Appendix

A Proofs and extensions

A.1 Lemma 3.1, steady-state convergence

For any environment state s, consider a reasoning Markov chain (RMC) defined on a compact action space A with transition probabilities given by $\pi^b(a'|a,s)$. Suppose that $\inf\{\pi^b(a'|a,s) : a', a \in A\} > 0$. Then there exists a steady-state probability distribution function $\pi^s(\cdot|s)$ such that:

$$\lim_{n\to\infty} \pi^b_n(a|a_0,s) \to \pi^s(a|s) \quad \textit{for all } a \in A. \tag{17}$$

Proof. Let $\delta = \inf\{\pi^b(a'|a,s) : a', a \in A\} > 0$ and let m be the *Lebesgue measure*. Since A is compact we have that $m(A) = |A|$ is finite. Thus, let v be the uniform probability measure on A, we have that:

$$\pi^b(a'|a) \geq \delta \times |A| \times v(a) \quad \text{for all } a', a \in A.$$

Hence we have that the resulting RMC is ergodic, irreducible, and aperiodic. Furthermore, this shows that the *entire action space* is a *small set* [87, 88] of the reasoning Markov chain. This property directly implies the existence of a unique π^s and quantifies an exponential bound on the rate of convergence [44, 89]:

$$||\pi^b_n(a|a_0,s) - \pi^s(a|s)||_{TV} \leq (1 - \delta \times |A|)^n, \quad \text{for all } a, a_0 \in A, \tag{18}$$

where $||\pi^b_n(a|a_0,s) - \pi^s(a|s)||_{TV}$ represents the *total variation distance* between the n-step transition probabilities of the RMC and the SS-policy. We refer to Nummelin [90] for a full formal derivation. Lemma 3.1 clearly follows from the above results.

□

A.2 Theorem 3.2, steady-state policy gradient

Let $\pi^b_\theta(\cdot|a,s)$ be a parameterized belief transition policy which defines a reasoning Markov chain with a stationary distribution given by the steady-state policy $\pi^s_\theta(\cdot|s)$. Let Q^s be a real function defined on $S \times A$, with a family of n-step extensions $\{Q^s_n\}$ as defined in Eq. 8. Suppose π^b, Q^s and their gradient with respect to the parameters θ (denoted ∇_θ) are continuous and bounded functions.

Then:

$$\nabla_\theta \, \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[Q^s(s,a)\right] = \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[\lim_{N\to\infty}\sum_{n=0}^{N} \nabla_\theta \, \mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[Q^s_n(s,a')\right]\right].$$

Proof. We start by using the invariance of the RMC's steady state distribution probabilities, $\pi^s_\theta(a|s)$, when performing a reasoning step with transition probabilities from the BT-policy $\pi^b_\theta(a'|a,s)$. Hence, we can decompose $\nabla_\theta \, \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[Q^s(s,a)\right]$ in the sum of two distinct terms by applying the product rule:

$$\begin{aligned}
&\nabla_\theta \, \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[Q^s(s,a)\right] = \nabla_\theta \, \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[\mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[Q^s(s,a')\right]\right] \\
&\quad = \nabla_\theta \int_A \pi^s_\theta(a|s) \int_A \pi^b_\theta(a'|a,s) Q^s(s,a') da' da \\
&\quad = \nabla_\theta \int_A \int_A \pi^s_\theta(a|s) \pi^b_\theta(a'|a,s) Q^s(s,a') da' da \\
&\quad = \int_A \pi^s_\theta(a|s) \nabla_\theta \int_A \pi^b_\theta(a'|a,s) Q^s(s,a') da' da + \int_A (\nabla_\theta \pi^s_\theta(a|s)) \int_A \pi^b_\theta(a'|a,s) Q^s(s,a') \\
&\hspace{25em} \text{Leibniz integral rule} \\
&\quad = \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[\nabla_\theta \, \mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[Q^s(s,a')\right]\right] + \int_A (\nabla_\theta \pi^s_\theta(a|s)) \int_A \pi^b_\theta(a'|a,s) Q^s(s,a') da' da. \\
&\hspace{30em} (1) + (2)
\end{aligned}$$

From the definition of the n-step extensions being a *local* approximation of the RMC with no dependence form θ (since $\nabla_\theta Q^s_n(s,a)=\mathbf{0}$), we can rewrite term (2) as:

$$\begin{aligned}\int_A (\nabla_\theta \pi^s_\theta(a|s)) \int_A \pi^b_\theta(a'|a,s) Q^s(s,a') da' da. &= \nabla_\theta \int_A \pi^s_\theta(a|s) Q^s_1(s,a) da \\ &= \nabla_\theta \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[Q^s_1(s,a)\right].\end{aligned}$$

Hence, from the relationship between $\nabla_\theta \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[Q^s_0(s,a)\right]$ (i.e., $\nabla_\theta \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[Q^s(s,a)\right]$) with (1) and (2) we see that:

$$\nabla_\theta \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[Q^s_0(s,a')\right] = \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[\nabla_\theta \mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[Q^s_0(s,a')\right]\right] + \nabla_\theta \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[Q^s_1(s,a)\right]. \tag*{(1) + (2)}$$

Since each n-step extension Q^s_n has the same exact recursive relationship with the subsequent Q^s_{n+1} n-step extension, more generally, we have that:

$$\nabla_\theta \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[Q^s_n(s,a')\right] = \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[\nabla_\theta \mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[Q^s_n(s,a')\right]\right] + \nabla_\theta \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[Q^s_{n+1}(s,a)\right]. \tag{19}$$

Thus, we can apply Equation 19 recursively to (2) and all resulting $\nabla_\theta \mathbb{E}_{a\sim\pi^s_\theta}\left[Q^s_n(s,a')\right]$ terms:

$$\begin{aligned}\nabla_\theta \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[Q^s_0(s,a')\right] &= \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[\nabla_\theta \mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[Q^s_0(s,a')\right]\right] + \nabla_\theta \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[Q^s_1(s,a)\right] \\ &= \mathbb{E}_{a\sim\pi^s_\theta}\left[\nabla_\theta \mathbb{E}_{a'\sim\pi^b_\theta}\left[Q^s_0(s,a')\right]\right] + \mathbb{E}_{a\sim\pi^s_\theta}\left[\nabla_\theta \mathbb{E}_{a'\sim\pi^b_\theta}\left[Q^s_1(s,a')\right]\right] + \nabla_\theta \mathbb{E}_{a\sim\pi^s_\theta}\left[Q^s_2(s,a)\right] \\ &= \ldots \\ &= \lim_{N\to\infty}\sum_{n=0}^{N} \mathbb{E}_{a\sim\pi^s_\theta}\left[\nabla_\theta \mathbb{E}_{a'\sim\pi^b_\theta}\left[Q^s_n(s,a')\right]\right] + \nabla_\theta \mathbb{E}_{a\sim\pi^s}\left[Q^s_N(s,a)\right]. \qquad (20)\end{aligned}$$

Moreover, from the definitions of the n-step extensions Q^s_n and the SS-policy π^s_θ, we see that:

$$\begin{aligned}\lim_{N\to\infty} \nabla_\theta \mathbb{E}_{a\sim\pi^s_\theta}\left[Q^s_N(s,a)\right] &= \lim_{N\to\infty}\int_A (\nabla_\theta \pi^s_\theta(a|s)) Q^s_N(s,a) da && \text{Leibniz integral rule} \\ &= \int_A (\nabla_\theta \pi^s_\theta(a|s)) \lim_{N\to\infty}\int_A \pi^b_N(a'|a,s) Q^s(s,a') da' da \\ &= \int_A (\nabla_\theta \pi^s_\theta(a|s)) \int_A \pi^s_\theta(a'|s) Q^s(s,a') da' da \\ &= \nabla_\theta \int_A \pi^s_\theta(a|s) da \times \int_A \pi^s_\theta(a'|s) Q^s(s,a') da' && \text{Leibniz integral rule} \\ &= \mathbf{0} && (21)\end{aligned}$$

