ converges and, for each $s \in \mathcal{S}, a \in \mathcal{A}$, and any $\pi \in \Pi$: there is a unique $X \in \mathsf{dom}(\mathtt{Q}[s,a])$ such that $\pi \in X$ and $Q^\pi(s,a) = \mathtt{Q}[s,a](X)$.
2. (Optimality and non-delusion). π^* is an optimal policy within Π and q^* is its value.
3. (Runtime bound). $\mathtt{Q}$ converges in finite time.

Proof. See Appendix A.6 of Lu et al. (2018) for the complete proof. □

We note, however, that this algorithm is intractable (or even uncomputable) in general. For example, consider Π to be the set of Turing machines (TMs), identifying the partitions $[s \mapsto a]$ requires enumerating Π, possibly forever if not all state-action pairs are covered. To formalize this, note that with a countable policy class, determining whether a particular state-action pair that is reachable by any policy is undecidable, consequently determining whether a reachable state-action pair has positive reward is also undecidable.

Proposition 26. For an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R \rangle$ and a policy class Π, determining whether there exists a $\pi \in \Pi$ such that $(s,a) \in \mathcal{S} \times \mathcal{A}$ is reachable from an initial state $\bar{s}_0 \in \mathcal{S}$ is undecidable. Furthermore determining whether π obtains any positive reward in reachable state-action pairs from $\bar{s}_0$ is undecidable.

Proof. By contradiction suppose we have a decider L for deciding whether $(\mathcal{M}, \Pi, s, a, \bar{s}_0)$ satisfies that (s,a) is reachable from $\bar{s}_0$ by any policy in Π. Suppose we are given a TM T and an input string x, we wish to construct an MDP $\mathcal{M}$ and a policy class $\Pi(T,x)$. For the MDP $\mathcal{M}$, we define the state space to be a single state $\mathcal{S} = \{\varepsilon\}$ and the action space to contain two actions $\mathcal{A} = \{\bot, \top\}$. The dynamics $P = \delta(S' = \varepsilon)$ is a deterministic self-loop and the reward function is $R(\varepsilon, \bot) = 0$ and $R(\varepsilon, \top) = 1$.

The policy class we consider is the countable set $\Pi(T,x) = \{\pi_1, \pi_2, \dots\}$, where we define the t'th policy to be

$$\pi_t = \begin{cases} \top & \text{if } T(x) \text{ halts within } t \text{ steps,} \\ \bot & \text{if } T(x) \text{ does not halt within } t \text{ steps.} \end{cases}$$

We note that each policy in $\Pi(T,x)$ is computable as π_t must terminate within t steps. Furthermore, the policy class $\Pi(T,x)$ is enumerable thus there is a finite description that enumerates each policy.

We now construct L' that decides HALT. L' receives a Turing machine encoding $\langle T \rangle$ and an input string x, and run L on $\langle \mathcal{M}, \Pi(T,x), \varepsilon, \top, \varepsilon \rangle$. If L accepts then L' accepts, otherwise L' rejects. Running L' on input (T,x) results in two cases:

1. Suppose T halts on x. There exists a policy π_t such that T halts on x within t steps thus outputting the action $\top$. In other words, the state-action pair $(\varepsilon, \top)$ is reachable from ε. This is also when the reward is $R(\varepsilon, \top) = 1$, thus satisfying the positive achievable reward. Consequently L accepts $\langle \mathcal{M}, \Pi(T,x), \varepsilon, \top, \varepsilon \rangle$.
2. Suppose T does not halt on x. There is no policy that halts so all policies outputs the action $\bot$. As a result the state-action pair $(\varepsilon, \top)$ is not reachable. Furthermore the reward is always $R(\varepsilon, \bot) = 0$, thus not satisfying the positive achievable reward. Consequently L rejects $\langle \mathcal{M}, \Pi(T,x), \varepsilon, \top, \varepsilon \rangle$.

Clearly, L' decides HALT and thus a contradiction. □

E.1 Optimal policy identification in the limit

With only the Witness oracle we can consider a simpler setting and develop an algorithm that outputs a best-in-class policy in the limit—this is slightly looser than the language identification in the limit setting where we simply care about obtaining a function that is behaviourally identical to the optimal policy (Gold, 1967; Kleinberg & Mullainathan, 2024). Specifically, the algorithm is

allowed to output infinitely many times a policy, the goal is to have the algorithm always output a best-in-class policy beyond some iteration C (in our case the algorithm ignores any feedback). The core idea of the algorithm will be considering the best policy with runtime upper bounded by C, we outline the pseudocode in Algorithm 2. We define the set of policies $\Pi(C)$ as follows:

Definition 27. For $C \in \mathbb{N}^\infty$, we define $\Pi(C) = \{\pi \in \Pi | \forall s \in \mathcal{S}, c^\pi(s) \leq C \text{ or } c^\pi(s) = \infty\}$.

We note that identifying $\Pi(C)$ is undecidable as obtaining this set also solves the halting problem. Intuitively, for any finite MDPs, there could still be at least two policy implementations, say $\pi_1, \pi_2 \in \Pi(C)$, such that for all state $s \in \mathcal{S}$ they compute the same action with the same runtime, i.e., $\pi_1(s) = \pi_2(s)$. For example, we can have two Python codes with different docstrings or variable names—see Arora & Barak (2009) for a similar construction but for Turing machines. This creates an issue where the set $\Pi(C)$ can have infinite cardinality.

