Policy Learning and Evaluation with Randomized Quasi-Monte Carlo

Sébastien M. R. Arnold
seb.arnold@usc.edu
U. of Southern California

Pierre L'Ecuyer
lecuyer@iro.umontreal.ca
U. de Montréal

Liyu Chen
liyuc@usc.edu
U. of Southern California

Yi-fan Chen
yifanchen@google.com
Google Research

Fei Sha
fsha@google.com
Google Research

Abstract

Hard integrals arise frequently in reinforcement learning, for example when computing expectations in policy evaluation and policy iteration. They are often analytically intractable and typically estimated with Monte Carlo methods, whose sampling contributes to high variance in policy values and gradients. In this work, we propose to replace Monte Carlo samples with low-discrepancy point sets. We combine policy gradient methods with Randomized Quasi-Monte Carlo, yielding variance-reduced formulations of policy gradient and actor-critic algorithms. These formulations are effective for policy evaluation and policy improvement, as they outperform state-of-the-art algorithms on standardized continuous control benchmarks. Our empirical analyses validate the intuition that replacing Monte Carlo with Quasi-Monte Carlo yields significantly more accurate gradient estimates.

1 INTRODUCTION

Policy gradient methods are widely used in reinforcement learning due to their broad applicability to optimizing linear or nonlinear policies on non-differentiable and stochastic objectives. In particular, they handle naturally high-dimensional and continuous action spaces (Benbrahim and Franklin, 1997; Peters et al., 2003; Schulman et al., 2015; Rajeswaran et al., 2018).

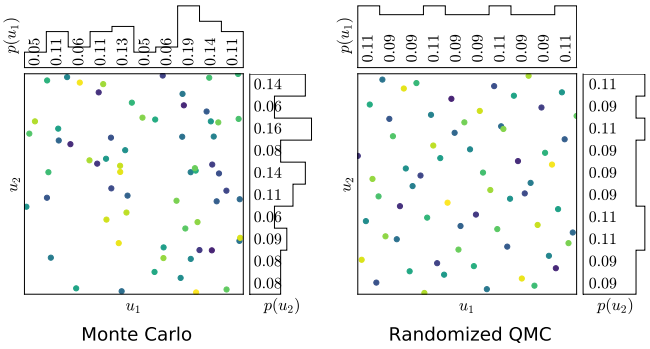

Figure 1: **Comparing MC and RQMC point sets.** 64 points drawn uniformly over the space $[0, 1)^2$ with Monte-Carlo (MC) and randomized Quasi-Monte Carlo (RQMC). Randomized QMC point sets minimize discrepancy thus covering the space more evenly, as indicated by the histograms of the marginals. These point sets improve the estimation of the expectation in Equation (1), under some smoothness assumptions.

At the core of those methods is to evaluate policies and improve them. Various formulations require to compute some integrals over state and action spaces that are often analyticaly intractable. In practice, those integrals are approximated and Monte Carlo (MC) sampling is one of the popular approaches to do so (Metropolis and Ulam, 1949). This approximation is inherently stochastic; the exact choice and implementation of MC variants directly impacts the quality of the estimated policy value and policy gradients, due to the variance in the approximation. Increasing the number of MC samples reduces the variance but it increases the computational cost as well as the need for the agents to interact with the environments (or their simulators). Specifically, the vanilla MC sampling, often implemented in

Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151.

practice as drawing from a random number generator (of a known distribution), has a convergence rate of $\mathcal{O}\left(N^{-1/2}\right)$, where N is the number of samples.

In this work, we explore using Randomized Quasi-Monte Carlo (RQMC), a type of Quasi Monte Carlo (QMC) methods as a drop-in replacement of the vanilla sampler (Niederreiter, 1978). The key difference of QMC from the standard MC is that QMC uses deterministic point sets to approximate stochastic integration. Under appropriate conditions, including that the integrated function has bounded Hardy-Krause variation, QMC can asymptotically improve the convergence rate from $\mathcal{O}\left(N^{-1/2}\right)$ to nearly $\mathcal{O}\left(N^{-1}\right)$ (Papageorgiou, 2003). RQMC improves the efficiency when multiple point sets are needed – a scenario we will encounter when we need to compute confidence intervals of policy values or compute policy gradients.

Figure 1 demonstrates the difference between the vanilla MC and RQMC with 2-d point sets. The RQMC point set covers the unit square more evenly than the MC's. Our empirical studies on several tasks demonstrate clearly the utility of this property in improving the estimation of both policy value and gradients.

Other types of methods for reducing variance in estimation have also been explored: control variates such as optimal baselines (Greensmith et al., 2001a; Lawrence et al., 2002; Weaver and Tao, 2001a) and value functions (Mnih et al., 2016; Schulman et al., 2016), carefully designed optimization methods (Frostig et al., 2015; Le Roux et al., 2012; Defazio et al., 2014; Shalev-Shwartz and Zhang, 2013). They are orthogonal to how to obtain samples for integration, and our work also explore the possibility of combining them with RQMC to gain further improvement in estimation quality.

Our main contribution is the introduction of the approach and the empirical studies of its performance and analysis. Our empirical results establish that RQMC always outperforms MC on both policy evaluation and improvement. In particular, RQMC improves sample complexity while reducing policy evaluation error at an order of magnitude. Further analysis shows that the strong performance of RQMC is due to reduced gradient variances and better alignment with the true gradients. RQMC is competitive with other variance reduction techniques, and we show it can be effectively combined with those other techniques to obtain best performance.

The rest of the paper is organized as follows. We introduce the basic idea behind Quasi-Monte Carlo methods and more specifically, RQMC (§2). We show how to apply RQMC for policy evaluation and policy improvement (§3). In §4, we empirically study the performance of MC and RQMC for policy evaluation and policy improvement on continuous control problems (*e.g.*, Brownian motion, LQR, and MuJoCo).

2 BACKGROUND

Monte Carlo methods (MC) have been used widely in computational statistics for estimating stochastic integrals. Consider the problem of estimating the expectation of a function f over the d-dimensional uniform distribution U^d on the unit cube $[0,1)^d$. We obtain an MC estimate by averaging the value of f for N vectors $u^{(i)} \in \mathbb{R}^d$ sampled from U^d:

$$\mathop{\mathbb{E}}_{u\sim U^d}[f(u)] \approx \frac{1}{N}\sum_{i=1}^{N} f(u^{(i)}). \tag{1}$$

It is well-known that the mean-squared error of this MC estimator converges at a rate of $\mathcal{O}\left(N^{-1/2}\right)$ (Robert and Casella, 2004).

Quasi Monte Carlo (QMC) The basic idea behind all QMC methods is to improve the standard MC convergence rate by replacing the random samples $u^{(i)}$ with a *deterministic, low discrepancy* point set. Intuitively, we construct this low-discrepancy point set so as to maximize the distance between each point $u^{(i)}$, thus providing a more even coverage of $[0,1)^d$, see Figure 1 for an illustration.

QMC with such low-discrepancy point sets can reach rates of convergence arbitrarily close to $\mathcal{O}\left(N^{-1}\right)$ when f satisfies some regularity conditions (Papageorgiou, 2003). In comparison, common variance-reduction techniques in reinforcement learning (*e.g.*, control variates or importance sampling) only improve the constant factors in front of the $\mathcal{O}\left(N^{-1/2}\right)$ rate (Glynn and Szechtman, 2002; Elvira and Martino, 2021).

We briefly present a concrete way – "digital nets in base 2" – to construct point sets as described by L'Ecuyer (2018, § 2.3). Other constructions such as stratification and lattice rules are detailed in (Niederreiter, 1992). For $N = 2^k$ points in d dimensions, we begin by choosing d matrices $C_1, \ldots, C_d \in [0,1]^{k\times k}$, *e.g.*, the upper-triangular non-singular ones generating the Sobol sequence (Sobol, 1976), as described by Joe and Kuo (2008). To obtain the i^{th} point $u^{(i)}$, we proceed in three steps. First, we obtain the binary vector representation of i: $b^{(i)} = [b_1^{(i)}, \ldots, b_k^{(i)}]$ such that $i = b_1^{(i)} + b_2^{(i)}2 + \cdots + b_k^{(i)}2^{k-1}$ for $i \in [1, N]$. Secondly, we multiply this representation with each

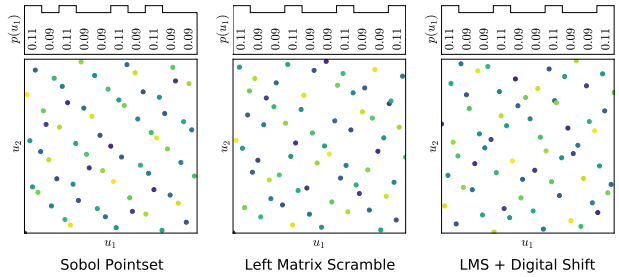

Figure 2: **Randomized point set Pipeline** QMC point sets (Sobol) are generated deterministically to minimize discrepancy, and cover the space more evenly. When several estimates are required, it is common to randomize the point set: first, by applying a left matrix scramble, then a digital shift to debias the point set.

generating matrix C_j:

$$\hat{u}_j^{(i)} = C_j b^{(i)} \mod 2, \qquad j = 1, \dots, d,$$

where the modulus is applied to each element of vector $\hat{u}_j^{(i)} \in \mathbb{R}^{k \times 1}$. Finally, we map this base-2 representation back to its decimal form: $u_j^{(i)} = \sum_{l=1}^{k} \hat{u}_{j,l}^{(i)} 2^{-l}$. The final point $u^{(i)} = [u_1^{(i)}, \dots, u_d^{(i)}]$ is the concatenation of all dimension values $u_1^{(i)}, \dots, u_d^{(i)}$. Note that the procedure is deterministic as the Sobol sequence (and its generating matrices) is determinstic — given the generating matrices, the QMC point set is unique.

Randomized QMC (RQMC) If we need two or more independent realizations of the expectation $\mathbb{E}_u[f(u)]$ (*e.g.*, for confidence intervals or iterative procedures), we can randomize QMC such that the resulting point set retains low-discrepancy while also yielding unbiased estimates (Owen, 1998; L'Ecuyer, 2018).

How to best randomize QMC point sets depends on the properties of f. For example, if f is sufficiently smooth, the convergence rate of RQMC can be improved to be close to $\mathcal{O}\left(N^{-3/2}\right)$ (Owen, 1997; Matoušek, 1998; Hickernell et al., 2000). Our presentation and experiments focus on randomization via *left matrix scrambling* followed by a *random digital shift*, as first introduced by Matoušek (1998). The operations are intuitively just random linear map and shift.