Therefore, we can use the identity from 21 to simplify Equation 20, and we are left with:

$$\begin{aligned}\nabla_\theta \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[\mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[Q^s(s,a')\right]\right] &= \lim_{N\to\infty}\sum_{n=0}^{n} \mathbb{E}_{a\sim\pi^s_\theta}\left[\nabla_\theta \mathbb{E}_{a'\sim\pi^b_\theta}\left[Q^s_n(s,a')\right]\right]. \\ &= \mathbb{E}_{a\sim\pi^s_\theta}\left[\lim_{N\to\infty}\sum_{n=0}^{N} \nabla_\theta \mathbb{E}_{a'\sim\pi^b_\theta}\left[Q^s_n(s,a')\right]\right].\end{aligned}$$

□

Convergence and finite approximations. We can use the results derived in Lemma 3.1 (see A.1) to establish some of the properties of the infinite series introduced in Theorem 3.2. To simplify notation, we will denote each term in the series from Equation 9 with $g_n(s,a)$, i.e.:

$$g_n(s,a) = \nabla_\theta \mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[Q^s_n(s,a')\right], \quad \text{such that } \nabla_\theta \mathbb{E}_{a\sim\pi^s_\theta}\left[Q^s(s,a)\right] = \mathbb{E}_{a\sim\pi^s_\theta}\left[\lim_{N\to\infty}\sum_{n=0}^{N} g_n(s,a)\right].$$

We rewrite the value of each $g_n(s,a)$ term with the *score function*, using the identity $\nabla_\theta \log(\pi^b_\theta(a'|a,s)) = \frac{\nabla_\theta \pi^b_\theta(a'|a,s)}{\pi^b_\theta(a'|a,s)}$:

$$\begin{aligned} g_n(s,a) &= \nabla_\theta \, \mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[Q^s_n(s,a')\right] \\ &= \int \nabla_\theta \pi^b_\theta(\cdot|a,s) Q^s_n(s,a')da' && \text{Leibniz integral rule} \\ &= \int \pi^b_\theta(\cdot|a,s)\nabla_\theta \log(\pi^b_\theta(a'|a,s))Q^s_n(s,a')da' \\ &= \mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[\nabla_\theta \log(\pi^b_\theta(a'|a,s))Q^s_n(s,a')\right]. \end{aligned}$$

Then, we explicitly subtract a baseline, $\mathbb{E}_{a''\sim\pi^s_\theta}[Q^s(s,a'')]$, from the n-step Q-function value $Q^s_n(s,a')$ multiplying the score. Note, that $\mathbb{E}_{a''\sim\pi^s_\theta}[Q^s(s,a'')]$ corresponds to the *value function* of the SS-policy, which is a baseline *independent* of a'. Therefore, $\mathbb{E}_{a'\sim\pi^b_\theta}\left[\nabla_\theta \log(\pi^b_\theta(a'|a,s))\,\mathbb{E}_{a''\sim\pi^s_\theta}[Q^s(s,a'')]\right] = 0$, showing that this subtraction leaves all the gradient terms unchanged [40]:

$$\begin{aligned} g_n(s,a) &= \mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[\nabla_\theta \log(\pi^b_\theta(a'|a,s))Q^s_n(s,a')\right] \\ &= \mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[\nabla_\theta \log(\pi^b_\theta(a'|a,s))\left(Q^s_n(s,a') - E_{a''\sim\pi^s_\theta(\cdot|s)}\left[Q^s\left(s,a''\right)\right]\right)\right] \\ &= \mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[\nabla_\theta \log(\pi^b_\theta(a'|a,s))\left(\int \pi^b_n(a''|a',s)Q^s(s,a'')da'' - \int \pi^s_\theta(a''|s)Q^s(s,a'')da''\right)\right] \\ &= \mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[\nabla_\theta \log(\pi^b_\theta(a'|a,s))\int\left(\pi^b_n(a''|a',s) - \pi^s_\theta(a''|s)\right)Q^s(s,a'')da''\right] \end{aligned} \tag{22}$$

From our boundedness assumptions, we define existing positive real values Q^+, S^+ such that:

$$Q^+ \geq |Q^s(s,a)|, \quad S^+ \geq |\nabla_\theta \log(\pi^b_\theta(a'|a,s))|, \quad \text{for all } a,a' \sim A, s \sim S.$$

Thus, by noting the relationship

$$\int \left|\pi^b_n(a''|a',s) - \pi^s_\theta(a''|s)\right| da'' \leq 2 \times ||\pi^b_n(a''|a',s) - \pi^s_\theta(a''|s)||_{TV}, \tag{23}$$

we use the bound defined in Equation 18 from Lemma 3.1 to show that:

$$\begin{aligned} g_n(s,a) &= \mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[\nabla_\theta \log(\pi^b_\theta(a'|a,s))\int\left(\pi^b_n(a''|a',s) - \pi^s_\theta(a''|s)\right)Q^s(s,a'')da''\right] \\ &\leq \mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[\nabla_\theta \log(\pi^b_\theta(a'|a,s))\int\left|\pi^b_n(a''|a',s) - \pi^s_\theta(a''|s)\right|Q^+da''\right] \\ &\leq \mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[2S^+Q^+||\pi^b_n(a''|a',s) - \pi^s_\theta(a''|s)||_{TV}\right] \\ &\leq 2S^+Q^+(1-\delta\times|A|)^n. \end{aligned} \tag{24}$$

From Equation 24, it is clear that each term in the series from Theorem 3.2 will converge *exponentially fast*. This shows we can estimate $\mathbb{E}_{a\sim\pi^s_\theta}\left[\lim_{N\to\infty}\sum_{n=0}^{N} g_n(s,a)\right]$ with any arbitrarily small error using a finite N. Moreover, we can see that each gradient term in Equation 22 is based on the expected deviation of the *n-step* transition function from the steady-state distribution of the RMC. This property provides concrete intuition for the proposed adaptive truncation strategy, which considers the expected number of steps before reaching *approximate convergence* (i.e., $\pi^b_N \approx \pi^s_\theta$). A similar gradient estimator for the steady-state distribution was also derived with more formal notation in prior works [91]. Heidergott et al. [92] further extended similar convergence results to broader classes of Markov chains and even certain unbounded performance functions, providing further motivation for our practical method.

A.3 Policy gradient estimation for arbitrary regularized objectives

We can extend Theorem 3.2 to estimate the policy gradient with respect to a wider class of objectives that involve the expectation over *regularized functions*:

$$J(\theta) = \mathbb{E}_{s,a\sim\pi^s_\theta(\cdot|s)}\left[Q^r_\theta(s,a)\right], \tag{25}$$

where the values of $Q^r_\theta(s,a)$ might depend on the parameters of the BT-policy, π^b_θ. We assume Q^r_θ and its gradients respect the same regularity assumptions of continuity and boundedness stated in Theorem 3.2. The MaxEnt objective from Equation 6 is a particular instance of this setting with the value of Q^r_θ being dependent on the policy's entropy, i.e., $Q^r_\theta(s,a) = Q^s(s,a) - \alpha \log(\pi^s_\theta(a|s))$. Analogously to the unregularized case, we define a set of *n-step extensions* for Q^r_θ for all $n = 0, 1, 2, \dots$:

$$Q^r_n(s,a) = \int_A \pi^b_n(a'|a,s) Q^r_\phi(s,a') da', \quad \textit{with} \quad \nabla_\theta Q^r_n(s,a) = \mathbf{0}. \tag{26}$$

For any state s, we can then apply the product rule to rewrite the gradient of Equation 25 as a sum of two expectations:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[Q^r_\theta(s,a)\right] \\
&= \int_A (\nabla_\theta \pi^s_\theta(a|s)) Q^r_\theta(s,a') da + \int_A \pi^s_\theta(a|s) \nabla_\theta Q^r_\theta(s,a') da. && \text{Leibniz integral rule} \\
&= \nabla_\theta \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[Q^r_0(s,a)\right] + \mathbb{E}_{a\sim\pi^s_\theta(\cdot|s)}\left[\nabla_\theta Q^r_\theta(s,a)\right]. && (1) + (2)
\end{aligned}$$

Here, we used the definition of each $Q^r_n(s,a)$ term being a *local approximation* and having no dependency on the BT-policy to rewrite term (1) as an expectation. Hence, we can directly apply the unregularized version of Theorem 3.2 to term (1), yielding:

$$\nabla_\theta J(\theta) = \mathbb{E}_{a\sim\pi^s_\theta}\left[\lim_{N\to\infty} \sum_{n=0}^{N} \nabla_\theta \mathbb{E}_{a'\sim\pi^b_\theta}\left[Q^r_n(s,a')\right]\right] + \mathbb{E}_{a\sim\pi^s_\theta}\left[\nabla_\theta Q^r_\theta(s,a)\right]. \qquad (1) + (2)$$