Algorithm 2 Runtime-aware PCVI in the limit

Require: $\mathcal{S}$, $\mathcal{A}$, P, R, γ, Π, initial state $\bar{s}_0$.

1: **for** $C := 1, 2, \ldots$ **do**
2: Get $\pi^* \leftarrow$ PCVI($\mathcal{S}, \mathcal{A} \times [C] \cup \{\infty\}, P, R, \gamma, \Pi(C), \bar{s}_0$)
3: Yield π^*
4: **end for**

We now prove the correctness of Algorithm 2, it may be helpful to first consider the following lemma:

Lemma 28. For any $0 < C < C'$, we have that $\Pi(C) \subseteq \Pi(C')$. Furthermore, there exists a $0 < C < C''$ such that $\Pi(C) \subsetneq \Pi(C'')$ if and only if $\Pi(C) \neq \Pi$.

Proof. By Definition 27, for every runtime bound $C \in \mathbb{N}^+$,

$$\Pi(C) = \{\pi \in \Pi | \forall s \in \mathcal{S}, c^\pi(s) \leq C \text{ or } c^\pi(s) = \infty\},$$

where $c^\pi(s)$ denotes the runtime required by π to compute an action at state s. Suppose $C \leq C'$ and consider any policy $\pi \in \Pi(C)$. It follows immediately that $c^\pi(s) \leq C'$ whenever the runtime is finite. so $\pi \in \Pi(C')$, establishing that $\Pi(C) \subseteq \Pi(C')$. Therefore applying the same argument inductively we have that the sequence $\Pi(0) \subseteq \Pi(1) \subseteq \cdots$ is monotone increasing.

For the second claim, observe that enlarging the runtime bound can only introduce policies whose maximal finite runtime exceeds the previous threshold. For the if direction, suppose $\Pi(C) \neq \Pi$. There is a policy $\pi \in \Pi$ but not $\Pi(C)$. Since $\Pi(C)$ includes those policies that never halts on any state, it must be the case that there is a state $s \in \mathcal{S}$ such that $c^\pi(s) > C$. Setting $C' = c^\pi(s)$ we have that $\pi \in \Pi(C')$ thus $\Pi(C) \subsetneq \Pi(C')$. For the only-if direction, consider the contrapositive. If $\Pi(C) = \Pi$, then for any $C' > C$ we have that $\Pi = \Pi(C) \subseteq \Pi(C') \subseteq \Pi$. □

Theorem 29. Algorithm 2 outputs an optimal policy in the limit.

Proof. At each iteration $C \in \mathbb{N}^+$, PCVI obtains an optimal policy within $\Pi(C)$. By Lemma 28, at some iteration $C' > C$ we will have that $\Pi(C) \subsetneq \Pi(C')$ unless $\Pi(C) = \Pi$. If $\Pi(C) = \Pi$, then we are done as PCVI always obtains an optimal policy in Π as the algorithm increases C. Otherwise, PCVI obtains an optimal policy that might still be suboptimal in Π. Since $\pi^* \in \Pi$, consider the maximal (finite) runtime $C_{\text{max}} = \max_{s \in \mathcal{S}: c^{\pi^*}(s) < \infty} c^{\pi^*}(s)$. At iteration C_{max}, the policy class $\Pi(C_{\text{max}})$ must include π^* by definition, thus PCVI obtains π^*. For $C' > C_{\text{max}}$, the algorithm must output an optimal policy since $\Pi(C_{\text{max}}) \subseteq \Pi(C')$, otherwise contradicting that π^* is optimal. If π^* does not halt at any state, then $\Pi(1)$ already includes all the policies that do not halt at any state, thus PCVI always obtains an optimal policy. □

Intuitively, as we increase the maximum runtime we consider more policies in Π—a variant of this algorithm where it aims to obtain the optimal Q-function is lower semicomputable with access to the Witness oracle.

E.2 KNOWING WHEN AN OPTIMAL POLICY IS OBTAINED

The next question is whether there is a way to know that an optimal policy has been obtained. To address this the algorithm has to determine whether a best-in-class policy is obtained, which is only possible when the algorithm is given access to extra oracles, both of which are, again, generally uncomputable: (1) An oracle is required to ensure all reachable state-action are considered by the algorithm—by Proposition 26 it is generally impossible to know if a policy obtains the optimal return. (2) An oracle is required to ensure that there is no obscured policy that takes unbounded runtime for some unreachable states even though it takes shortest runtime for the reachable states—this requires iterating through all policies and ensuring that there is no same state-to-action mapping with strictly better runtime.