With the left matrix scramble, we scramble the point set by transforming generating matrices $C_1, \dots, C_d$ with a set of random binary matrices $L_1, \dots, L_d$. To that end, we first sample d lower-triangular, non-singular, random matrices $L_j \in [0,1]^{k \times k}$, $j = 1, \dots, d$ and multiply the generating matrices C_j:

$$C_j \leftarrow L_j C_j \mod 2, \qquad j = 1, \dots, d,$$

where the modulus is applied element-wise. We then generate our point set as for QMC, using the newly scrambled generating matrices $C_1, \dots, C_d$. While scrambling effectively and efficiently randomizes the point set, repeating this procedure over multiple point sets results in biased estimates (*e.g.*, $u_j^{(i)} = 0$ remains 0 across all point sets). Fortunately, this bias is easily addressed by computing the bit-wise exclusive-or of a random binary vector $v_j \in [0,1]^k$ to shift each point of the point set:

$$\hat{u}_j^{(i)} \leftarrow \hat{u}_j^{(i)} \oplus v_j, \qquad j = 1, \dots, d,$$

The procedure is illustrated in Figure 2; we refer the reader to (L'Ecuyer, 2018) for a more detailed exposition, and to the Supp. Material for interfaces with popular software packages.

Policy Gradient Methods Policy gradient methods learn a parameterized policy π_θ by computing the gradient of an objective function with respect to the policy parameters θ. This objective function is the expected value of the policy, denoted by

$$V^{\pi_\theta} = \mathop{\mathbb{E}}_{s,a}\left[Q^{\pi_\theta}(s,a)\right], \tag{2}$$

and defined over the states $s \in \mathcal{S}$ and actions $a \in \mathcal{A} = \mathbb{R}^{\dim(\mathcal{A})}$ of a Markov decision process. The states s are sampled according to the visitation frequency while sampling the actions a from the policy. While in principle the policy can be any distribution, our work assumes a multivariate Gaussian, suitable for continuous control:

$$a \sim \pi_\theta(s,u) = \mu_\theta(s) + \sigma_\theta(s) \odot F^{-1}(u), \tag{3}$$

where $\mu_\theta(s)$ and $\sigma_\theta^2(s)$ are the (parameterized) mean and diagonal covariance of the Gaussian for the state s, $\odot$ denotes element-wise product, F^{-1} is the inverse cumulative density function of the standard normal distribution, and $u \sim U^{\dim(\mathcal{A})}$ is a uniformly distributed random variable. Other policy distributions can be obtained via inverse transform sampling or with the probability integral transform. When clear from context, we omit the dependency of π_θ on u and let $\pi_\theta(a \mid s)$ denote the density for taking a in state s.

The policy gradient theorem (Sutton et al., 2000) states that the gradient of the policy value can be computed as:

$$\nabla V^{\pi_\theta} = \mathop{\mathbb{E}}_{s,a}\left[Q^{\pi_\theta}(s,a) \nabla_\theta \log \pi_\theta(a \mid s)\right], \tag{4}$$

where the right-hand side is obtained by applying the score-function estimator (Miller, 1967; Rubinstein, 1969) on the policy's action-value function $\hat{Q}^{\pi_\theta}$.

To estimate the gradient, the agent needs to interact with the environment by collecting trajectories of state and actions $(s_1, a_1, \ldots, s_T, a_T)$. Q^{π_θ} can be estimated as expected returns:

$$Q^{\pi_\theta}(s_k, a_k) = \mathop{\mathbb{E}}_{s_t, a_t}\left[\sum_{t=k}^{T} R(s_t, a_t)\right], \tag{5}$$

where $R(s, a)$ is the immediate reward observed when taking action a in state s.

Another approach, used in actor-critic methods, learns the action-value function (*i.e.*, the critic) $\hat{Q}^{\pi_\theta}$ with π_θ, for example, by minimizing the on-policy squared Bellman error:

$$\mathop{\mathbb{E}}_{s,a,s'}\left[\left(\mathop{\mathbb{E}}_{a'}\left[\hat{Q}^{\pi_\theta}(s', a')\right] + R(s, a) - \hat{Q}^{\pi_\theta}(s, a)\right)^2\right], \tag{6}$$

where the next-state action $a' \sim \pi_\theta(\cdot \mid s')$ is sampled from the current policy, and s, a, s' from a (possibly off-policy) replay buffer. The critic $\hat{Q}^{\pi_\theta}$ is then used in computing the policy gradient.

Algorithm 1 RQMC Policy Evaluation w/ Returns

```
from scipy.stats import Sobol
rqmc = Sobol(T · dim(A), scramble=True)
u = rqmc.random(N)   # u ∈ [0,1)^{N×(T·dim(A))}
R = 0
for i in 1,...,N:
    env.reset()
    for t in 1,...,T:
        a_t^(i) = μ_θ(s_t^(i)) + σ_θ(s_t^(i)) ⊙ F^{-1}(u_t^(i))
        R += env.step(a_t^(i))
return R/N
```

3 LEARNING AND EVALUATION WITH RANDOMIZED QUASI-MONTE CARLO

We now show how to use RQMC point sets for both policy evaluation and policy improvement. We consider both the vanilla policy gradient and actor-critic settings. With the former, an (approximate) action-value function is not available and the policy value must be estimated by sampling trajectories from the environment and collecting returns. This setting is more general than the actor-critic one, where the value functions are approximated with parameteric functions. The actor-critic methods tend to accelerate learning as illustrated by several recent state-of-the-art methods (Fujimoto et al., 2018; Haarnoja et al., 2018).

3.1 POLICY EVALUATION WITH RQMC

The goal of policy evaluation is to estimate the policy value V^{π_θ} for a given policy π_θ.

Through expected returns We estimate with

$$V^{\pi_\theta} \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} R(s_t^{(i)}, a_t^{(i)}), \tag{7}$$

where we have N trajectories $(s_1^{(i)}, a_1^{(i)}, \ldots, s_T^{(i)}, a_T^{(i)})$ for all i.

Because the stochasticity underlying the dynamics of the environment is typically inaccessible, we use RQMC to sample the action sequence. Concretely, we use RQMC to generate N points point $u^{(i)} \in \mathbb{R}^d$. Then, we obtain action $a_t^{(i)} = \pi_\theta(s_t^{(i)}, u_t^{(i)})$ by applying π_θ to $s_t^{(i)}$ and $u_t^{(i)}$ as in Equation (3), where $u_t^{(i)}$ is the t^{th} segment of length $\dim(\mathcal{A})$ from $u^{(i)}$ (*i.e.*, scalar entries $(t-1)\cdot\dim(A)$ to $t\cdot\dim(\mathcal{A})$ in vector $u^{(i)}$). Pseudocode for this approach is listed in Algorithm 1.

Note that since we wish to evaluate the *sum* of immediate rewards, RQMC is effectively integrating over all actions in the trajectory and each of our N points ought to have dimension $d = \dim(\mathcal{A}) \cdot T$ (for example, $\dim(\mathcal{A}) = 4$ for a robotic arm with 4 degrees of freedom, one for each of its 4 joints). If instead we were to use T independent RQMC point sets, each with N points of dimension $\dim(\mathcal{A})$, the estimator would remain unbiased w.r.t. uniformity but would lose low-discrepancy on $[0,1)^d$. In practice, this dependency of d on both the action dimension $\dim(\mathcal{A})$ and horizon T can be an impediment because RQMC implementations typically limit the integration dimension[1]. Fortunately, this impediment can be sidestepped with learned action-value functions.

Through learned critic The above procedure can be significantly simplified if we assume access to an approximate (parametric) form of action-value function $Q^{\pi_\theta}(s, a)$. We also assume access to a set of states used to compute the expectation over s in the right-hand side of Equation (7). Those states should be collected by rolling out the evaluated policy π_θ. Then, the expected returns can be approximated with

$$V^{\pi_\theta} \approx \mathop{\mathbb{E}}_{s_k}\left[\frac{1}{N}\sum_{i=1}^{N}\hat{Q}^{\pi_\theta}\left(s_k, \pi_\theta(s_k, u_k^{(i)})\right)\right], \tag{8}$$

where $(u_k^{(1)}, \ldots, u_k^{(N)})$ is the RQMC point set over $[0,1)^{\dim(\mathcal{A})}$ associated with state s_k. Note that be-

[1] E.g., the most popular implementation (Joe and Kuo, 2008) limits $d \leq 1,110$, while the latest versions of PyTorch and TensorFlow both limit $d \leq 21,201$.

cause $\hat{Q}^{\pi_\theta}$ implicitly computes the sum of rewards over the entire trajectory, the horizon T does not factor in the integration dimension $d = \dim(\mathcal{A})$, significantly reducing the sampling cost.

3.2 POLICY ITERATION WITH RQMC

In policy iteration, we compute the gradient of V^{π_θ} with respect to the policy parameters θ and use them in an iterative procedure, *e.g.*, stochastic gradient descent, to maximize the policy value.

When Q^{π_θ} is estimated through expected returns, policy iteration simply consists of taking the trajectories collected during policy evaluation as in Equation (7) and using the states s and executed actions a to compute the estimate in Equation (4). Thus, the RQMC procedure described above applies.

When the action-value functions are learned from a parameteric estimator $\hat{Q}^{\pi_\theta}$, for example, with Equation (6), we can compute the gradients in two ways. The first is to use the estimator as a plug-in:

$$\nabla V^{\pi_\theta} \approx \mathop{\mathbb{E}}_{s}\left[\frac{1}{N}\sum_{i=1}^{N}\hat{Q}^{\pi_\theta}(s,a^{(i)})\nabla \log \pi_\theta(a^{(i)} \mid s)\right] \quad (9)$$

where $a^{(i)} = \pi_\theta(s, u^{(i)})$.

Alternatively, when $\hat{Q}^{\pi_\theta}$ is differentiable with respect to the actions, for example, when $\hat{Q}^{\pi_\theta}$ is instantiated as a neural network, we can use the reparameterization trick and view the policy as a deterministic function and directly differentiate the action-value w.r.t. the policy parameters θ, yielding the following estimator:

$$\nabla V^{\pi_\theta} \approx \mathop{\mathbb{E}}_{s}\left[\frac{1}{N}\sum_{i=1}^{N}\nabla_{a^{(i)}}\hat{Q}^{\pi_\theta}\left(s,a^{(i)}\right)\nabla_\theta \pi_\theta(s,u^{(i)})\right]. \quad (10)$$

This type of deterministic policy gradient estimator (Silver et al., 2014) is common in recent state-of-the-art actor-critic methods (e.g., Lillicrap et al. (2016); Fujimoto et al. (2018); Haarnoja et al. (2018)), notably because it enables off-policy learning which drastically improves sample efficiency. We detail and provide an implementation of soft actor-critic (SAC; Haarnoja et al. (2018)) with RQMC in the Supp. Material.