Since $\mathbb{E}_{a\sim\pi^s_\theta}\left[\nabla_\theta Q^r_\theta(s,a)\right] = \mathbb{E}_{a\sim\pi^s_\theta}\left[\mathbb{E}_{a'\sim\pi^b_\theta(\cdot|a,s)}\left[\nabla_\theta Q^r_\theta(s,a')\right]\right]$, we can merge back the two expectations, obtaining:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{a\sim\pi^s_\theta}\left[\nabla_\theta Q^r_\theta(s,a') + \lim_{N\to\infty} \sum_{n=0}^{N} \nabla_\theta \mathbb{E}_{a'\sim\pi^b_\theta}\left[Q^r_n(s,a')\right]\right] \\
&= \mathbb{E}_{a\sim\pi^s_\theta}\left[\nabla_\theta \mathbb{E}_{a'\sim\pi^b_\theta}\left[Q^r_\theta(s,a')\right] + \lim_{N\to\infty} \sum_{n=1}^{N} \nabla_\theta \mathbb{E}_{a'\sim\pi^b_\theta}\left[Q^r_n(s,a')\right]\right].
\end{aligned} \tag{27}$$

The resulting form of the policy gradient generalizes the original Equation 9 from Theorem 3.2, where now the *first term* in the infinite series explicitly considers the full gradients from the regularized objective. Hence, applying truncation and the reparameterization trick (as in Equation 13) now yields:

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\epsilon_0,\dots,\epsilon_N}\left[\nabla_\theta Q^r_\theta(s,a_1) + \sum_{n=0}^{N} \nabla_{a_n} Q^r_\theta(s,a_n) \left(\prod_{i=0}^{n-1} \nabla_{a_i} f^b(a_i, s, \epsilon_{i+1})\right) \nabla_\theta f^b_\theta(a, s, \epsilon_0)\right],$$
$$\text{where } a \sim \pi^s_\theta(\cdot|s), \quad a_0 = f^b(a, s, \epsilon_0), \quad \text{and } a_i = f^b(a_{i-1}, s, \epsilon_i) \quad \text{for all } i = 1, \dots, n. \tag{28}$$

Practical considerations for MaxEnt RL. The reparameterized extension in Equation 28 is directly analogous to the extension of DDPG policy gradients [12] in SAC [13, 39] when considering the MaxEnt regularized objective:

$$Q^r_\theta(s,a) = Q^s(s,a) - \alpha \log \pi^s_\theta(a|s), \quad \nabla_\theta Q^r_\theta(s,a) = -\nabla_\theta \alpha \log \pi^s_\theta(a|s). \tag{29}$$

To estimate the entropy of the SS-policy, we use the approximation $\pi^s_\theta(a|s) = \mathbb{E}_{\pi^s_\theta(a'|s)}\left[\pi^b_\theta(a|s)\right] \approx \frac{1}{N+1}\sum_{n=0}^{N} \pi^b_\theta(a|a_n, s)$, which results in a *nested Monte-carlo estimator* [93] for evaluating this component of the policy gradient with reparameterization. We find that using $N = \lceil \hat{N} \rceil$ (Section 3.4) ensures enough samples for practical effectiveness, without necessitating additional debiasing tricks (e.g., [94]).

A.4 Unlimited expressivity

The unlimited representation power of Serial Markov Chain Reasoning comes from the fact that the distribution of agent behavior given by the SS-policy, π^s_θ, is a mixture model with potentially infinitely many components. Hence, even a simple Gaussian parameterization of the BT-policy π_θ^b makes such distribution an arbitrary mixture of Gaussian distributions, which enables agent behavior to approximate any action distribution to arbitrary precision [53]. This is due to Gaussian mixtures being universal approximators of densities [95]. Recent works also show that the same property extends to arbitrary mixture models, with implications for future non-Gaussian extensions of our framework [96].

B Convergence detection

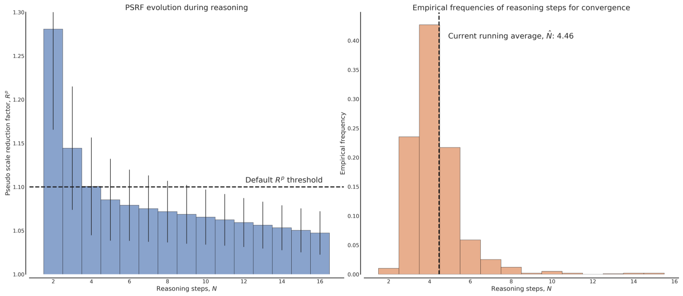


Figure 5: Pseudo scale reduction factor statistics collected after training an agent in the Ant-v2 environment from OpenAI Gym Mujoco with SSPG. (**Left**) Evolution of the average PSRF throughout an episode by performing up to 16 reasoning steps for each action selection decision. (**Right**)) Empirical frequencies of the reasoning steps performed before approximate convergence to the steady-state distribution of the RMC, as determined by $R^p < 1.1$.

As described in Section 3.4, we detect convergence in the reasoning process using the multivariate version of the seminal Gelman-Rubin (GR) convergence diagnostic [50, 51]. Given a set of M parallel reasoning chains of length N, $a_{1:N}$, this diagnostic computes the *pseudo scale reduction factor* (PSRF), a score that should expectedly converge to 1 as N increases and the distribution of the different chains approaches the steady-state distribution. We provide an example in Figure 5 Left where we display how the average PSRF evolves after performing 2-16 reasoning steps after training an agent for the Ant-v2 Mujoco task. As described in the main text, the PSRF (R^p) is based on the largest eigenvalue from the matrix product of the sample covariance *within* (W) each of the parallel chains ($\left[a_1^i, \dots, a_M^i\right]$, for $i = 1, 2, \dots, M$) and an unbiased estimate of the true covariance of the steady-state distribution, utilizing the sample covariance *between* (B) the different chains:

$$R^p = \sqrt{\frac{N-1}{N} + \lambda_{max}(W^{-1}B)}.$$

This score can be interpreted as the largest variability in any direction between these two estimates, which can be much higher than the ratio of individual variances computed by the univariate GR scores [52]. Intuitively, as the variance within each chain approaches the variance between independent chains, we can take this as a strong indication that we sufficiently explored the Markov chain's state-space within each chain. For our problem settings, we found the PSRF to work well with the default convergence threshold $R^p < 1.1$. After detecting convergence, we use the number of performed reasoning steps (N) to update a running mean $\hat{N} = \rho\hat{N} + (1-\rho)N$, as shown in Algorithm 1. To perform this update, we found a 'untuned' choice of $\rho = 0.99$ to also work appropriately. We observed that for any individual task and training iteration, the majority of the agent's action selection decisions requiring a number of steps close to $\hat{N}$, as shown by the example in Figure 5 Right. Hence, rather than computing the PSRF at every step we found it more efficient to compute the first score only after simulating the RMC for $N = \lfloor\hat{N}\rfloor$ reasoning steps. In particular, starting from $\mathbf{a}_{\mathbf{1}:\lfloor\hat{\mathbf{N}}\rfloor}$, we consider two cases: If $R^p \geq 1.1$, we simulate and compute the PSRF for all further steps until convergence. Otherwise, if $R^p < 1.1$, we backtrack to find the minimum N, such that the subsequence $\mathbf{a}_{\mathbf{1:N}}$ still satisfies our threshold. With this strategy, we effectively limit the number of evaluations of the PSRF and bolster overall efficiency.

Considerations. While the assumptions underlying the properties of the GR convergence diagnostic [50, 51] are hard to ensure in practical scenarios, we believe its widespread adoption and empirical effectiveness are strong motivators for its use. Recent extensions [52] incorporated modern techniques for variance/covariance estimation together with less conservative termination criteria. Such methods could provide further efficiency gains to our framework, but might also introduce alternative forms of bias while acting and learning. We leave such exploration and analysis to future work.