In the following we consider a slight modification of the RAMDP, denoting $\bar{\mathcal{S}} := \mathcal{S} \cup \{s_{\mathsf{abs}}\} \times \mathbb{N}^{\infty}$ and $\bar{\mathcal{A}} := \mathcal{A} \times \mathbb{N}^{\infty}_{+}$. For any $(s, a) \in \mathcal{S} \times \mathcal{A}$, we write $\bar{s} := (s, w)$ and $\bar{a} := (a, c)$ for $(s, a) \in \mathcal{S} \times \mathcal{A}$ and $(w, c) \in \mathbb{N}^{\infty} \times \mathbb{N}^{\infty}_{+}$. We define a new transition function P and a new reward function r to be

$$P(\bar{s}, \bar{a}, \bar{s}') = \begin{cases} \delta(s' = s_{\mathsf{abs}})\delta(w' = \infty) & w + c = \infty \\ P(s, a, s')\delta(w' = w + c - 1) & \text{otherwise.} \end{cases},$$

$$R(\bar{s}, \bar{a}) = \begin{cases} 0 & w + c = \infty \\ \gamma^{w+c-1} r(s, a) & \text{otherwise.} \end{cases}$$

If the runtime $c = \infty$ at a given state, then effectively the environment transitions to an absorbing state s_{abs} and the policy will no longer obtain any reward onwards in the environment. Now, consider the occupancy measure of the environment state-and-action pairs:

Definition 30 (Occupancy measure (set))**.** Fix any policy class Π. We define the occupancy measure $\nu^{\pi}(s, a; s_0)$ of a state-action pair $(s, a) \in \mathcal{S} \cup \{s_{\mathsf{abs}}\} \times \mathcal{A}$ given a policy $\pi \in \Pi$ and a start state $s_0 \in \mathcal{S}$ to be

$$\nu^{\pi}(s, a; s_0) = \sum_{t=0}^{\infty} \gamma^t \mathbb{P}^{\pi}(S_t = s, A_t = a | S_0 = s_0).$$

Furthermore, we define the occupancy measure set induced by Π to be

$$\mathcal{M}^{\nu}(\Pi) = \{\{\nu^{\pi}(\cdot, \cdot; s_0) | s_0 \in \mathcal{S}\} | \pi \in \Pi\}.$$

Note that we have $\sum_{a \in \mathcal{A}} \nu^{\pi}(s_{\mathsf{abs}}, a; \bar{s}_0) > 0$ if and only if there is a state $s \in \mathcal{S}$ such that $\pi(s)$ does not halt. The occupancy measure only considers the discounted expected probability of reaching an environment state at decision step t following π—it ignores the wall time as it has no impact on which environment states are transitioned unless π does not halt for some states.

The required oracles are outlined in Assumptions 2 and 3 respectively, and we provide the pseudocode in Algorithm 3:

Assumption 2 (Coverage oracle)**.** We have access to an oracle Coverage where for any $C \in \mathbb{N}^{+}$, and any policy class Π, outputs True if $\mathcal{M}^{\nu}(\Pi) = \mathcal{M}^{\nu}(\Pi(C))$, and outputs False otherwise.

Assumption 3 (Runtime oracle)**.** We have access to an oracle Runtime where for any $C \in \mathbb{N}^{+}$, and any policy class Π, outputs True if no $\pi \in \Pi \backslash \Pi(C)$ is such that $\forall \pi' \in \Pi(C)$ with $\nu^{\pi}(s_0) = \nu^{\pi'}(s_0)$, we have that $\nu^{\pi}(s; s_0) > 0 \implies c^{\pi}(s) < c^{\pi'}(s)$ for some $s \in \mathcal{S}$, and outputs False otherwise.

We first provide some helpful lemmas and provide the correctness proof. First, we have that $\mathcal{M}^{\nu}(\Pi(C)) \subset \mathcal{M}^{\nu}(\Pi(C'))$ for $C' > C > 0$. We now show that at some finite runtime limit $C \in \mathbb{N}$, the occupancy measure set no longer changes.

Lemma 31. There exists a $C^* \in \mathbb{N}$ such that $\mathcal{M}^{\nu}(\Pi(C^*)) = \mathcal{M}^{\nu}(\Pi)$.

Proof. By Lemma 28, we have $\Pi(C) \subseteq \Pi(C')$ whenever $C \leq C'$, so $\mathcal{M}^{\nu}(\Pi(C)) \subseteq \mathcal{M}^{\nu}(\Pi(C'))$ for all $C \leq C'$. Consequently the sequence of occupancy-measure sets is monotonically non-

Algorithm 3 Runtime-aware PCVI

Require: $\mathcal{S}$, $\mathcal{A}$, P, R, γ, Π, initial state s_0.

$\texttt{Q}[s, a, \infty] \leftarrow$ initialize to mapping $\mathcal{A}^{\mathcal{S}} \to 0$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$
$\texttt{ConQ}[s, a, \infty] \leftarrow$ initialize to mapping $[s \mapsto (a, \infty)] \to 0$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$
for $C := 1, 2, \ldots$ **do**
  $\texttt{Q}[s, a, C] \leftarrow$ initialize to mapping $\mathcal{A}^{\mathcal{S}} \to 0$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$
  $\texttt{ConQ}[s, a, C] \leftarrow$ initialize to mapping $[s \mapsto (a, C)] \to 0$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$
  Update $\texttt{ConQ}[s]$ for all s.
  repeat
    for $(s, a, c) \in \mathcal{S} \times \mathcal{A} \times (\{\infty\} \cup [C]_+)$ **do**
      $\texttt{Q}[s, a, c] \leftarrow \gamma^{c-1} R(s, a) + \gamma^c \oplus_{s' \in \mathcal{S}} P(s'|s, a)\texttt{ConQ}[s']$
      $\texttt{ConQ}[s, a, c](Z) \leftarrow \texttt{Q}[s, a, c](X)$ for all X s.t. $Z = X \cap [s \mapsto (a, c)] \neq \emptyset$
      Update $\texttt{ConQ}[s]$
    end for
  until $\texttt{Q}$ converges
  if $\mathsf{Coverage}(C, \Pi)$ and $\mathsf{Runtime}(C, \Pi)$ **then**
    $X^* = \arg\max_X \texttt{ConQ}[s_0](X)$
    $q^* = \texttt{ConQ}[s_0](X^*)$
    $\pi^* = \mathsf{Witness}(X^*)$
    return π^*
  end if
end for