In general, which approach to use is application-dependent as there are scenarios where the score-function estimator Equation (9) yields lower-variance than the deterministic policy gradient Equation (10) (Gal, 2016, § 3.1.2), and *vice-versa* (Mohamed et al., 2019, § 8.3). Moreover, both estimators can be combined (Gu et al., 2017) but it is unclear whether the interpolation yields any benefit. In our experiments, we focus on Equation (10) as it is the current state-of-the-art method for continuous control.

4 EXPERIMENTS

In this section, we demonstrate the utility of RQMC when applied to continuous control tasks. To that end, we propose to answer the following questions:

- How effective are the RQMC policy gradient and actor-critic formulations in ameliorating policy evaluation (§4.1) and improvement (§4.2) on a range of simulated continuous control tasks?
- Does RQMC effectively reduce variance and improve gradient estimation (§4.3)?
- How does RQMC fare against other variance reduction techniques, and can it complement them (§4.4)?

Tasks We conduct our study on three sets of standardized tasks: Brownian motion, Linear-Quadratic Regulator (LQR), as well as the popular MuJoCo robotics suite (Todorov et al., 2012). Throughout, we estimate Q^{π_θ} by sampling trajectories (Equation (7)) for the Brownian motion and LQR tasks, and learn a parameterized $\hat{Q}^{\pi_\theta}(s,a)$ with the soft actor-critic (SAC) of Haarnoja et al. (2018) for the MuJoCo tasks. We randomize point sets with the left matrix scramble and digital shift for Brownian motion and LQR, and with Owen's scrambling (Owen, 1998) for MuJoCo tasks.

The Brownian motion environment – as studied in (L'Ecuyer et al., 2008, § 4.5) – consists of a 1D point-mass particle whose objective is to reach the origin. At every step, an action is sampled from a Gaussian distribution parameterized by its mean and covariance. The reward is the Euclidean distance of the particle's current position to the origin, and we set the horizon to 20 timesteps.

The LQR environment is a well-studied testbed from the optimal control literature which provides a unified formalism for many continuous control problems. LQR has also been the focus of recent theoretical studies of policy gradient methods (Fazel et al., 2018; Recht, 2019). The transition dynamics of LQR are a linear function of the state and action, while the reward is a quadratic function of both values. In our experiments, we randomly sample transition and reward matrices, with an 8-dimensional state-space, 6-dimensional action-space, and horizon set to 20. As is standard in the literature, we use a linear Gaussian policy. For a

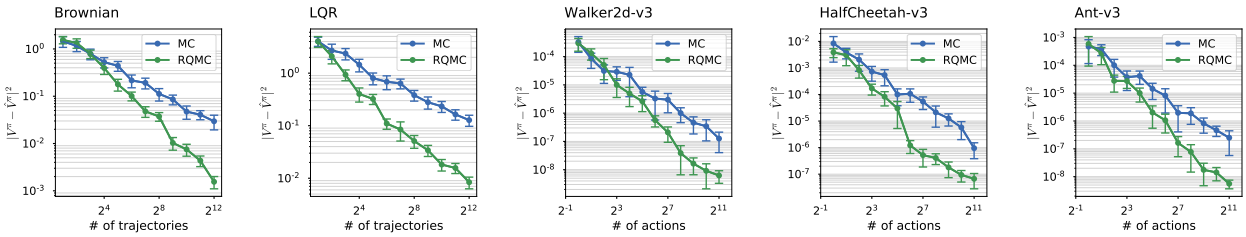

Figure 3: **RQMC reduces value estimation error.** RQMC is much more efficient than MC in estimating the value of a policy on a given number of samples. As suggested by the theory, the gap between RQMC and MC grows with the number of trajectories (in Brownian and LQR) or the number of sampled actions per state (in MuJoCo tasks, using the learned $\hat{Q}^{\pi_\theta}$). For MuJoCo tasks where ground-truth gradient is not available, we approximate it using 2^{16} actions.

more thorough treatment on LQR, we refer the reader to (Kwakernaak and Sivan, 1972).

We use the MuJoCo robotic environments available in the standardized OpenAI Gym suite (Brockman et al., 2016). For those experiments, we use a tanh-Gaussian policy whose mean and diagonal covariance are given by a 2-layer neural network. Unless specified otherwise, we use Adam (Kingma and Ba, 2015) as the optimization algorithm, share the learning rate among MC and RQMC, and set the horizon to 1000 timesteps. The main text presents results on `Walker2d-v3`, `HalfCheetah-v3`, and `Ant-v3`, with results on `Hopper-v3` and `Swimmer-v3` in the Supp. Material.

4.1 POLICY EVALUATION

Our first set of experiments compares the sample efficiency of MC and RQMC. To do so, we take a fixed policy (random policy in Brownian motion, optimal policy for LQR and trained policy for MuJoCo) and sample a predefined number of trajectories using either MC or RQMC. With those trajectories, we compute an estimated $\hat{V}^\pi$ of the expected return of the policy and compare it against the ground-truth policy value V^π. For the Brownian motion and LQR, the ground-truth is computed analytically and the estimation $\hat{V}^\pi$ is computed with Equation (7) with actions generated using MC or RQMC.

For the MuJoCo tasks, the ground-truth V^π is estimated on 2^{10} states for each of which we sample 2^{16} actions with MC. The estimated $\hat{V}^\pi = \mathbb{E}_s\left[\frac{1}{N}\sum_{i=1}^{N}\hat{Q}^{\pi_\theta}(s,\pi_\theta(s,u^{(i)}))\right]$ uses a $\hat{Q}^{\pi_\theta}$-function learned with SAC, and $u^{(i)}$ are sampled with either RQMC or MC. By varying the number of samples to estimate $\hat{V}^\pi$ we can observe the convergence rate of different integration methods. For each method, we repeat those experiments 30 times and report the mean squared error $|V^\pi - \hat{V}^\pi|^2$ in Figure 3.

For the Brownian motion, RQMC outperforms MC when using more than 128 trajectories. On LQR, RQMC outperforms MC with as little as 4 trajectories. Importantly, RQMC converges significantly faster than MC and ultimately reduces the evaluation error by more than an order of magnitude on both of these tasks.

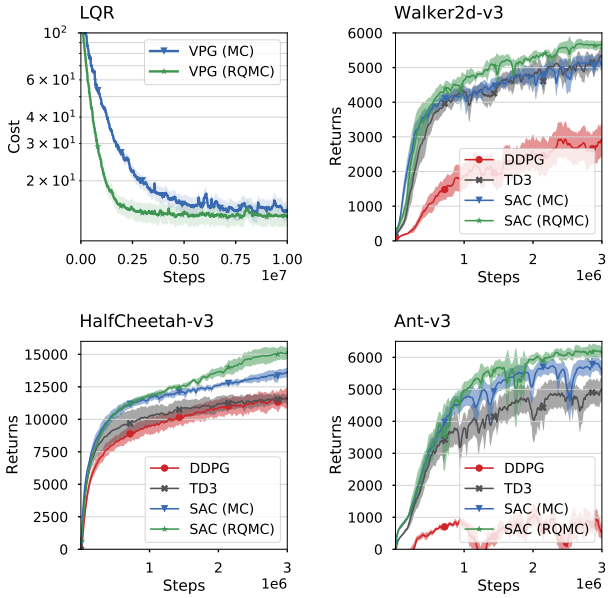

Figure 4: **RQMC improves policy learning.** Using RQMC for policy learning outperforms MC on LQR and Mujoco tasks. In particular, RQMC improves upon MC with SAC – a state-of-the-art actor-critic method – in terms asymptotic performance on all Mujoco tasks.

For MuJoCo, RQMC significantly outperforms MC when using as little as 16 actions per state. The gap between MC and RQMC continues to increase with more actions, and RQMC reaches over an order of magnitude lower estimation error with 2048 actions per states on all tasks.

4.2 POLICY IMPROVEMENT

To examine RQMC's effectiveness in learning a policy, we use the LQR and MuJoCo testbeds to optimize a ran-

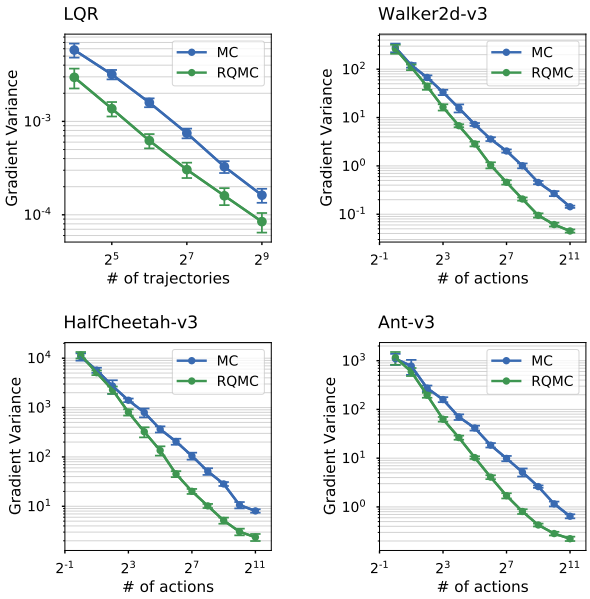

Figure 5: **RQMC reduces gradient variance.** On both LQR and MuJoCo tasks, RQMC achieves lower gradient variance than MC for the same number of trajectories. Here, variance refers to the trace of the gradient covariance matrix. The 95% confidence intervals are computed over 30 random seeds.