C Implementations details

In this Section, we provide descriptions of the SSPG implementations and experimental setups, together with comprehensive hyper-parameters lists. Since serial Markov chain reasoning is a novel framework, where action-selection is an adaptive iterative process, it adds additional implementation complexity compared to traditional reinforcement learning. Hence, we share our implementation to ensure reproducibility and facilitate future extensions/comparisons. By default, the only main requirement of implementing the *serial Markov chain reasoning* framework is to substitute traditional policies $\pi(\cdot|s)$ with BT-policies $\pi^b(\cdot|a, s)$, which take an additional input from the agent's action space to perform *reasoning*. In the considered implementations, we also store a *short-term action memory buffer*, $\hat{A}$, for acting in the environment, as described in Section 3.3. Overall, we mostly re-use hyper-parameters and practices from existing policy gradient implementations without major SSPG-specific tuning. In total, our algorithm only introduces 4 new main hyper-parameters: 1. The number of initial action beliefs, M. 2. The pseudo scale reduction factor (PSRF) threshold to detect approximate convergence. 3. The size of the short-term memory action buffer, $\hat{A}$. 4. The coefficient for updating the running mean of performed reasoning steps, ρ (i.e., to update the value of $\hat{N}$ for learning). The values for these new hyper-parameters were mostly chosen based on default values from similar practices and preliminary experimentation. Overall, SSPG's performance appeared to be quite robust to a range of reasonable choices and we kept them *fixed* for all tested domains. We provide ablation studies and examine performance with different settings in Appendix E. These results further confirm the claimed robustness.

C.1 OpenAI Gym Mujoco

Table 1: SSPG hyper-parameters on OpenAI Gym Mujoco tasks

General MaxEnt RL hyper-parameters (following common practices [60, 61]	
Replay data buffer size	1000000
Batch size	256
Minimum data before training	5000
Random exploration steps	5000
Optimizer	*Adam* [97]
Policy/critic learning rate	0.0003
Policy/critic β_1	0.9
Critic UTD ratio	10 (half the other baselines)
Policy UTD ratio	1
Discount γ	0.99
Target critic polyak coefficient	0.995
Hidden dimensionality	256
Fully-connected hidden layers	3
Nonlinearity	ReLU
Initial entropy coefficient α	1
Entropy coefficient learning rate	0.0001
Entropy coefficient β_1	0.5
Policy target entropy	*Hopper* : -1, *HalfCheetah*: -3, *Walker2d*: -3, *Ant*: -4, *Humanoid*: -2
Critic ensemble size	10
Critic penalization (uncertainty regularizer [98])	0.75
SSPG-specific hyper-parameters	
Number initial action beliefs, M	64
PSRF threshold	1.1
Short-term action memory size	64
Reasoning steps moving average coefficient, ρ	0.99

Our Mujoco SSPG implementation follows common practices employed in the considered baselines and recent literature. Following REDQ [60] and MBPO [61], we employ a critic ensemble of 10 models and use the suggested task-specific target entropy values for automatic tuning of the MaxEnt coefficient, α [39]. We parameterize each critic with a *modern* architecture [99] employing 3 fully connected hidden layers as in [98]. Inline with observations for IAPO [62], we find that performing iterative computations with the policy model can easily lead to a value overestimation phenomenon. Hence, to compute the Q-function's targets, we use the uncertainty regularizer [98] with a slightly-increased fixed coefficient of 0.75 (the default value of 0.5 is equivalent to the penalization from clipped double Q-learning [100]). See Table 1 for a full list of hyper-parameters.

C.2 DeepMind Control

Table 2: SSPG hyper-parameters on DeepMind Control tasks from pixels

Pixel-based RL hyper-parameters (same as DrQv2 [66])	
Replay data buffer size	1000000 (quadruped_run: 100000)
Batch size	256 (walker_run: 512)
Minimum data before training	4000
Random exploration steps	2000
Optimizer	*Adam* [97]
Policy/critic learning rate	0.0001
Policy/critic β_1	0.9
Critic UTD ratio	0.5
Policy UTD ratio	0.5
Discount γ	0.99
Target critic polyak coefficient	0.99
N-step	3 (walker_run: 1)
Convolutional kernel size	3×3
Convolutional filters	32
Convolutional layers	4
Convolutional strides	2, 1, 1, 1
Feature dimensionality	50
Hidden dimensionality	256
Fully-connected hidden layers	3
Nonlinearity	ReLU
General MaxEnt RL hyper-parameters	
Initial entropy coefficient α	1
Entropy coefficient learning rate	0.0001
Entropy coefficient β_1	0.5
Policy target entropy	$0.5 \rightarrow -1\times$ dim(A) in 500K steps
SSPG-specific hyper-parameters	
Number initial action beliefs, M	64
PSRF threshold	1.1
Short-term action memory size	64
Reasoning steps moving average coefficient, ρ	0.99

Our DeepMind Control pixel-based SSPG implementation mostly follows the exact hyper-parameters and model specifications from DrQv2 [66], but re-introduces MaxEnt RL in the policy gradient objectives. The policy target entropy is annealed from $0.5 \times dim(\mathrm{A})$ to $-1 \times dim(\mathrm{A})$ in the first 500K steps, which was done to mimic the exploration standard deviation annealing (from 1 to 0.2 in the first 500K steps) again from the DrQv2 implementation. Both the other MaxEnt RL parameters and SSPG-specific parameters are kept identical to our experiments on Mujoco tasks. See Table 2 for a full list of hyper-parameters.

C.3 Positional bandits

To provide visualizations and intuition for the behavior of RL agents adopting the serial Markov chain reasoning framework, we design simple few-dimensional *toy tasks*, which we call *positional bandits*. We used a 1-dimensional positional bandit for the example in Figure 2 and 2-dimensional positional bandits for our expressiveness experiments in Section 4.2. Positional bandits are defined by an arbitrary list of goal coordinates within a bounded state-space centered around 0. The action space represents position coordinates, determining which part of the state-space the agent will immediately visit. The reward of each action is proportional to the distance it brings the agent to the closest goal coordinate. Thus, in each *single-step episode*, the agent receives *information* about its proximity to only one of the goals. In these environments, we train all agents with the MaxEnt objective, making optimal behavior correspond to visiting all-goals with similar frequencies. These minimal problems aim to isolate and assess the ability of different agents of modeling arbitrary multi-modal distributions of behavior in the *non i.i.d.* RL problem setting which also involves a non-trivial exploration challenge.

We evaluate light-weight versions of SSPG, SAC, and a flow-based version of SAC. We largely follow the specifications from the Mujoco experiments, with a shared UTD of 1, a single critic, 50 random exploration steps, a fixed α, and with all models (except the flow-based policy) using 2-layer fully-connected architectures with 32 hidden dimensions. We implemented the flow-based version of SAC, based on inverse autoregressive flows [101], as also considered in related RL works [62, 75]. In particular, we apply to the policy model's outputs two additional flow transformations, each parameterized by a 2-layer fully connected network. We found that this class of models benefits from wider policy networks, thus, we increased back the hidden dimensionality to 256 (as in our main Mujoco experiments). We found 1000 steps of experience collection/network updates was sufficient for behavior to convergence in SSPG and standard SAC. Instead, the flow-based version of SAC required much additional experience/training before attaining behavior resembling any actual multi-modality, hence, we considered its results after 1M steps.

C.4 Other considerations

In this work, we implemented SSPG with minimal extensions to existing policy gradient models and parameters. The purpose of this choice was to remove confounding factors related to hyper-parameter and model tuning, and evaluate our framework mainly for its conceptual properties over traditional RL agents. For the same reason, we concentrated our empirical analysis on established evaluation benchmarks and toy problems, rather than exploring new challenging applications. This leaves much potential for future work to investigate new model architectures, training practices, and push the limit of *serial Markov chain reasoning*.

Early versions of our framework considered using some recurrent memory components as input to the BT-policy. However, to preserve the Markov property, this modification would require considering an extended state space for the RMC, consisting of both action-beliefs and memory hidden states. Preliminary results showed that such extended state space makes convergence require an increased amount of reasoning steps and optimization of the BT-policy more unstable.

D Extended evaluation results

In this Section, we provide granular per-task results and further aggregate evaluations with the *Rliable* evaluation protocol [55] after collecting different numbers of steps. We also compare the performance of SSPG with a new baseline integrating prior state-of-the-art algorithms with the normalizing flow model introduced in the first part of Section 4.2. Additionally, we report the mean running times for the considered algorithms and the main baselines we run to obtain performance results. Together with details about our hardware setup, these should give a solid intuition of the relative computational training cost of each algorithm. Furthermore, we also provide results for the performance of SSPG when 'clipping' the maximum number of reasoning steps allowed for each action-selection, and the relative evaluation times to analyze potential deployment-time efficiency and trade-offs.