decreasing. Notably, for sufficiently large C' we have that $\Pi(C')$ is a strict superset of $\Pi(C)$—this strict increase can only occur finite many times. First, observe that the RAMDP has finite environment state and action spaces. Furthermore we consider only deterministic policies and the occupancy measure of a policy only considers the discounted probability of reaching an environment state across the number of decisions (not runtime). Consequently there are at most $|\mathcal{A}|^{|\mathcal{S}|}$ distinct occupancy measures.

We now show that, for any $\nu \in \mathcal{M}^\nu(\Pi)$, there is a constant $C^\nu \in \mathbb{N}^+$ such that $\nu \in \mathcal{M}^\nu(\Pi(C^\nu))$, then by the fact that $|\mathcal{M}^\nu(\Pi)| \leq |\mathcal{A}|^{|\mathcal{S}|}$ we must have some $C^* = \max_{\nu \in \mathcal{M}^\nu(\Pi)} C^\nu < \infty$ that covers $\mathcal{M}^\nu(\Pi)$. Fix any $\nu \in \mathcal{M}^\nu(\Pi)$ and consider the set of policies $\Pi(\nu) = \{\pi \in \Pi | \nu^\pi = \nu\}$. Consider any states $s \in \mathcal{S}$ for which any $\pi \in \Pi(\nu)$ has finite runtime. Any other $\pi' \in \Pi(\nu)$ must also have finite runtime for the same state s, as otherwise $\nu^\pi \neq \nu^{\pi'}$. Secondly, since the runtime is countable and is lower bounded by 1 by definition of RAMDP, the well-ordering principle states that there exists a state-dependent minimum runtime $C^\nu(s) = \min_{\pi \in \Pi(\nu)} C^\pi(s)$. We can then obtain the maximum runtime $C^\nu = \max_{s \in \mathcal{S}} C^\nu(s)$ across any state $s \in \mathcal{S}$ as the (environment) state space $\mathcal{S}$ is finite. By construction the set $\Pi(C^\nu)$ contains all the policies with occupancy measure, i.e., we have that $\nu\ \mathcal{M}^\nu(\Pi(C^\nu)) \subseteq \mathcal{M}^\Pi(C^\nu)$. Finally, take the maximum runtime $C^* = \max_{\nu \in \mathcal{M}^\nu(\Pi)} C^\nu$ across the finite set of occupancy measures. Any policy with occupancy measure $\nu \in \mathcal{M}^\nu(\Pi)$ is covered by $\Pi(C^\nu)$ and by Lemma 28 we have that $\Pi(C^\nu) \subseteq \Pi(C^*)$ as $C^\nu \leq C^*$. Thus we obtain the runtime $C^* \in \mathbb{N}^+$ for which $\mathcal{M}^\nu(\Pi(C^*)) = \mathcal{M}^\nu(\Pi)$. □

While the occupancy measure set no longer changes, we still have to ensure that the policy with the maximal return in $\Pi(C^*)$ is optimal—recall that RAMDP has a preference over shorter runtime.

Lemma 32. There exists a $C^* \in \mathbb{N}$ such that no $\pi \in \Pi \setminus \Pi(C^*)$ satisfies the following condition: $\forall \pi' \in \Pi(C^*)$ with $\nu^\pi(s_0) = \nu^{\pi'}(s_0)$, we have that $\nu^\pi(s; s_0) > 0 \implies c^\pi(s) < c^{\pi'}(s)$ for some $s \in \mathcal{S}$.

Proof. We can first fix an occupancy measure ν. Once we establish that there is a finite $C \in \mathbb{N}$ for ν such that no policy satisfies the condition, we can take the maximum C across all $\mathcal{M}^\nu(\Pi)$, where $\mathcal{M}^\nu(\Pi)$ is a finite set, obtaining a finite C^*.

Suppose by contradiction that there is no such a C. For this to be true there must be an infinite sequence of $\{C_i\}_{i \geq 0}$ where $C_i < C_{i+1}$ such that a $\pi \in \Pi \setminus \Pi(C_i)$ satisfies the condition. By

Lemma 28 we know that $\Pi(C)$ is non-decreasing as C increases, and furthermore there is a $C' > C$ such that $\Pi(C) \subsetneq \Pi(C')$ if and only if $\Pi(C) \neq \Pi$. Equivalently this means $\Pi \setminus \Pi(C') \subsetneq \Pi \setminus \Pi(C)$ if and only if $\Pi(C) \neq \Pi$. Let C_i be the runtime where the i'th strict containment occurs. If $\Pi(C_i) = \Pi$ at some point, then there is a contradiction due to the fact that $\Pi \setminus \Pi(C_i) = \emptyset$ thus no policy is in this set.