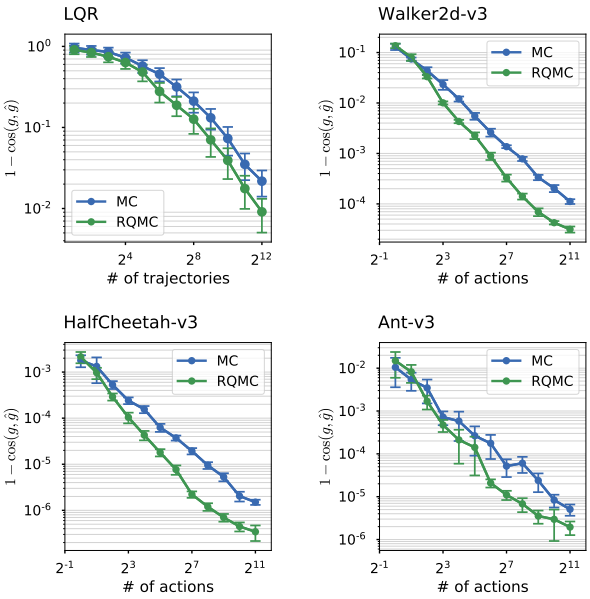

Figure 6: **RQMC improves gradient alignment.** For a given number of trajectories, the gradient direction computed with RQMC is better aligned than when computed with MC. The y-axis displays the angle between ground-truth and stochastic gradient. On LQR, the ground-truth is computed with $48k$ trajectories; on MuJoCo, it is estimated using 2^{16} actions. The 95% confidence intervals are computed over 30 random seeds.

domly initialized policy. On LQR we train with vanilla policy gradient (VPG) using Equation (8) for 10M environment interactions and compute the gradients using 16 trajectories drawn with either MC or RQMC. On MuJoCo, we train with SAC for 3M environment interactions and draw 8 actions per state using MC or RQMC during off-policy learning. For reference, we also include learning curves for DDPG (Lillicrap et al., 2016) and TD3 (Fujimoto et al., 2018). At test-time, we do not sample actions and instead use the mean of the policy. We use identical learning hyper-parameters for all methods (chosen to maximize convergence rate of MC); details are reported in the Supp. Material.

Figure 4 shows the mean convergence curves and standard errors computed over 15 different random seeds. On LQR, RQMC converges approximately 4 times faster than MC and also reaches lower asymptotic cost. On MuJoCo, RQMC outperforms MC in all cases and especially later in training where noisy gradients hinder exploitation. Notably, swapping MC for RQMC yields improvements comparable to those obtained when upgrading the learning algorithm from TD3 to SAC.

4.3 IMPROVED GRADIENT ESTIMATION

In this section, we verify the hypothesis that RQMC improves learning by reducing the variance of the stochastic gradient. To demonstrate this empirically, we take a trained policy and compare the variance and alignment of its gradient when estimated with MC versus RQMC. On LQR, we use the optimal policy obtained analytically and estimate gradients by replacing Q^{π_θ} with the sum of rewards in Equation (4); on MuJoCo, we train a policy until convergence with SAC and use Equation (10) to compute gradients. We define gradient variance be the trace of the covariance matrix $\mathbb{E}_{\hat{g}}\left[(\hat{g}-g)^\top(\hat{g}-g)\right]$, where $\hat{g}$ is the gradient estimated with a fixed number of trajectories (for LQR) or actions (for MuJoCo), and g is the ground-truth gradient computed with $48,000$ trajectories (for LQR) or 2^{16} actions (for MuJoCo). Similarly, we define gradient alignment $1-\cos(\hat{g},g)$, where $\cos(\hat{g},g)$ is the cosine similarity between estimated and ground-truth gradient. We compute 95% confidence intervals by repeating each measurement with 30 different random seeds.

Figure 5 reports the gradient variance for MC and RQMC. On LQR, we observe that RQMC always gives lower gradient variance regardless of the number of trajectories. On MuJoCo, we observe similar results where RQMC compares favorably with as little as 4 actions, and converges at a faster rate than MC.

We observe similar results for gradient alignment in Figure 6. On all settings, RQMC provides gradient

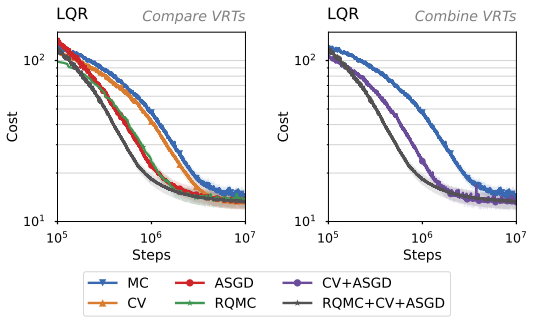

Figure 7: **RQMC outperforms but also complements other variance reduction techniques.** Left: On the LQR, RQMC matches or outperforms control variates (CV) and Accelerated SGD (ASGD; Jain et al. (2018)). Right: Combining those variance reduction techniques performs best, showing their complementarity.

estimates that converge faster and are better aligned with the ground-truth than MC.

4.4 COMBINATION WITH VARIANCE REDUCTION TECHNIQUES

In our last experiments, we compare RQMC against other variance reduction techniques and test whether it can complement those existing approaches. We focus on policy improvement with VPG on LQR since SAC already includes several advances that address issues related to variance reduction (*e.g.*, deterministic policy gradient, Adam, off-policy learning). We consider two alternative variance reduction techniques: control variates (CV) and variance-reduced optimizers. For control variates, we use the generalized advantage estimator (Schulman et al., 2016) and learn a linear baseline $b(s)$ to estimate the returns V^{π_θ} when starting from state s. This technique is ubiquitous in continuous control for actor-critic methods that do not directly learn an action-value function Q^{π_θ} (Mnih et al., 2016; Schulman et al., 2015). For variance-reduced optimizers, we compare SGD (with momentum (Polyak, 1964)) to Accelerated SGD (ASGD; Kidambi et al. (2018); Jain et al. (2018)), one of the first methods with provable accelerated asymptotic convergence with constant learning rate on noisy linear regression problems.

We compare MC, CV, ASGD, and RQMC in Figure 7 (left panel) and observe that all variance reduction techniques outperform MC, with ASGD and RQMC performing best. When combining those techniques Figure 7 (right panel), we see that adding CV and ASGD to MC (purple-dotted curve) performs on par with ASGD alone, suggesting that the benefits of CV are already captured by ASGD. On the other hand, adding CV and ASGD on top of RQMC (black-starred curve) accelerates convergence, thus demonstrating that RQMC complements these variance reduction techniques.

5 RELATED WORK

Randomized Quasi Monte-Carlo Methods Randomized Quasi-Monte Carlo (RQMC) methods are used throughout science and engineering to improve estimation of intractable integrals. Previous application domains of RQMC include finance (Glasserman, 2004; Joy et al., 1996; L'Ecuyer, 2009), computational biology (Beentjes and Baker, 2019; Puchhammer et al., 2021), and statistics and machine learning (Gerber and Chopin, 2015; Buchholz et al., 2018; Liu and Owen, 2021). From a theoretical perspective, L'Ecuyer et al. (2008) show that RQMC can reduce the variance in Markov chain settings and they propose *Array-RQMC*, an RQMC method taylored to Markov chains with long horizons (see the Supp. Material for preliminary experiments). Up to our knowledge, this work is the first that investigates RQMC in the context of reinforcement learning. For a more thorough review on the details of RQMC, we refer to (Niederreiter, 1992; L'Ecuyer, 2018).

Policy Gradient and Continuous Control Policy gradient methods (Williams, 1992; Sutton et al., 2000) are popular for continuous control problems as they naturally handle continuous action-space problems. A early applications of policy gradients to continuous control type problems (including LQR) can be found in Benbrahim and Franklin (1997) and Kimura (1998). Subsequently, notable applications include the control of a humanoid biped (Peters et al., 2003), dexterous manipulation (Rajeswaran et al., 2018), quadrupedal locomotion (Kohl and Stone, 2004), and simulated car driving from high-dimensional inputs (Wierstra et al., 2007). Peters and Schaal (2008) provide a review.

Specific to variance reduction, Weaver and Tao (2001b) and Greensmith et al. (2001b) derive bounds for optimal control variates (CV) on immediate and expected returns. Those optimal baselines are often hard to estimate, and more generally replaced with (approximate) state-value functions (Mnih et al., 2016; Schulman et al., 2016). Several works have proposed to further condition the CV on the actions (Grathwohl et al., 2018; Liu et al., 2018), but it is unclear if this those kinds of baselines are beneficial (Tucker et al., 2018). Romoff et al. (2018) show that learning a reward function in addition to a baseline effectively reduces gradient variance for environment with noisy rewards. From an optimization perspective, Papini et al. (2018) adopt a variance-reduced optimizer (SVRG; (Johnson and Zhang, 2013)) for policy improvement. Du et al. (2017) propose to also use SVRG to learn a state-value function, effectively tackling a policy evaluation problem. Peng et al. (2020) further extended this approach to

the mini-batch setting.

RQMC is orthogonal to all the above approaches and can be combined with them, see §4. Additionally, it has several additional appealing properties. First, the same approach (and code) equally handles policy evaluation and learning, unlike CV or optimization methods which require different design choices for both. Second, RQMC is simple to implement and does not require additional hyper-parameters. Finally, RQMC retains all appealing factors of policy gradient, remaining universally applicable.

6 CONCLUSION

We propose to replace the MC steps in policy evaluation and learning with Randomized Quasi-Monte Carlo sampling. This drop-in sampling technique is compatible with several existing state-of-the-art algorithms and has resulted in improved empirical results in continuous control problems. In particular, we observe reduced variances in policy value estimation as well as improved estimation of policy gradients, while reducing the number of samples required to estimate those quantities.

We foresee a few directions for future research. While our work is empirical in nature, a more formal characterization is needed to understand when RQMC is guaranteed to improve policy learning and evaluation. For example, it is well-known that RQMC can provably underperform MC on some (contrived) cases where smoothness requirements are not satisfied (Sloan and Woźniakowski, 1998). Do those cases also arise in reinforcement learning? Then, from a practical standpoint, our RQMC experiments took approximately 1.2x longer to run than MC ones — can we address this slow-down with specialized software? Finally, what role do specialized Markov chain methods (*e.g.*, Array-RQMC (L'Ecuyer et al., 2008)) play in reinforcement learning? We hope the promising results presented in this paper can help motivate those lines of inquiry.

Acknowledgements

We thank Florian Wenzel for stimulating discussion in the early stages of the project. We appreciate the feedback from the reviewers. This work is partially supported by NSF Awards IIS-1513966/ 1632803/1833137, CCF-1139148, DARPA Award#: FA8750-18-2-0117, FA8750-19-1-0504, DARPA-D3M - Award UCB-00009528, Google Research Awards, gifts from Facebook and Netflix, and ARO# W911NF-12-1-0241 and W911NF-15-1-0484.

References

Achiam, J. (2018). Spinning Up in Deep Reinforcement Learning.

Beentjes, C. H. L. and Baker, R. E. (2019). Quasi-Monte Carlo Methods Applied to Tau-Leaping in Stochastic Biological Systems. *Bull. Math. Biol.*, 81(8):2931–2959.