D.1 OpenAI Gym Mujoco

Table 3: Per-task results for the considered OpenAI Gym Mujoco tasks. Each displayed return is averaged over 5 random seeds from 100 test rollouts from the preceding 10K training steps.

	100K steps					200K steps				
Tasks	**SSPG**	REDQ	IAPO	MBPO	SAC-20	**SSPG**	REDQ	IAPO	MBPO	SAC-20
Invertedpendulum-v2	1000±0	1000±0	1000±0	963±37	1000±0	1000±0	1000±0	1000±0	963±37	1000±0
Hopper-v2	**3314±68**	2994±510	425±229	3271±192	2718±908	**3487±87**	3060±617	426±149	3303±203	3356±26
Walker2d-v2	**4428±230**	1989±1003	476±107	3393±528	2043±757	**4793±186**	2969±861	570±74	4034±485	3039±903
Halfcheetah-v2	8897±496	5613±436	4122±566	**9533±332**	5831±723	10309±653	6633±568	5303±597	**10670±750**	7187±839
Ant-v2	**5163±275**	3132±1243	5±19	1596±446	496±105	**5513±238**	3792±1064	118±83	4309±632	1801±776
Humanoid-v2	**4992±140**	1402±657	441±90	559±62	495±96	**5148±51**	4721±648	390±160	3316±774	405±200

Per-task results. In Table 3 we provide the per-task results collected at 100K and 200K steps for the considered OpenAI Gym Mujoco tasks. For both experience thresholds, SSPG obtains the best average performance in 5/6 tasks, and still lags very close the model-based MBPO [61] algorithm for the remaining task (HalfCheetah-v2). SSPG converges much earlier than other algorithms, even while performing many less optimization steps (REDQ, REDQ-FLOW, MBPO, and SAC-20 all employ a UTD of 20, while we train SSPG with a UTD of 10, see Section 4).

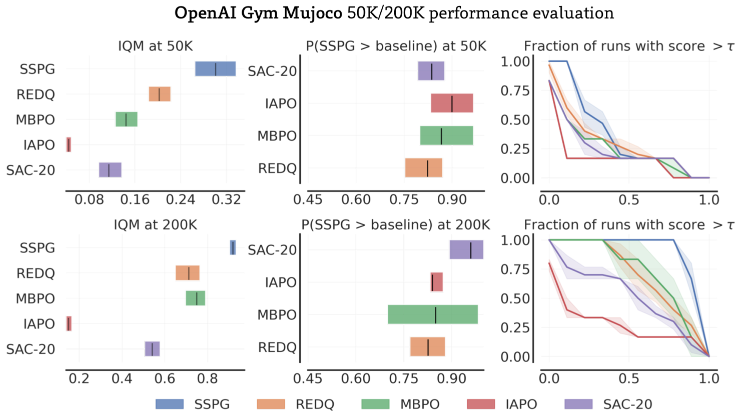


Figure 6: OpenAI Gym Mujoco aggregate performance evaluation with *Rliable* [55] after 50K (**Top**) and 200K (**Bottom**) training steps. These results complement the analogous results after 100K training steps together with the performance curves provided in Section 4.

Extended aggregate results. In Figure 6, we provide additional aggregate metrics collected at 50K and 200K steps, using the same statistical tools described in Section 4. We see most of the considerations for the evaluation at 100K steps, equally hold at these different experience regimes. Most notably, SSPG aggregately outperforms all prior algorithms with *statistically meaningful* gains, for the Neyman-Pearson statistical testing criterion [64] and also exhibits stochastic dominance [65] after 200K training steps.

D.2 DeepMind Control

Table 4: Per-task results for the considered DeepMind Control tasks. Each displayed return is averaged over 5 random seeds from 100 test rollouts from the preceding 150K training steps.

	1.5M steps					3M steps				
Tasks	**SSPG**	DrQv2	DrQ	CURL	SAC	**SSPG**	DrQv2	DrQ	CURL	SAC
acrobot_swingup	218±49	**272±40**	24±18	6±3	8±5	371±41	**422±48**	43±37	6±4	10±6
cartpole_swingup_sparse	**797±43**	478±391	321±393	497±330	135±268	**837±15**	503±411	319±391	530±353	174±290
cheetah_run	755±47	**781±32**	777±63	520±106	8±6	**888±10**	873±55	832±31	589±93	7±7
finger_turn_easy	**794±127**	757±156	184±63	305±107	170±60	**974±6**	932±43	218±106	310±139	211±98
finger_turn_hard	**637±138**	506±229	89±48	224±134	79±53	**945±42**	913±60	100±65	172±75	100±55
hopper_hop	**246±28**	200±102	272±88	185±129	0±0	**344±28**	239±123	290±86	223±133	0±0
quadruped_run	**570±22**	402±213	135±77	182±97	61±47	**760±64**	494±288	115±36	170±86	56±32
quadruped_walk	**855±23**	591±271	147±131	126±41	68±52	888±22	**905±44**	124±40	154±31	52±28
reach_duplo	**221±7**	219±7	9±15	10±11	0±1	218±9	**228±1**	9±6	7±7	2±2
reacher_easy	**978±4**	973±3	587±198	713±87	64±59	**982±3**	954±22	600±163	645±156	100±61
reacher_hard	**913±77**	802±113	343±242	482±179	7±15	**974±6**	944±25	426±288	651±324	15±19
walker_run	**634±16**	568±273	477±153	378±235	26±4	**738±7**	616±297	549±139	449±223	26±4
Average score	**634.79**	545.72	280.34	302.20	52.28	**743.32**	668.60	301.97	325.41	62.77
Median score	**695.85**	537.37	228.13	264.37	43.49	**862.73**	744.66	253.97	266.34	39.08

Per-task results. In Table 4 we provide the per-task results collected at 1.5M and 3M steps for the considered DeepMind Control tasks from pixel observations from [66]. SSPG obtains the best average performance in 10/12 and 9/12 tasks, respectively. We can observe most notable gains in some of the tasks that pose harder exploration challenges (e.g., cartpole_swingup_sparse) and especially in lower-data regimes (e.g., finger_turn_hard). We believe this could be an indication that *serial Markov chain reasoning* complements particularly well the MaxEnt reinforcement learning framework and is able to overcome some of its observed limitations for tackling sparse reward, pixel-based tasks [66].

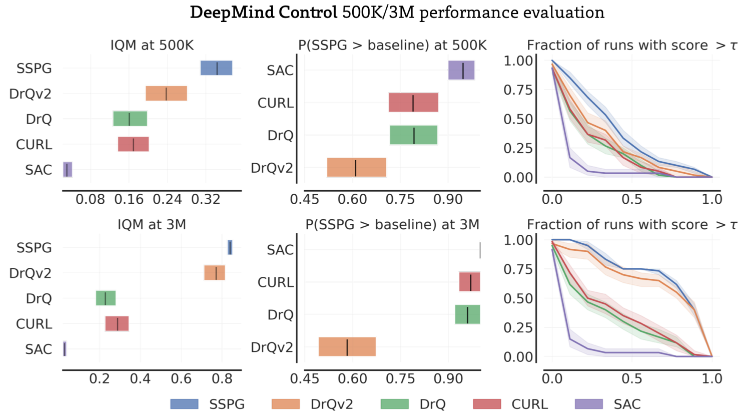


Figure 7: DeepMind Control aggregate performance evaluation with *Rliable* [55] after 500K (**Top**) and 3M (**Bottom**) training steps. These results complement the analogous results after 1.5M training steps together with the performance curves provided in Section 4.

Extended aggregate results. In Figure 7, we provide additional aggregate metrics collected at 500K and 3M steps, using the same statistical tools described in Section 4. Again, these results are consistent with the evaluation at 1.5M steps. In particular, we highlight ubiquitous statistical significance and stochastic dominance [65] after training from 500K environment steps.