We now consider the more complicated scenario where $\Pi(C_i) \subsetneq \Pi(C_{i+1})$ for all $i \geq 0$. Consider any $\pi \in \Pi(C_{i+1}) \cap \Pi(C_i)$, either $c^\pi(s)$ is smaller than any other policy in $\Pi(C)$ for some $s \in \mathcal{S}$ with positive discounted probability or $c^\pi(s)$ is larger than all policies in $\Pi(C)$. If the former is always true, then we are done since $c^\pi(s) \in \mathbb{N}^+$, at some point no other policy can decrease the runtime due to the well-ordering principle. Note that this decrease can only happen finitely many times as the state space is finite and the runtime is countably infinite with a minimum element.

We now show that the latter case only holds true finitely many times before a decrease happens, which contradicts that there is an infinite sequence of $\{C_i\}_{i\geq 0}$ satisfying the condition. Consider any $\pi \in \Pi \setminus \Pi(C_{i+1})$ satisfying the condition. For all the positive discounted probability states $s \in \mathcal{S}$, we have $c^\pi(s) < \infty$ by definition. Let $C^\pi = \max_{s\in\mathcal{S}:\nu(s;s_0)>0} c^\pi(s)$ be the maximum runtime of all such states, then at iteration C^π we must have that $\pi \in \Pi(C^\pi)$. By the definition of π we must have decreased runtime for at least one positive discounted probability state. □

With these properties we can now show the correctness of Algorithm 3.

Theorem 33. Algorithm 3 halts and outputs an optimal policy.

Proof. We first show that the algorithm terminates. By Theorem 1 of Lu et al. (2018) the inner loop halts as the algorithm runs the original PCVI with a finite state-and-action spaces. Our only terminating condition comes from line 14. By Lemmas 31 and 32, there exists a finite $C \in \mathbb{N}^+$ such that both conditions are satisfied (this can be done by taking the max of the two constants in the Lemmas), hence the algorithm halts.

We now show that the algorithm returns the optimal policy. For finite state-and-action space, we know that by Theorem 1 of Lu et al. (2018) lines 4–13 obtains the Q-function consistent with policies within $\Pi(C)$. It remains to show if both conditions on line 14 are true, the algorithm must have obtained an optimal policy—the goal is to show that q^* is the optimal value function. Let $C^* \in \mathbb{N}^+$ be the iteration where both conditions are satisfied on line 14. By Lemma 32, for every $\pi \in \Pi \setminus \Pi(C^*)$, there is a $\pi' \in \Pi(C^*)$ with $\nu^\pi(s_0) = \nu^{\pi'}(s_0)$ such that $c^\pi(s) \geq c^{\pi'}(s)$ for all positive discounted probability states $s \in \mathcal{S}$. Consequently by Proposition 2 we must have $V^\pi(s_0) \leq V^{\pi'}(s_0)$. Taking the max over the policies with the same occupancy measure in $\Pi(C)$, which exists due to finiteness of state-and-action spaces and finite runtime, we obtain $V^\nu(s_0) = \max_{\pi''\in\Pi(C^*):\nu^{\pi''}(s_0)=\nu(s_0)} V^{\pi''}(s_0) \geq V^{\pi'}(s_0)$. Finally, by Lemma 31 all occupancy measures $\mathcal{M}^\nu(\Pi)$ are considered. Defining $V(\Pi(C^*)) = \{V^\nu(s_0)|\nu \in \mathcal{M}^\nu(\Pi(C^*))\}$ to be the set of maximal value functions of policies with the same occupancy measure. It must be the case that $\max_{V\in V(\Pi(C^*))} V(s_0)$ is optimal. □

We leave the possible relaxation of the oracles to future work—the key takeaway here is that even with access to the transition function and reward function, the current dynamic-programming algorithms cannot be implemented without some sort of uncomputable oracles.

F Experiment details

Our implementation is based on Stoix (Toledo, 2024). For lightsout we use the implementation by Ghugare et al. (2026); for sliding puzzle and Sokoban we use the implementation by Bonnet et al. (2024). For the analysis, we aggregate the statistics across the last 3 epochs for training, and sample 100 rollouts for evaluation. Depending on the environment, the number of samples used for computing the training statistics differ. We outline the hyperparameters for the simplistic experiment in Table 1 and the larger-scale experiments in Table 2. We now describe each environment.

Lightsout. Lightsout is a puzzle where the goal is to set a $m \times n$ grid match a target $m \times n$ grid. Each state is represented by two binary matrices $s \in \{0,1\}^{m \times n} \times \{0,1\}^{m \times n}$, where the first binary matrix corresponds to the current grid state and the second binary matrix corresponds to the target grid. The action space corresponds to one of $m \times n$ cells: Upon taking the action that corresponds to cell (i, j), the environment flips the bit of cell (i, j) and its adjacent neighbours. The agent receives a $+1$ reward when the current grid matches the target grid, and receives a $+0$ reward otherwise. For the IRU experiments we set $m = 3, n = 3$ and for the transformer experiments we set $m = 5, n = 4$. During training, the agent is only given problems where the target grid is up to $\lceil m \times n/2 \rceil$ actions away from the initial grid. During evaluation, the agent is given any target grid that is reachable by the initial grid. We truncate the rollout after $m \times n$ steps as it is the maximum number required to reach from one grid to another.