Benbrahim, H. and Franklin, J. A. (1997). Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems*, 22(3-4):283–302.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *ArXiv preprint*, abs/1606.01540.

Buchholz, A., Wenzel, F., and Mandt, S. (2018). Quasi-Monte Carlo Variational Inference. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 667–676. PMLR.

Defazio, A., Bach, F. R., and Lacoste-Julien, S. (2014). SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 1646–1654.

Du, S. S., Chen, J., Li, L., Xiao, L., and Zhou, D. (2017). Stochastic Variance Reduction Methods for Policy Evaluation. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1049–1058. PMLR.

Elvira, V. and Martino, L. (2021). Advances in Importance Sampling.

Fazel, M., Ge, R., Kakade, S. M., and Mesbahi, M. (2018). Global Convergence of Policy Gradient Methods for the Linear Quadratic Regulator. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1466–1475. PMLR.

Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., and Bachem, O. (2021). Brax - A Differentiable Physics Engine for Large Scale Rigid Body Simulation.

Frostig, R., Ge, R., Kakade, S. M., and Sidford, A. (2015). Competing with the Empirical Risk Minimizer in a Single Pass. In Grünwald, P., Hazan, E., and Kale, S., editors, *Proceedings of The 28th Conference on Learning Theory, COLT 2015, Paris, France, July 3-6, 2015*, volume 40 of *JMLR Workshop and Conference Proceedings*, pages 728–763. JMLR.org.

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic Methods. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1582–1591. PMLR.

Gal, Y. (2016). *Uncertainty in Deep Learning*. PhD thesis.

Gerber, M. and Chopin, N. (2015). Sequential Quasi Monte Carlo. *J. R. Stat. Soc. Series B Stat. Methodol.*, 77(3):509–579.

Glasserman, P. (2004). *Monte Carlo Methods in Financial Engineering*. Springer Science & Business Media.

Glynn, P. W. and Szechtman, R. (2002). Some New Perspectives on the Method of Control Variates. In *Monte Carlo and Quasi-Monte Carlo Methods 2000*, pages 27–49. Springer Berlin Heidelberg.

Grathwohl, W., Choi, D., Wu, Y., Roeder, G., and Duvenaud, D. (2018). Backpropagation through the Void: Optimizing control variates for black-box gradient estimation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Greensmith, E., Bartlett, P. L., and Baxter, J. (2001a). Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 1507–1514. MIT Press.

Greensmith, E., Bartlett, P. L., and Baxter, J. (2001b). Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 1507–1514. MIT Press.

Gu, S., Lillicrap, T., Turner, R. E., Ghahramani, Z., Schölkopf, B., and Levine, S. (2017). Interpolated Policy Gradient: Merging On-Policy and Off-Policy Gradient Estimation for Deep Reinforcement Learning. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3846–3855.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR.

Hickernell, F. J., Hong, H. S., L'Écuyer, P., and Lemieux, C. (2000). Extensible Lattice Sequences for Quasi-Monte Carlo Quadrature. *SIAM J. Sci. Comput.*, 22(3):1117–1138.

Jain, P., Kakade, S. M., Kidambi, R., Netrapalli, P., and Sidford, A. (2018). Accelerating Stochastic Gradient Descent for Least Squares Regression. In Bubeck, S., Perchet, V., and Rigollet, P., editors, *Conference On Learning Theory, COLT 2018, Stockholm, Sweden, 6-9 July 2018*, volume 75 of *Proceedings of Machine Learning Research*, pages 545–604. PMLR.

Joe, S. and Kuo, F. Y. (2008). Constructing sobol sequences with better Two-Dimensional projections. *SIAM J. Sci. Comput.*, 30(5):2635–2654.

Johnson, R. and Zhang, T. (2013). Accelerating Stochastic Gradient Descent using Predictive Variance Reduction. In Burges, C. J. C., Bottou, L., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 315–323.

Joy, C., Boyle, P., and Tan, K. (1996). Quasi-Monte Carlo Methods in Numerical Finance. *Management Science*, 42:926–938.

Kidambi, R., Netrapalli, P., Jain, P., and Kakade, S. M. (2018). On the insufficiency of existing momentum schemes for Stochastic Optimization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Kimura, H. (1998). Reinforcement learning for continuous action using stochastic gradient ascent. *Intelligent Autonomomus Systems*.

Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kohl, N. and Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 2619–2624. IEEE.

Kwakernaak, H. and Sivan, R. (1972). *Linear Optimal Control Systems*. John Wiley & Sons, Inc., USA.

Lawrence, G., Cowan, N., and Russell, S. (2002). Efficient Gradient Estimation for Motor Control Learning. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, UAI'03, pages 354–361, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Le Roux, N., Schmidt, M., and Bach, F. R. (2012). A Stochastic Gradient Method with an Exponential Convergence Rate for Finite Training Sets. In Bartlett, P. L., Pereira, F. C. N., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 2672–2680.

L'Ecuyer, P. (2009). Quasi-Monte Carlo Methods with Applications in Finance. *Finance and Stochastics*, 13(3):307–349.

L'Ecuyer, P. (2016). SSJ: Stochastic Simulation in Java, Software Library. `http://simul.iro.umontreal.ca/ssj/`.

L'Ecuyer, P. (2018). Randomized Quasi-Monte Carlo: An Introduction for Practitioners. In Glynn, P. W. and Owen, A. B., editors, *Monte Carlo and Quasi-Monte Carlo Methods: MCQMC 2016*, pages 29–52, Berlin. Springer.

L'Ecuyer, P., Lécot, C., and L'Archevêque-Gaudet, A. (2009). On array-RQMC for Markov chains: Mapping alternatives and convergence rates. In *Monte Carlo and Quasi-Monte Carlo Methods 2008*, pages 485–500. Springer Berlin Heidelberg, Berlin, Heidelberg.

L'Ecuyer, P., Lécot, C., and Tuffin, B. (2008). A Randomized Quasi-Monte Carlo Simulation Method for Markov Chains. *Oper. Res.*, 56(4):958–975.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Liu, H., Feng, Y., Mao, Y., Zhou, D., Peng, J., and Liu, Q. (2018). Action-dependent Control Variates for Policy Optimization via Stein Identity. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Liu, S. and Owen, A. B. (2021). Quasi-Monte Carlo Quasi-Newton in Variational Bayes. *J. Mach. Learn. Res.*, 22(243):1–23.

Makoviychuk, V., Wawrzyniak, L., Guo, Y., Lu, M., Storey, K., Macklin, M., Hoeller, D., Rudin, N., Allshire, A., Handa, A., and State, G. (2021). Isaac gym: High performance gpu-based physics simulation for robot learning. *CoRR*, abs/2108.10470.

Matoušek, J. (1998). On the L2-discrepancy for anchored boxes. *J. Complex.*, 14(4):527–556.

Metropolis, N. and Ulam, S. (1949). The Monte Carlo Method. *J. Am. Stat. Assoc.*, 44(247):335–341.

Miller, L. B. (1967). Monte Carlo analysis of reactivity coefficients in fast reactors general theory and applications. Technical Report ANL-7307, Argonne National Lab. (ANL), Argonne, IL (United States).

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. In Balcan, M. and Weinberger, K. Q., editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1928–1937. JMLR.org.

Mohamed, S., Rosca, M., Figurnov, M., and Mnih, A. (2019). Monte Carlo Gradient Estimation in Machine Learning. *J. Mach. Learn. Res.*, (132):1–62.

Niederreiter, H. (1978). Quasi-Monte Carlo methods and pseudo-random numbers. *Bull. Am. Math. Soc.*, 84(6):957–1041.

Niederreiter, H. (1992). Random Number Generation and Quasi-Monte Carlo Methods. pages 23–45. Society for Industrial and Applied Mathematics.

Owen, A. B. (1997). Scrambled net variance for integrals of smooth functions. *The Annals of Statistics*, 25(4):1541–1562.

Owen, A. B. (1998). Scrambling Sobol' and Niederreiter–Xing Points. *J. Complex.*, 14(4):466–489.

Papageorgiou, A. (2003). Sufficient conditions for fast quasi-Monte Carlo convergence. *J. Complex.*, 19(3):332–351.

Papini, M., Binaghi, D., Canonaco, G., Pirotta, M., and Restelli, M. (2018). Stochastic Variance-Reduced Policy Gradient. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4023–4032. PMLR.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Peng, Z., Touati, A., Vincent, P., and Precup, D. (2020). SVRG for Policy Evaluation with Fewer Gradient Evaluations. In Bessiere, C., editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 2697–2703. ijcai.org.

Peters, J. and Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697.

Peters, J., Vijayakumar, S., and Schaal, S. (2003). Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pages 1–20.

Petrenko, A., Huang, Z., Kumar, T., Sukhatme, G., and Koltun, V. (2020). Sample Factory: Egocentric 3D Control from Pixels at 100000 FPS with Asynchronous Reinforcement Learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 7652–7662. PMLR.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

Puchhammer, F., Ben Abdellah, A., and L'Ecuyer, P. (2021). Variance Reduction with Array-RQMC for Tau-Leaping Simulation of Stochastic Biological and Chemical Reaction Networks. *Bull. Math. Biol.*, 83(8):91.

Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. (2018). Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*.

Recht, B. (2019). A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2:253–279.

Robert, C. P. and Casella, G. (2004). Monte Carlo Statistical Methods. In *Springer Texts in Statistics*.

Romoff, J., Henderson, P., Piche, A., Francois-Lavet, V., and Pineau, J. (2018). Reward Estimation for Variance Reduction in Deep Reinforcement Learning. In Billard, A., Dragan, A., Peters, J., and Morimoto, J., editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 674–699. PMLR.

Rubinstein, R. Y. (1969). *Some problems in Monte Carlo optimization*. PhD thesis, Riga Polytechnical Institute.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. (2015). Trust Region Policy Optimization. In Bach, F. R. and Blei, D. M., editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1889–1897. JMLR.org.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. (2016). High-Dimensional Continuous Control Using Generalized Advantage Estimation. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Shalev-Shwartz, S. and Zhang, T. (2013). Stochastic dual coordinate ascent methods for regularized loss minimization. *J. Mach. Learn. Res.*

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. A. (2014). Deterministic Policy Gradient Algorithms. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 387–395. JMLR.org.

Sloan, I. H. and Woźniakowski, H. (1998). When are Quasi-Monte carlo algorithms efficient for high dimensional integrals? *J. Complex.*, 14(1):1–33.