D.3 Comparison with normalizing flows

We provide results comparing SSPG with additional new baselines, extending state-of-the-art algorithms based on the traditional RL framework with the normalizing flow policy model considered in Section 4.2. To counteract the exploration collapse phenomenon observed in the positional bandits experiments, we found it beneficial to add a small amount of fixed Gaussian noise on top of the already-stochastic policy when collecting training data ($\sigma = 0.05$). While powerful flow models can also provide unlimited expressiveness, they are still based on the traditional RL framework and represent decision-making as the output of a *fixed* process. Thus, these experiments can provide insights regarding the contribution of adaptivity to our SSPG framework. We compare the relative improvements from SSPG and flows with respect to the original performance of the considered state-of-the-art baselines in either the OpenAI Gym Mujoco and DeepMind Control benchmarks.

Table 5: Per-task results comparing improvements from SSPG and normalizing flows over REDQ for the considered OpenAI Gym Mujoco tasks. The results were collected as described in Table 3.

	100K frames			200K frames		
Tasks	**SSPG**	REDQ-FLOW	REDQ	**SSPG**	REDQ-FLOW	REDQ
Invertedpendulum-v2	**1000±0 (0%)**	868±60 (-13%)	1000±0	**1000±0 (0%)**	960±69 (-4%)	1000±0
Hopper-v2	**3314±68 (+11%)**	2476±480 (-17%)	2994±510	**3487±87 (+14%)**	2940±420 (-4%)	3060±617
Walker2d-v2	**4428±230 (+123%)**	3477±440 (+75%)	1989±1003	**4793±186 (+61%)**	4199±565 (+41%)	2969±861
Halfcheetah-v2	8897±496 (+58%)	**8905±1034 (+59%)**	5613±436	**10309±653 (+55%)**	9510±1065 (+43%)	6633±568
Ant-v2	**5163±275 (+65%)**	3695±987 (+18%)	3132±1243	**5513±238 (+45%)**	4750±795 (+25%)	3792±1064
Humanoid-v2	**4992±140 (+256%)**	3947±1542 (+181%)	1402±657	**5148±51 (+9%)**	4693±238 (-1%)	4721±648

In Table 5 we provide the per-task comparisons. incorporating flows with REDQ appears to provide limited and inconsistent performance benefits as compared to incorporating our serial Markov chain reasoning framework. This is in line with our intuition justifying SSPG's superiority and with results reported in prior off-policy reinforcement learning work which show flows provide marginal gains over standard Gaussian or deterministic policies (e.g., [62, 75]).

Table 6: Per-task results comparing improvements from SSPG and normalizing flows over DrQv2 for the considered DeepMind Control tasks. The results were collected as described in Table 4.

	1.5M frames			3.0M frames		
Tasks	**SSPG**	DrQv2-FLOW	DrQv2	**SSPG**	DrQv2-FLOW	DrQv2
Acrobot swingup	218±49 (-20%)	**296±16 (+9%)**	272±40	371±41 (-12%)	398±37 (-5%)	**422±48**
Cartpole swingup sparse	**797±43 (+67%)**	622±360 (+30%)	478±391	**837±15 (+67%)**	633±366 (+26%)	503±411
Cheetah run	755±47 (-3%)	743±58 (-5%)	**781±32**	**888±10 (+2%)**	849±51 (-3%)	873±55
Finger turn easy	794±127 (+5%)	**834±120 (+10%)**	757±156	**974±6 (+5%)**	946±50 (+2%)	932±43
Finger turn hard	**637±138 (+26%)**	555±222 (+10%)	506±229	**945±42 (+4%)**	866±60 (-5%)	913±60
Hopper hop	**246±28 (+23%)**	172±103 (-14%)	200±102	**344±28 (+44%)**	217±131 (-9%)	239±123
Quadruped run	570±22 (+42%)	**605±64 (+51%)**	402±213	**760±64 (+54%)**	734±20 (+48%)	494±288
Quadruped walk	**855±23 (+45%)**	831±52 (+41%)	591±271	888±22 (-2%)	**912±23 (+1%)**	905±44
Reach duplo	**221±7 (+1%)**	218±7 (-0%)	219±7	218±9 (-4%)	227±1 (-0%)	**228±1**
Reacher easy	**978±4 (+1%)**	976±3 (+0%)	973±3	**982±3 (+3%)**	973±16 (+2%)	954±22
Reacher hard	**913±77 (+14%)**	895±20 (+12%)	802±113	**974±6 (+3%)**	956±18 (+1%)	944±25
Walker run	**634±16 (+12%)**	520±287 (-9%)	568±273	**738±7 (+20%)**	575±319 (-7%)	616±297
Average score	**634.79 (+16%)**	605.41 (+11%)	545.72	**743.32 (+11%)**	690.45 (+3%)	668.60
Median score	**695.85 (+29%)**	613.28 (+14%)	537.37	**862.73 (+16%)**	791.32 (+6%)	744.66

In Table 5 we provide the per-task comparisons. Similarly to our results in the OpenAI Gym Mujoco tasks, incorporating flows with DrQv2 also appears to provide limited and inconsistent benefits as compared to the benefits of SSPG. We note that in both benchmarks, flows appear to particularly help earlier in training (100K steps and 1.5M steps, respectively), likely due to some initial exploration benefits. However, its gains over DrQv2 after 3M steps remain marginal in 10 out of the 12 examined DeepMind Control tasks.

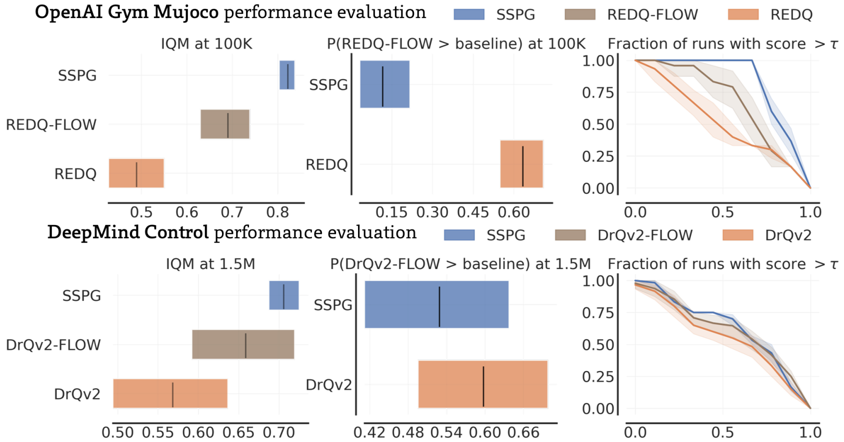


Figure 8: Performance evaluation of our normalizing flow agents with *Rliable* [55]. We integrate flows with REDQ for the OpenAI Gym Mujoco tasks (**Top**) and with DrQv2 for the DeepMind Control tasks from pixels (**Bottom**).

In Figure 8, we provide aggregate metrics for the normalizing flows agents, using the same statistical tools described in Section 4. We considered performance after 100K and 1.5M steps for the OpenAI Gym Mujoco and DeepMind Control benchmarks, since its where normalizing flows appear to yield the most benefits. However, even in this experience regime, incorporating flows does not provide statistically significant gains and does not exhibit stochastic dominance [65] as SSPG.

D.4 Computational requirements

Table 7: Average training times for the tested algorithms and ablations in the OpenAI Gym and DeepMind Control tasks.

OpenAI Gym Mujoco	Training time (seconds) for 1000 env. steps
SSPG (UTD=10)	50.9
SSPG (UTD=20)	84.2
SSPG, 1 reasoning step (UTD=20)	69.1
SAC-20	51.6
IAPO (Original implementation)	193.1
REDQ (Original implementation)	183.8
DeepMind Control (pixels)	**Training time (seconds) for 10000 env. steps**
SSPG (UTD=0.5)	160.2
SSPG, 1 reasoning step (UTD=0.5)	116.7
DrQv2	111.2

We record the average training time from executing the different considered algorithms for each *training epoch*, consisting of 1000 environment steps for OpenAI Gym and 10000 environment steps for DeepMind Control. The most relevant specifications of our hardware setup are an *NVIDIA RTX 3090* GPU and an *AMD Ryzen Threadripper 3970x* CPU. We based our implementations on the shared code from [98] and [66] for OpenAI Gym and DeepMind Control tasks, respectively.