Sliding puzzle. The sliding (tile) puzzle is a puzzle that requires the agent to slide pieces to establish a fixed target configuration. For a $n \times n$ puzzle, the state space is a matrix $s \in \{0, \ldots, n^2 - 1\}^{n \times n}$. Here, the entry with 0 corresponds to an empty tile. The action space corresponds to moving the empty tile in one of the four cardinal directions. The agent receives a $+1$ reward when the puzzle is solved, and receives a $+0$ reward otherwise. There are 181,440 reachable states from the target state, and the longest shortest path between them is 31. In the experiments we set $n = 3$ and truncate the rollout after 40 steps during training and 80 steps during evaluation.

Sokoban. Sokoban is a box-pushing environment that requires the agent to place all boxes onto their targets. The state is represented by a binary tensor $s \in \{0,1\}^{10 \times 10 \times 4}$, where each channel respectively corresponds to the boxes, the agent, the walls, and the targets. The action space is the four cardinal directions. Here, we use the default dense reward formulation, where the agent receives a $+1$ reward when a box is placed onto a target, receives a $+10$ reward for solving the level, and receives a -0.1 reward otherwise. During training, the agents only observe instances from the unfiltered-training set and during evaluation the agents observe instances from the unfiltered-validation set. The rollout truncates after 120 steps if the level is not solved before then.

Table 1: Hyperparameters used by the unshared IRU on 3×3 lightsout. Uniform-compute runs use budgets $c \in \{1, 2, 4, 8, 16\}$; the variable-compute run uses $c \in [1, 16]$.

Category	Hyperparameter	Lightsout
Environment	Grid size	3×3
	Episode length (train)	5
	Episode length (eval)	9
	Difficulty threshold	0.5
Actor	Observation encoder	Flatten
	Architecture	IRU without parameter sharing
	Hidden dim	16
	Layers per step	1
	Input layer norm	True
Critic	Architecture	MLP
	Layer sizes	[256, 256]
	Activation	ReLU
Optimization (PPO)	Actor learning rate	3×10^{-4}
	Critic learning rate	3×10^{-4}
	Actor weight decay	1×10^{-4}
	Critic weight decay	0
	Epochs per update	4
	Num. minibatches	8
	Rollout length	32
	Discount γ	0.99
	GAE λ	0.95
	PPO clip ϵ	0.2
	Clip value loss	False
	Value loss	Expectile
	Expectile α	0.5
	Value loss coefficient	0.5
	Entropy coefficient	0.01
	Max. grad. norm	0.5
	Standardize advantages	True
Training budget	Total env. steps	1×10^8
	Parallel envs	1024
	Num. updates	3051
	Updates per eval	61
	Num. seeds	10
Adaptive mechanism	Max. thinking steps	16
	Halting temperature	1.0
	Halting entropy coefficient	0.01
	Halting learning rate	3×10^{-4}
	Halting weight decay	1×10^{-4}
	Stop-gradient halting input	False

G ADDITIONAL ANALYSIS

Compute time curve for implicit CoT. We include the compute-time curves over the training in lightsout, sliding puzzle, and Sokoban in Figure 10.

Transformer with explicit CoT. We include experiments where we run transformer with explicit CoT. For explicit CoT, we vary between vocabulary sizes to be $|\mathcal{V}| = \{1, 2, 4, 8, 16\}$. Figure 11 illustrates that $c = 1$ is generally the best fixed-compute policy across different vocabulary sizes. When $|\mathcal{V}| = 1$, the benefit of the computational space is limited and the variable-compute policy is unable to match $c = 1$. However, with $|\mathcal{V}| \in \{2, 4\}$ we observe that the variable-compute policy can match and even outperform $c = 1$. While increasing $|\mathcal{V}|$ might increase the expressivity of the policy, it also becomes more difficult to train the variable-compute policy. Not only does the performance decrease when $|\mathcal{V}| \in \{8, 16\}$, we also see that the CI becomes wider compared to that

Table 2: Hyperparameters used by transformer with iCoT. Values spanning all three environment columns are shared across environments.