Sobol, I. M. (1976). Uniformly distributed sequences with an additional uniform property. *USSR Computational Mathematics and Mathematical Physics*, 16(5):236–242.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.

Tucker, G., Bhupatiraju, S., Gu, S., Turner, R. E., Ghahramani, Z., and Levine, S. (2018). The Mirage of Action-Dependent Baselines in Reinforcement Learning. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5022–5031. PMLR.

Weaver, L. and Tao, N. (2001a). The Optimal Reward Baseline for Gradient-Based Reinforcement Learning. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, page 538–545, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Weaver, L. and Tao, N. (2001b). The Optimal Reward Baseline for Gradient-Based Reinforcement Learning. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, pages 538–545, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Wierstra, D., Foerster, A., Peters, J., and Schmidhuber, J. (2007). Solving deep memory POMDPs with recurrent policy gradients. In *International Conference on Artificial Neural Networks*, pages 697–706. Springer.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

A EXPERIMENTAL DETAILS

This section provides additional details on the tasks, algorithms, and implementations in our experimental setups.

Our code (and other resources) is available at: `http://seba1511.net/projects/qrl/`

A.1 MDP BACKGROUND

All the tasks considered in our experimental section can be modelled as a Markov decision process (MDP). A MDP can be formalized as a 4-tuple $(\mathcal{S}, \mathcal{A}, T, R)$, where $\mathcal{S}$ and $\mathcal{A}$ are the spaces of states and actions, respectively. The transition distribution $T(s' \mid s, a)$ indicates the probability of transitioning from state $s \in \mathcal{S}$ to next state $s' \in \mathcal{S}$ when taking action $a \in \mathcal{A}$. In case the transition dynamics are deterministic (as in some MuJoCo tasks), T degenerates to a Kronecker delta function. Finally, the reward function $R(s, a)$ assigns a reward scalar $r \in \mathbb{R}$ to each state-action pair (s, a), and defines the objective the agent is tasked to optimize. As we consider finite horizon tasks, we omit possible discount factors and assume they are implicitly absorbed into the reward function.

A.2 BROWNIAN MOTION

For the Brownian motion experiments, the agent is a point-mass with position $s \in \mathbb{R}$ and takes actions $a \in \mathbb{R}$. Given current state s and action a, the next state is deterministically computed with $s' = s + 0.1 \cdot a$ where $a \sim \mathcal{N}(\mu, \sigma)$ for state-independent parameters $\mu \in \mathbb{R}$ and $\sigma \in \mathbb{R}$, and the immediate reward is $R(s, a) = \|s'\|_2$. The initial state is always $s_1 = 0$ and the episode terminates after $T = 20$ timesteps.

A.3 LINEAR-QUADRATIC REGULATOR

In LQR experiments, we use a Gaussian policy $\pi_K(a_t \mid s_t) = \mathcal{N}(Ks_t, I_6)$ where $K \in \mathbb{R}^{6\times 8}$ is a learnable matrix and I_6 is the identity matrix in $\mathbb{R}^6$. Given initial state $s_1 = \frac{\epsilon_0}{\|\epsilon_0\|_2}$ with $\epsilon_0 \sim \mathcal{N}(0, I_8)$, states and immediate rewards at timestep $t \in [1, \ldots, 20]$ are computed with

$$s_{t+1} = As_t + Ba_t + \epsilon_t \qquad \text{where} \quad \epsilon_t \sim \mathcal{N}(0, \Sigma_s) \tag{11}$$

$$R(s_t, a_t) = -s_t^\top P s_t - a_t^\top Q a_t, \tag{12}$$

where $A \in \mathbb{R}^{8\times 8}$ and $B \in \mathbb{R}^{8\times 6}$ are constructed by first sampling entries from a standard normal distribution $\mathcal{N}(0, 1)$ and then normalizing the matrices to have unit Frobenius norm. Matrices $P \in \mathbb{R}^{8\times 8}$ and $Q \in \mathbb{R}^{6\times 6}$ are random positive semi-definite and constructed to have unit condition number. The state covariance matrix Σ_s is also random positive semi-definite with unit condition number.

A.4 MUJOCO

We use the standardized MuJoCo tasks (`Swimmer-v3`, `HalfCheetah-v3`, `Hopper-v3`, `Walker2d-v3`, and `Ant-v3`) as described by Brockman et al. (2016). Specifically, we use the implementations in `gym` version `0.20.0`, `mujoco-py` version `1.50.1.68`, and `MuJoCo` version `1.50`.

A.5 METHODS DETAILS

This section provides more details on learning algorithms. As the VPG implementation for Brownian motion and LQR tasks closely follows the presentation in the main text, we focus on our implementation of SAC.

The two main components of SAC are the loss functions for $\hat{Q}^{\pi_\theta}$ and π_θ augmented with a maximum entropy objective. To learn $\hat{Q}^{\pi_\theta}$, SAC minimizes the on-policy squared Bellman error

$$\mathcal{L}_{\hat{Q}^{\pi_\theta}} = \mathop{\mathbb{E}}_{s_t, a_t, s_{t+1}} \left[\left(\perp \left(\mathop{\mathbb{E}}_{a_{t+1}} \left[\hat{Q}^{\pi_\theta}(s_{t+1}, a_{t+1}) - \alpha \log \pi_\theta(a_{t+1} | s_{t+1}) \right] + R(s_t, a_t) \right) - \hat{Q}^{\pi_\theta}(s_t, a_t) \right)^2 \right], \tag{13}$$

Table 1: Hyper-parameters for LQR tasks.

Hyper-parameter	Value
Learning Rate	0.0007
Momentum	0.99
Trajectories / Updates	16
CV's γ (discount)	0.99
CV's λ (GAE interpolation)	0.95
ASGD's κ (long to short step ratio)	1000.0
ASGD's ξ (statistical advantage)	10.0

Table 2: Hyper-parameters for MuJoCo tasks.

Hyper-parameter	Value
Learning Rate	0.001
Entropy Regularization	0.2
Mini-batch Size	100
Minimum Replay Size	4000
Maximum Replay Size	10^6
MLP Depth (π_θ & $\hat{Q}^{\pi_\theta}$)	2
MLP Width (π_θ & $\hat{Q}^{\pi_\theta}$)	256

where $\perp(\cdot)$ is the stop-gradient operator, and $\alpha \in \mathbb{R}$ is the weight of the entropy bonus encouraging exploration. In the above equation, s_t, a_t, s_{t+1} are sampled from a replay buffer of past experience while a_{t+1} is freshly sampled for each evaluation of $\mathcal{L}_{\hat{Q}^{\pi_\theta}}$. While most implementations use a single action to estimate the inner expectation $\mathbb{E}_{a_{t+1}}$ over next actions a', we found 8 actions to work as well or better with MC and performed significantly better with RQMC.

To optimize the policy π_θ, SAC maximizes the expected returns as approximated by $\hat{Q}^{\pi_\theta}$ (Equation (8)). The policy loss augmented with a max-entropy term is given by

$$\mathcal{L}_{\pi_\theta} = -\mathop{\mathbb{E}}_{s}\left[\mathop{\mathbb{E}}_{a}\left[\hat{Q}^{\pi_\theta}(s,a) - \alpha \log \pi_\theta(a \mid s)\right]\right], \tag{14}$$

where s is sampled from a replay buffer, and a from the current policy π_θ. In this case too, we found it beneficial to use 8 actions for the expectation over actions $\mathbb{E}_a$ for a given state s.

For more details, please refer to the provided code implementation.

A.6 LEARNING HYPER-PARAMETERS

LQR We report hyper-parameters for the LQR experiments (including those combining other variance reduction techniques) in Table 1.

MuJoCo We use the same hyper-parameters for all tasks and algorithms (SAC, TD3, and DDPG) on MuJoCo tasks, given in Table 2.

A.7 CODE SNIPPETS

This section describes how to implement RQMC with popular software packages. For the Brownian motion and LQR experiments, we use the left matrix scramble and digital shift implementation in SSJ (L'Ecuyer, 2016), and for the MuJoCo tasks we use Owen's scrambling as implemented in PyTorch (Paszke et al., 2019). Pseudocodes are listed in Algorithm 2 and Algorithm 3. A complete implementation of SAC with RQMC (built on `Spinning-Up` (Achiam, 2018)) is included with the supplementary material.

Algorithm 2 Left Matrix Scramble and Digital Shift in SSJ (L'Ecuyer, 2016)

```
public double[][] sample(int pow, int dim) {
    MRG32k3a stream = new MRG32k3a();
    int n_samples = (int)Math.pow(2, pow);
    double[][] pointset = new double[n_samples][dim];

    DigitalNetBase2 p = new SobolSequence(pow, 31, dim);
    p.leftMatrixScramble(stream);
    p.addRandomShift(stream);

    PointSetIterator point_stream = p.iterator ();
    for (int i = 0; i < n_samples; ++i) {
        point_stream.nextPoint(pointset[i], dim);
    }
    return pointset; // Use pointset as in l. 4 - 10 of Algorithm 1
}
```

Algorithm 3 RQMC Policy Evaluation w/ Critic in PyTorch (Paszke et al., 2019)

```
from torch.quasirandom import SobolEngine
rqmc = SobolEngine(dim(𝒜), scrambled=True)
R = 0.0
for k in 1,...,M:  # evaluation on M states
    s_k = replay_buffer.states[k]
    rqmc._scramble()
    rqmc.reset()
    u^(k) = rqmc.draw(N)   # u^(k) ∈ ℝ^{N×dim(𝒜)}
    for j in 1,...,N:  # N actions per state
        a_j^(k) = μ_θ(s_k) + σ_θ(s_k) ⊙ F^{-1}(u_j^(k))
        R += Q^{π_θ}(s_k, a_j^(k))
return R / (M · N)
```

B ADDITIONAL EXPERIMENTAL RESULTS

This section presents experiments that supplement the ones from the main text. In particular, it includes results for all MuJoCo tasks (`Swimmer-v3`, `HalfCheetah-v3`, `Hopper-v3`, `Walker2d-v3`, `Ant-v3`), and an extension of RQMC (*Array*-RQMC) specifically designed for Markov chains.