Training times. We show the average training times in Table 7. For the OpenAI Gym Mujoco tasks, the increased UTD ratio severely affects computation requirements. Thus, by halving the UTD ratio of SSPG, we are able to bring its epoch training time lower than SAC-20 and all other considered baselines. However, we did not explore lowering the number of updates for our experiments in DeepMind Control, as DrQv2 already employs a low UTD ratio of 0.5. Thus, DrQv2 is about 30% computationally faster than SSPG. We believe the extra computation for carrying out the reasoning process is one of the main limitations of the *serial Markov chain reasoning* framework. However, for real-world problems, physical constraints related to environment feedback are often the main bottlenecks for the overall time efficiency. In such regard, we argue that the considerable sample-efficiency improvements from our novel framework should be much more relevant than increased computational cost.

D.5 Analysis of the cost of acting during deployment

Table 8: Performance of SSPG after 'clipping' the maximum number of reasoning steps during evaluation only. We write the number of clipping steps after the algorithm's name.

Tasks	SSPG	SSPG-1	SSPG-4	SSPG-8	REDQ
Ant-v2	5513±238	5356±261	5501±114	5527±219	3792±1064
Humanoid-v2	5148±51	4971±98	5120±56	5139±76	4721±648
Tasks	SSPG	SSPG-1	SSPG-4	SSPG-8	DrQv2
cheetah_run	888±10	880±13	889±10	891±5	873±55
quadruped_run	760±64	715±85	734±81	752±66	494±288

We report the performance of SSPG when 'clipping' the maximum number of reasoning steps allowed for each action-selection to fixed values, during evaluation only. We evaluate final checkpoints of agents learned without any clipping by our unmodified SSPG. We consider a subset of four total environments from both OpenAI Gym Mujoco (i.e., Humanoid-v2, and Ant-v2) and DeepMind Control (i.e., cheetah_run, and quadruped_run) for which SSPG displays different average reasoning requirements. As shown in Table 8, clipping to as low as four reasoning steps marginally affects the performance of SSPG, which still always surpasses the scores achieved by standard reinforcement learning baselines. SSPG is less affected by this form of deployment-only clipping than by fixing the number of reasoning steps for both training and evaluation phases. We motivate this finding by noting that better capturing the canonical distribution of returns from the critic by performing additional reasoning steps has also significant benefits during the training phase of off-policy algorithms. In particular, this allows the agent to achieve better exploration and more easily correct the critic in the areas of the action space where its predictions are erroneously optimistic. When training with the unmodified SSPG algorithm, this benefit is fully retained, justifying the performance superiority than our previous fixed-steps ablations.

Table 9: Average deployment times for the tested algorithms and ablations in the OpenAI Gym and DeepMind Control tasks.

OpenAI Gym Mujoco	Deployment time (seconds) for 1000 env. steps
Random (only simulation)	0.583
SSPG	3.523
SSPG-1 (clipping)	2.462
SSPG-4 (clipping)	2.917
SSPG-8 (clipping)	3.257
SAC-20	2.451
IAPO (Original implementation)	5.657
REDQ (Original implementation)	2.671
DeepMind Control (pixels)	Deployment time (seconds) for 10000 env. steps
Random (only simulation)	0.997
SSPG	1.701
SSPG-1 (clipping)	1.637
SSPG-4 (clipping)	1.693
SSPG-8 (clipping)	1.699
DrQv2	1.604

We report the average rollout time during deployment of each of our implementations. As shown in Table 9, using SSPG does increase the average rollout time over standard reinforcement learning baselines. However, the additional time required for action-selection scales sub-linearly with the number of reasoning steps, and appears to be dominated by other fixed costs, such as simulating the environment and converting observations to tensor objects. This is in contrast with the other more expensive iterative baseline, IAPO [62], which performs gradient-based optimization at each acting step. Differences with standard reinforcement learning are even more marginal in the visual DeepMind Control environments, where the most expensive part of the computation is from encoding the RGB input observation with a convolutional encoder (which needs to occur only once before performing the reasoning steps with the policy 'head'). Moreover, we note that in many real-world applications, the additional acting cost would be greatly inferior to the actuation time costs when using distributed hardware. However, clipping the number of reasoning steps still remains a valuable option, as examined in the experiment above.

E Ablations and parameter studies

We perform additional experiments to evaluate SSPG's design choices and test alternative optimization setups for *serial Markov chain reasoning*. We focus on the Humanoid-v2 and quadruped_run tasks, which we found to be generally representative of overall agent behavior in the OpenAI Gym Mujoco and DeepMind Control domains, respectively. For each experiment, we report the performance curves for average returns (higher is better) and number of reasoning steps (lower is better).

E.1 Loss backpropagation

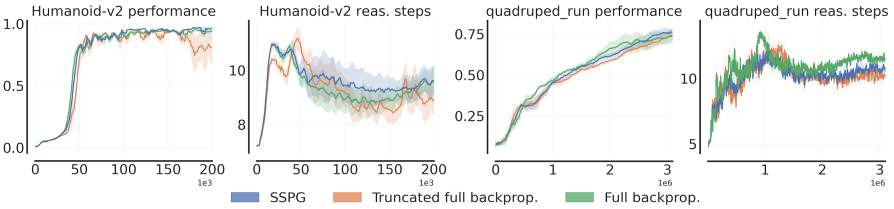


Figure 9: Performance and reasoning steps with alternative backpropagation strategies for training SSPG's BT-policy.

Following the policy gradient estimation derived in Section 3.2, we optimize the BT-policy by recording its gradients with respect to the following operations: 1) We perform one (reparameterized) reasoning step to output an action-belief, $a_0 \sim \pi^b_\theta(s, a)$. 2) We simulate a *local approximation* of the (reparameterized) RMC for N steps, yielding $a_{1:N}$. 3) We compute and backpropagate the outputs of all $Q^s_\phi(s, a_n)$, for $n = 0, ..., N$. A practical alternative would be to carry out step (2) by performing N reasoning steps with the (reparameterized) BT-policy itself, rather than with a local approximation of the RMC. While this change would not affect the objective value from each $Q^s_\phi(s, a_n)$, it would make the computation record additional gradient dependencies to the BT-policy's parameters *from each reasoning step*, rather than just the first. As we already have to perform forward and backward passes for the N actions computed in step (2), this extension would not come at a significant computational overhead and would be reminiscent of differentiation through a traditional recurrent model. In practice, we expect this change would enable to reduce optimization variance, as we are optimizing the *direct influence* that the BT-policy has on the RMC starting from N different initial action-beliefs for each sample. Yet, this alternative optimization procedure is not supported by our theoretical intuition and would likely introduce further bias into the policy gradient estimation.

We show experimental results in Figure 9. We evaluate both a truncated version (truncated full backprop.) and an un-truncated version (full backprop.) that perform *full backpropagation* through the RMC when optimizing SSPG. The truncated version still optimizes the BT-policy with respect to the full N reasoning steps, but prevents the gradients of future steps backpropagating to earlier action-beliefs (somewhat reminiscent of truncation in RNN optimization). Surprisingly, we observe no major differences performance-wise. However, we do observe somewhat higher variance between different experiments when performing full backpropagation and slightly lower final returns, likely symptomatic of the additional bias. On the other hand, the un-truncated version does experience marginally faster initial learning in quadruped_run, indicating there could be some practical benefits from backpropagating multiple reasoning steps to the BT-policy's parameters. We leave exploration of such potential extensions for future work.

E.2 Initial action beliefs

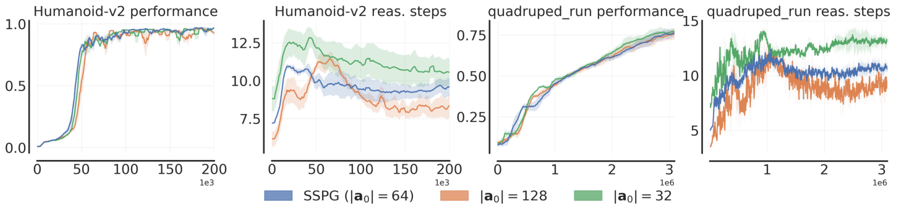


Figure 10: Performance and reasoning steps with double and half SSPG's number of initial action-beliefs.