Category	Hyperparameter	Lightsout	Sliding puzzle	Sokoban
Environment	Grid size	5×4	3×3	10×10
	Episode length (train)	10	40	120
	Episode length (eval)	20	80	120
	Difficulty threshold	0.5	–	–
	Scramble moves	–	200	–
	Levels (train / eval)	–	–	Sokoban unfiltered (train / valid)
Actor	Observation encoder	Flatten	CNN	CNN
	CNN channels	–	[8, 8]	[64]
	CNN kernel sizes	–	[2, 1]	[3]
	CNN strides	–	[1]	[2]
	CNN output MLP	–	[64]	[128]
	Hidden dim	128	16	128
	MLP dim	256	256	512
	QKV dim	256	256	512
	Num. heads	8	16	16
	Num. layers		2	
	Activation	GELU	ReLU	ReLU
Critic	Architecture	MLP	CNN + MLP	CNN + MLP
	CNN output MLP	–	[64]	[128]
	Layer sizes	[256, 256]	[64, 64]	[128, 128]
	Activation		ReLU	
Optimization (PPO)	Actor learning rate		3×10^{-4}	
	Critic learning rate		3×10^{-4}	
	Actor weight decay	0.1	0.02	0.01
	Epochs per update	4	4	2
	Num. minibatches	8	8	4
	Rollout length	32	32	20
	Discount γ		0.999	
	GAE λ		0.95	
	PPO clip ϵ		0.2	
	Clip value loss	False	True	False
	Value loss		Expectile	
	Expectile α	0.1	0.2	0.9
	Value loss coefficient		0.5	
	Entropy coefficient		0.01	
	Max. grad. norm	0.5	0.5	5.0
	Standardize advantages		True	
Training budget	Total environment steps	3×10^8	1×10^8	3×10^9
	Parallel envs	1024	256	128
	Num. updates	9155	12207	1171875
	Updates per eval	183	244	23437
	Num. seeds	10	10	5
Halting mechanism	Max. compute steps		5	
	Halting temperature	5.0	1.0	1.0
	Halting entropy coefficient	0.01	0.001	0.01
	Halting learning rate	3×10^{-4}	3×10^{-4}	1×10^{-3}
	Halting weight decay		0	
	Clip halting-head gradients		True	

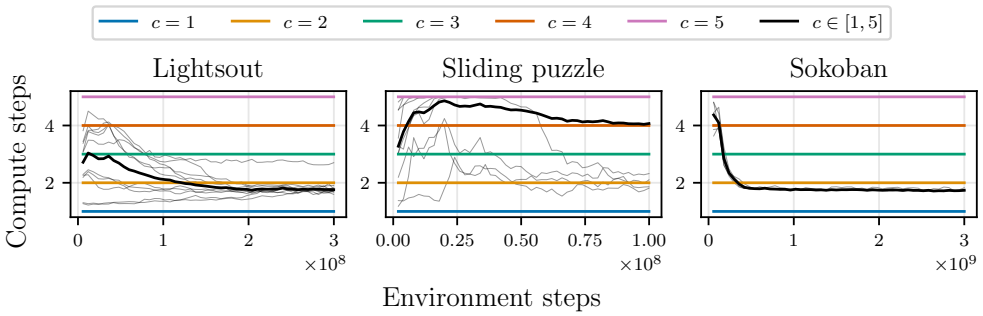


Figure 10: The compute times during training of uniform budgets and variable budget in 5×4 lightsout, 3×3 sliding puzzle, and Sokoban. The policies are transformers with implicit chain-of-thought reasoning. We plot both the IQM and the curve for each seed.

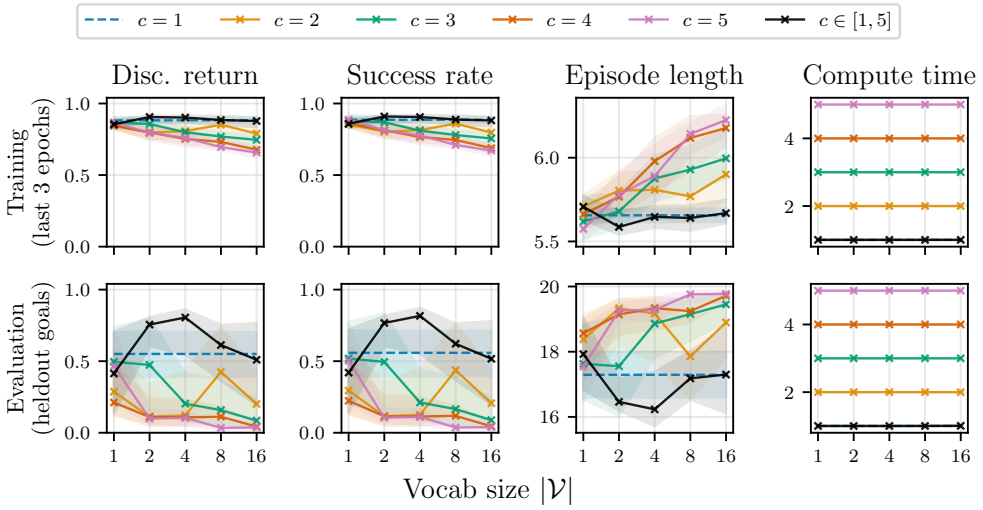


Figure 11: A sensitivity analysis over vocabulary sizes $|\mathcal{V}|$ in `lightsout-5x4` with explicit chain-of-thought. The solid lines correspond to the IQM and the shaded regions correspond to 95% stratified bootstrap CI.

of $|\mathcal{V}| \in \{2, 4\}$. When we consider fixed-compute policies with $c > 1$, we see that these policies generally perform worse with increasing $|\mathcal{V}|$, in fact the larger the c the greater the degradation. We hypothesize that the fixed-compute policies are forced to explore $|\mathcal{V}|^c$ possible computational paths before "identifying" the best one—the variance becomes larger with increasing c and $|\mathcal{V}|$, thereby requiring significantly more samples to converge.