B.1 POLICY EVALUATION

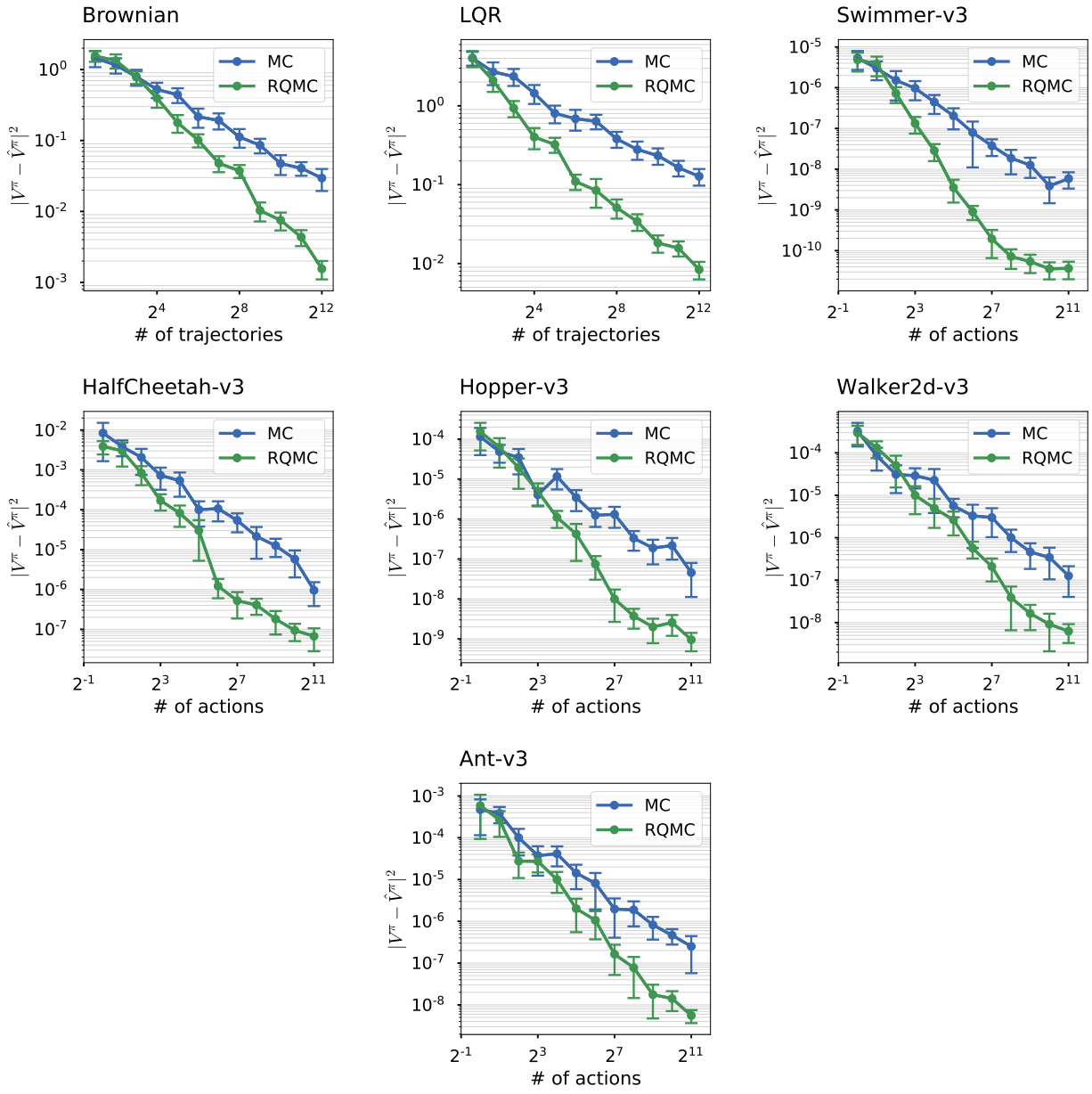

Figure 8: **RQMC reduces value estimator error.** RQMC is much more efficient than MC in estimating the value of a policy on a given number of trajectories. As suggested by theory, the gap between RQMC and MC grows with the number of trajectories. For Mujoco tasks where ground-truth gradient is not available, we approximate it using 2^{16} actions.

Figure 8 reports value estimation error akin to Figure 3 of the main text. On the two new MuJoCo tasks, we observe again that:

1. RQMC significantly reduces value estimation error (by up to 2 orders of magnitude on `Swimmer-v3`), even with relatively few (16) trajectories or actions (128 on `Walker2d-v3`, 16 on others).
2. RQMC converges more rapidly than MC in value estimation error.

B.2 POLICY IMPROVEMENT

Figure 9 complements Figure 4 from the main text, with policy improvement curves for `Swimmer-v3` and `Hopper-v3`. We observe:

1. RQMC improves the performance of SAC compared to MC on both `Swimmer-v3` and `Hopper-v3`. Typically, this improvement is on the same order as upgrading the learning algorithm from TD3 to SAC.
2. RQMC is no panacea: when SAC (MC) fails, adding RQMC does not perform much better. This is seen on `Swimmer-v3`, where simpler methods (DDPG and TD3) drastically outperform SAC with MC or RQMC.

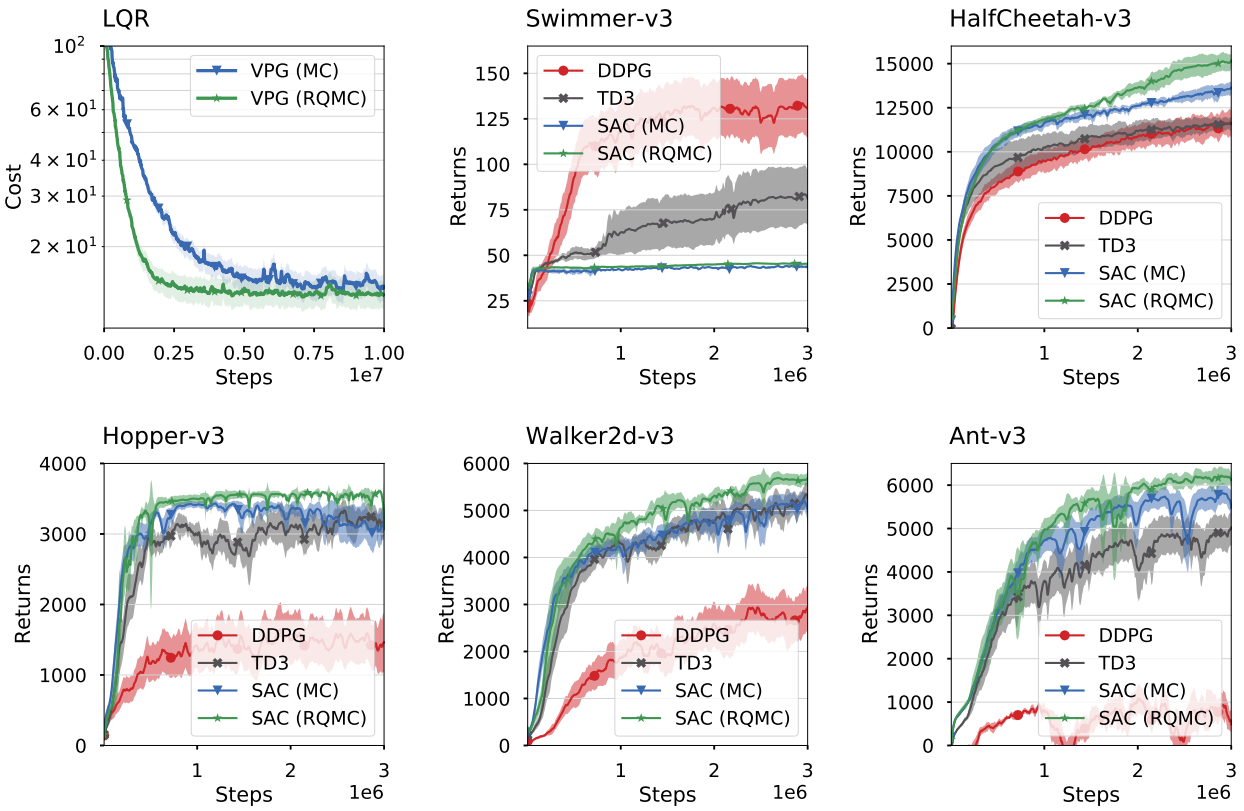

Figure 9: **RQMC improves policy learning.** Using RQMC during policy learning outperforms MC on LQR and Mujoco tasks. In particular, RQMC improves upon MC with SAC – a state-of-the-art actor-critic method – in terms asymptotic performance on all Mujoco tasks. Unsurprisingly, RQMC alone does not suffice to fix SAC's poor performance on `Swimmer-v3` where it is surpassed by simpler methods (*i.e.*, DDPG, TD3).

B.3 IMPROVED GRADIENT ESTIMATION

In this subsection, we complete the gradient variance and alignement results of Figures 5 and 6 with `Swimmer-v3` and `Hopper-v3` in Figures 10 and 11. On both sets of figures, we observe similar trends as in the main text: both gradient variance and alignment improve with more samples (trajectories or actions), and they typically improve at a faster rate with RQMC.