As described in Section 3.3, we perform the reasoning process from a set of initial action-beliefs $\mathbf{a}_0$. The size of $\mathbf{a}_0$ (denoted $|\mathbf{a}_0|$) influences the number of initial starting modes and the accuracy of the empirical PSRF computation for convergence detection. By default SSPG employs $|\mathbf{a}_0| = 64$. As shown in Figure 10, doubling or halving $|\mathbf{a}_0|$ affects convergence speed, with a greater number of initial action-beliefs leading to a slightly faster reasoning process. We attribute this phenomenon to the conservativeness of the proposed PSRF convergence rule [52], which becomes harder to satisfy when evaluated from fewer parallel chains. However, these results do not take into account the cost of performing each reasoning step which is itself affected by $|\mathbf{a}_0|$. SSPG's overall efficiency should be quite robust to most choices for this hyper-parameter with hardware optimized for parallel computation.

E.3 Short term action memory buffer size

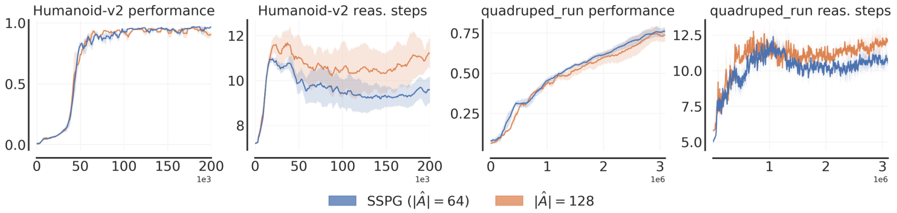
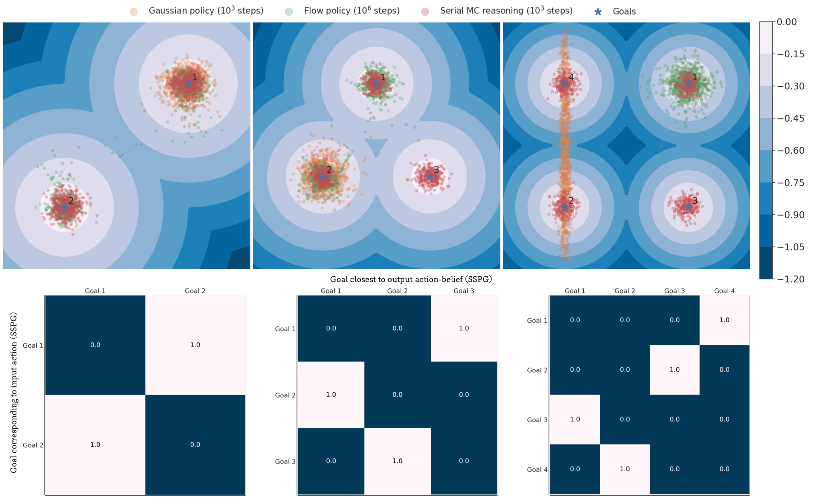


Figure 11: Performance and reasoning steps using a *short-term action memory* buffer with increased size.

As described in Section 3.3, we sample initial action-beliefs for reasoning from the *short-term action memory* buffer, $\hat{A}$. The size of $\hat{A}$ (denoted $|\hat{A}|$) determines how far into the past to store action-beliefs that can facilitate future reasoning. In practice, we found that the simple heuristic of setting $|\hat{A}| = |\mathbf{a}_0|$ works well. In Figure 11, we examine the effects of doubling the buffer size from 64 to 128. Again, we observe this change leads to an increase in the expected reasoning steps before reaching convergence. This result indicates that the most recent past actions are generally most relevant, which we believe is a consequence of the fine temporal discretization of the considered environments.

F Additional supporting analysis results

F.1 Positional bandits quantized transition probabilities



	Goal 1	Goal 2
Goal 1	0.0	1.0
Goal 2	1.0	0.0

	Goal 1	Goal 2	Goal 3
Goal 1	0.0	0.0	1.0
Goal 2	1.0	0.0	0.0
Goal 3	0.0	1.0	0.0

	Goal 1	Goal 2	Goal 3	Goal 4
Goal 1	0.0	0.0	0.0	1.0
Goal 2	0.0	0.0	1.0	0.0
Goal 3	1.0	0.0	0.0	0.0
Goal 4	0.0	1.0	0.0	0.0

Figure 12: Quantized transition matrices visualizations for the learned BT-policy trained in the positional bandits with 2, 3, and 4 goals (see Figure 4 A).

We compute the quantized transition probabilities of the RMCs in the considered positional bandits from our *Policy expressiveness* analysis (Section 4.2). For each positional bandit we randomly sample a set of 1000 action-beliefs within a radius around each goal and compute the likelihood of transitioning between any two such action-beliefs. We sum and normalize these probabilities both with respect to the input and output action beliefs within each goal. Hence, we obtain a transition matrix for the discretized RMC, showing the probability of updating the current action-belief based on the resulting closest goal. We find that the BT-policy intuitively learns to consider action-beliefs to reach all different goals in turn. For instance, in the positional bandit with three goals, performing a reasoning step with the learned BT-policy from an action-belief that would land nearby goal 1 would almost certainly lead to an action-belief that would land near goal 3. Similarly, an action-belief landing near goal 2 would follow from an action-belief that would land near goal 3, and would almost certainly lead to an action-belief that would land near goal 1 (with less than 10^{-6} probability of deviating). This behavior is summarized from visualizing the relative transition matrices following the described quantization in Figure 12.

F.2 Reasoning with fixed action beliefs

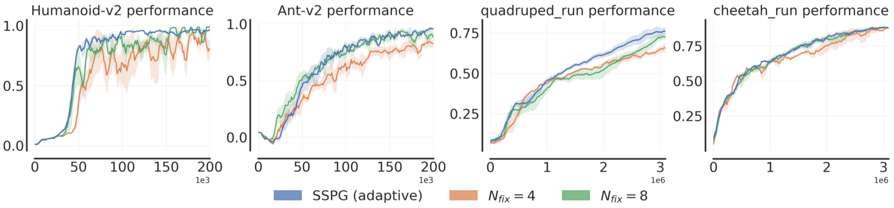


Figure 13: Performance from performing reasoning with alternative *fixed* number of steps in different tasks. The average number of reasoning steps until convergence with SSPG for Humanoid-v2 quadruped_run is around 10, while for Ant-v2 and cheetah_run is around 5.

We provide further results in support of our *Policy adaptivity* analysis by comparing the performance of our adaptive SSPG with performing alternative fixed numbers of reasoning steps in 4 different tasks with diverse average reasoning costs (see Figure 4 B). As shown in Figure 13 and also Figure 4 C, increasing the number of reasoning steps monotonically improves performance. Yet, we did not observe any case where our adaptive strategy underperforms as compared to any fixed number of steps. We believe this is further empirical evidence of how a framework that can adaptively dedicates different amounts of computation to different action-selection problem can provide great efficiency and scaling benefits.

F.3 Effects of short-term action memory buffer

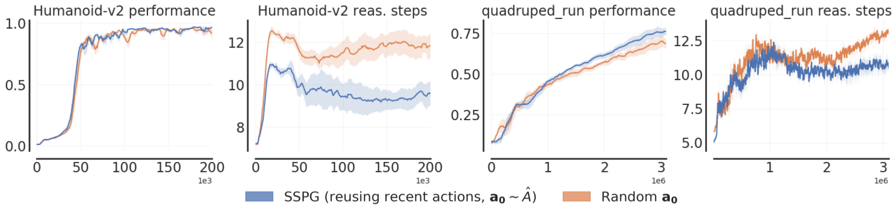


Figure 14: Performance and reasoning steps with and without reusing recent actions as initial action-beliefs.

As stated in the *Solution reuse* analysis, and shown in Figure 14, re-initializing randomly $\mathbf{a}_0$ hurts efficiency but not performance. We believe this is an indication that the BT-policy effectively learns to *bootstrap* information in each previous action-belief while reasoning. Intuitively, starting from an initial action-belief that contains no information about optimal behavior makes the relative action-selection problem harder. Yet, SSPG still successfully enables to recover very similar performance via adaptively performing additional reasoning steps to compensate.

G Societal impact

We proposed a new framework for modeling autonomous decision-making in reinforcement learning. Thus, the societal impact of our work is tightly related to the impact of the reinforcement learning field. While autonomous systems can offer many benefits, as they become more commonplace, they might introduce ethical issues such as privacy, surveillance, and bias. If unregulated, automation might also accentuate economic inequalities and have non-trivial environment impact. Policy making strategies and regulations are increasingly needed to mitigate these risks, which we believe should not compromise the field's advancements given its countless potential positive implications.