Varying discount factor. Here we vary the discount factor and analyze how it impacts the variable-compute policy. Figure 12 shows that no matter how we vary $\gamma \in \{0.99, 0.995, 0.999, 0.9995\}$, the variable-compute policy achieves the best performance or is comparable to the best fixed-compute policies. Perhaps not too surprisingly, increasing γ corresponds to penalizing extra computation less. Hence, we can see that the variable-compute policy spends more compute time as we increase γ, a sign that the agent has learned to effectively trade off performance based on how much the environment penalizes the agent for computation.

Visualizing hidden states. In addition to visualizing the hidden state by the transformer in the sliding puzzle environment, we provide the visualization for the transformer in the lightsout environment in Figure 13. As we had mentioned in §5, it appears that the fixed-compute policy with $c = 2$ is already near optimal, thus increasing computation budget is unnecessary to gain more performance during training. Different than the result for sliding puzzle, our visualization on the

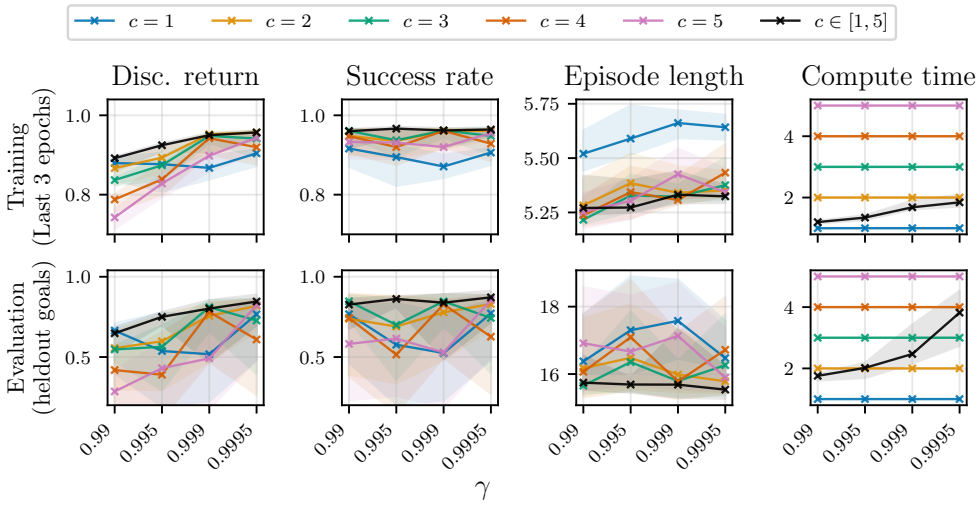


Figure 12: The impact of changing γ across uniform budgets and variable budget in `lightsout-5x4`. The solid lines correspond to the IQM and the shaded regions correspond to 95% stratified bootstrap CI.

lightsout evaluation instances suggests that the transformer generally outputs a hidden state where the halt predictor continues at the first step but halts at the second step.

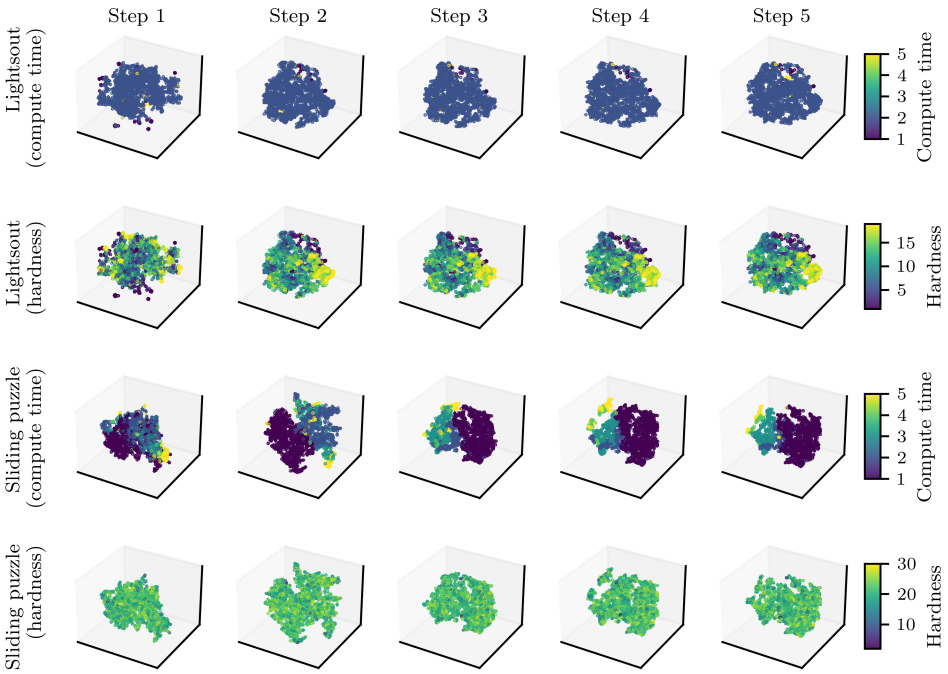


Figure 13: A t-SNE visualization of the last hidden state generated by the trained variable-compute policy. The points in rows 1 and 3 are coloured based on the compute steps expended to compute an action given the state, and those in rows 2 and 4 are coloured based on the shortest-path length to the solution.