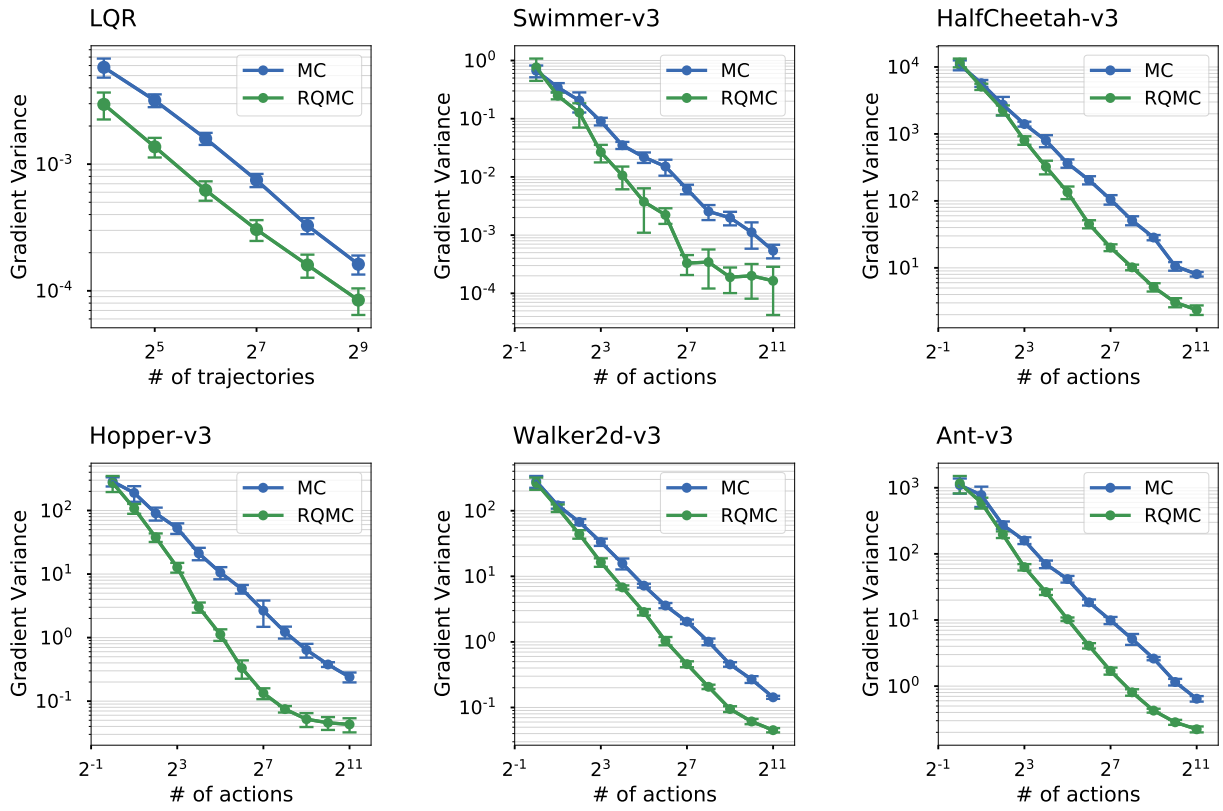

Figure 10: **RQMC reduces gradient variance.** On both LQR and MuJoCo tasks, RQMC achieves lower gradient variance than MC for the same number of trajectories. Here, variance refers to the trace of the gradient covariance matrix. The 95% confidence intervals are computed over 30 random seeds.

B.4 ROBUSTNESS TO INCREASED STATE TRANSITION NOISE

We now verify the robustness of RQMC under varying state transition noise. We repeat the VPG experiments on the LQR setting while varying the scaling of the state transition covariance noise from 0.1 to 0.8. Figure 12 reports these results. As expected, increasing dynamics noise increases asymptotic cost for both MC and RQMC, but RQMC retains its advantage over MC in terms of convergence rate.

B.5 EXTENSION TO MARKOV CHAINS WITH ARRAY-RQMC

We experiment with *Array*-RQMC (L'Ecuyer et al., 2008, 2009), a formulation of RQMC specifically designed for Markov chains. *Array*-RQMC aims to overcome the dependency on the dimension when rolling out a policy for long horizons T. As underlined in the main text, those challenges are twofold: the advantages of RQMC diminish with higher integration dimension, and popular implementations don't support dimensions higher than 21,201.

To address these issues, *Array*-RQMC assumes that M trajectories can be collected in parallel, which is almost always the case with simulated environments (Petrenko et al., 2020; Freeman et al., 2021; Makoviychuk et al., 2021). Then, *Array*-RQMC samples a new RQMC point set $u_1^{(t)}, \dots, u_M^{(t)}$ at every timestep t, with one point per trajectory. Each of the M points $u_m^{(t)} \in \mathbb{R}^{|\mathcal{A}|}$ has dimensionality $|\mathcal{A}|$ thus dropping the dependency on T. In order to effectively reduce variance, *Array*-RQMC has to carefully assign the current state of each trajectory with points in the RQMC point set. This assignment is done by sorting all states at timestep t according to an application-dependent value function. Intuitively, the goal of this sorting function is to improve approximation of the state distribution at timestep t such that, *e.g.*, the first point of the point set is always assigned to the state with lowest state value. For more detailed treatments of *Array*-RQMC, we refer the reader to L'Ecuyer (2018) and Puchhammer et al. (2021).

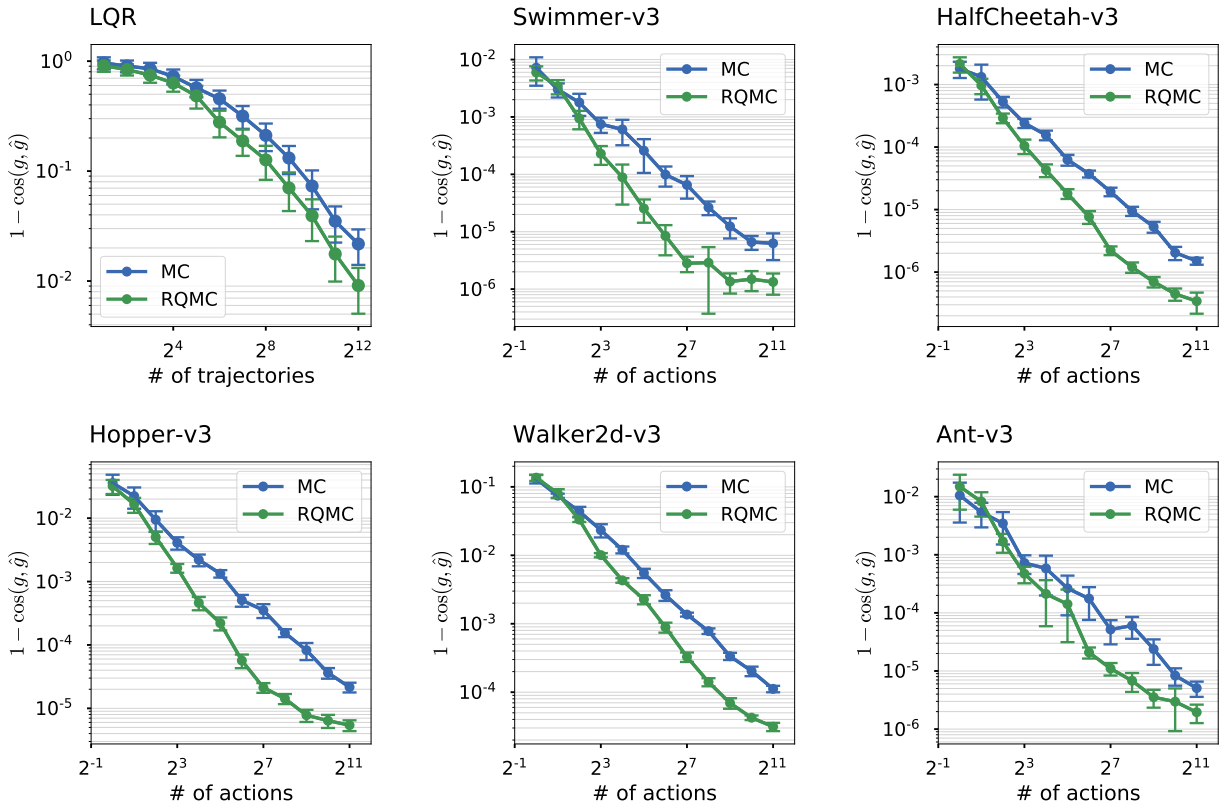

Figure 11: **RQMC improves gradient alignment.** For a given number of trajectories, the gradient direction computed with RQMC is better aligned than when computed with MC. The y-axis displays the angle between ground-truth and stochastic gradient. On LQR, the ground-truth is computed with $48k$ trajectories; on MuJoCo, it is estimated using 2^{16} actions. The 95% confidence intervals are computed over 30 random seeds.

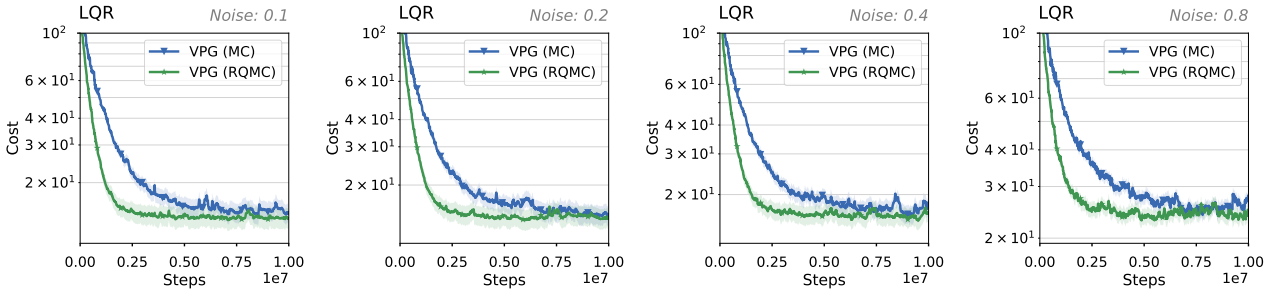

Figure 12: **RQMC outperforms MC under various noise settings.** In the LQR setting, we vary the scaling of the state transition covariance noise from 0.1 to 0.8. Both MC and RQMC perform worse (higher cost) with more noise, but RQMC retains a faster convergence rate than MC on all settings.

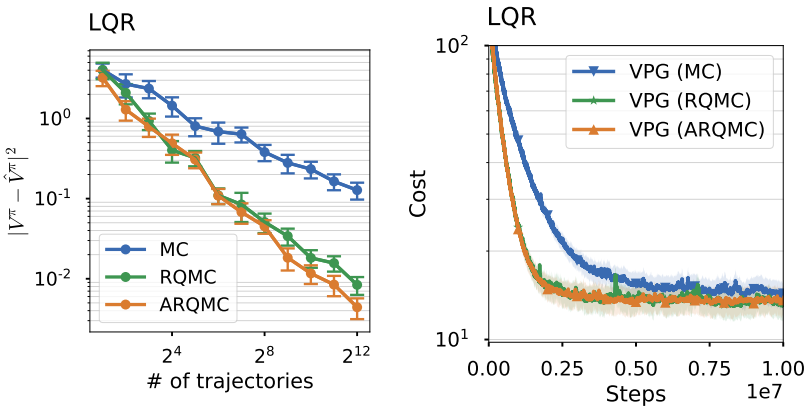

Figure 13: **Benchmarking RQMC extensions to Markov chains.** *Array*-RQMC (ARQMC), an RQMC formulation for Markov chains, further improves upon RQMC on the LQR for policy evaluation (left) while closely matching RQMC in learning performance (right).

We compare *Array*-RQMC (ARQMC) against MC and RQMC on the LQR in Figure 13. Throughout, we assign RQMC points to states according the state's ℓ_1-norm, as it is a reasonable proxy for the state's true value and outperformed the ℓ_2 and ℓ_∞ norms in practice. For policy evaluation (left panel), ARQMC improves upon RQMC, reaching approximately 2.5x lower value estimation error with $4,096$ trajectories. For policy learning (right panel), ARQMC closely closely matches RQMC but does not provide additional benefit.

We hope those experiments can help motivate research on *Array*-RQMC in the context of reinforcement learning.