Harnessing the Power of Reinforcement Learning for Adaptive MCMC

Congye Wang Newcastle University
Matthew A. Fisher Newcastle University
Heishiro Kanagawa Newcastle University
Wilson Chen University of Sydney
Chris. J. Oates Newcastle University

Abstract

Sampling algorithms drive probabilistic machine learning, and recent years have seen an explosion in the diversity of tools for this task. However, the increasing sophistication of sampling algorithms is correlated with an increase in the tuning burden. There is now a greater need than ever to treat the tuning of samplers as a learning task in its own right. In a conceptual breakthrough, Wang et al. (2025) formulated Metropolis–Hastings as a Markov decision process, opening up the possibility for adaptive tuning using Reinforcement Learning (RL). Their emphasis was on theoretical foundations; realising the practical benefit of Reinforcement Learning Metropolis–Hastings (RLMH) was left for subsequent work. The purpose of this paper is twofold: First, we observe the surprising result that natural choices of reward, such as the acceptance rate, or the expected squared jump distance, provide insufficient signal for training RLMH. Instead, we propose a novel reward based on the contrastive divergence, whose superior performance in the context of RLMH is demonstrated. Second, we explore the potential of RLMH and present adaptive gradient-based samplers that balance flexibility of the Markov transition kernel with learnability of the associated RL task. A comprehensive simulation study using the `posteriordb` benchmark supports the practical effectiveness of RLMH.

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300.

1 INTRODUCTION

Probabilistic machine learning leverages probability theory as the basis for making inferences and predictions in a machine learning task. Although only a subset of machine learning, the probabilistic approach is particularly useful in applications where it is important to quantify uncertainty, handle incomplete data, or incorporate prior knowledge into a model. At the core of probability theory is the concept of conditioning, and we are typically interested in the distribution of unobserved quantities conditional on an observed training dataset. Such conditional distributions are characterised by Bayes' theorem, and are called *posterior* distributions in the Bayesian context. However, the lack of a closed form for the normalisation constant in Bayes' theorem necessitates numerical approximation in general.

Since the early development of posterior sampling methodology, such as Markov chain Monte Carlo (MCMC), the scope and ambition of numerical methods for posterior approximation has expanded to include techniques such as variational approximations (Fox and Roberts, 2012), sequential Monte Carlo (Chopin and Papaspiliopoulos, 2020), gradient flows (Chen et al., 2023), normalising flows (Rezende and Mohamed, 2015), and more recently samplers based on diffusion models (Blessing et al., 2025). As a result, one can argue that the main research challenge is no longer the design of new methods for posterior approximation, but rather the *tuning* of existing methods to perform well on a given task. Unfortunately, the tuning problem is difficult; despite half a century of research, the design and use of convergence diagnostics for tuning MCMC remains an active research topic. Further, the increasing sophistication of emerging posterior approximation methods is usually associated with increasing difficulty of the associated tuning task. These observations lead us to contend that there is now a greater need than ever to treat the tuning of samplers as a learning task in its own right.

The idea in this manuscript is to harness the po-

tential of Reinforcement Learning (RL) to provide a powerful and principled solution to the tuning of MCMC. Our inspiration comes from Wang et al. (2025), who demonstrated how Metropolis–Hastings samplers can be formulated as Markov decision processes, and are therefore amenable to tuning using modern RL. That framework was dubbed *Reinforcement Learning Metropolis–Hastings* (RLMH), and its correctness (i.e. ergodicity) was established using appropriately modified techniques from the literature on adaptive MCMC. However, actually realising the practical benefit of RLMH was left for subsequent work. The purpose of this paper is twofold: First, we observe the surprising result that natural choices of reward, such as the acceptance rate, or the expected squared jump distance, provide insufficient signal for training RLMH. Instead, we propose a novel reward based on the contrastive divergence, whose superior performance in the context of RLMH is demonstrated. Second, we explore the potential of RLMH and present adaptive gradient-based samplers that balance flexibility of the Markov transition kernel with learnability of the associated RL task. A comprehensive simulation study using the `PosteriorDB` benchmark supports the effectiveness of RLMH.

1.1 Related Work

Throughout we consider a posterior distribution P with positive and differentiable density $p(\cdot)$ supported on $\mathbb{R}^d$, but in principle much of our discussion can be generalised with additional technical effort.

Gradient-Based MCMC There are an enormous diversity of MCMC algorithms in the literature, but for high-dimensional posterior distributions it is usually essential to have access to gradients of the target. The most common mechanism through which gradient information is exploited is through the *overdamped Langevin diffusion*

$$\mathrm{d}X_t = (\nabla \log p)(X_t)\mathrm{d}t + \sqrt{2}\mathrm{d}W_t \tag{1}$$

which, under regularity conditions, defines a stochastic process $(X_t)_{t\geq 0}$ whose law converges to P in the $t \to \infty$ limit (see e.g. Eberle, 2016). Since it is usually not possible to exactly simulate (1), one can consider a (biased) Euler–Maruyama discretisation

$$X^*_{n+1} = X_n + \epsilon(\nabla \log p)(X_n) + \sqrt{2\epsilon}Z_n \qquad Z_n \overset{\text{iid}}{\sim} N(0,1), \tag{2}$$

as a proposal, which can then be either accepted or rejected within the Metropolis–Hastings framework to ensure convergence to the correct target; this is known as the Metropolis-adjusted Langevin algorithm (MALA) (Roberts and Tweedie, 1996). Selection of the *step size* $\epsilon > 0$ is critical; for ϵ too small, many iterations will be required, while for ϵ too large the discretisation error in (2) will be substantial and the proposed states will be rejected. Unfortunately, the range of reasonable values for ϵ is often extremely small (Livingstone and Zanella, 2022). This observation motivated the development of Riemannian MALA (RMALA),

$$X^*_{n+1} = X_n + G(X_n)^{-1}(\nabla \log p)(X_n) + \sqrt{2}G(X_n)^{-\frac{1}{2}}Z_n, \tag{3}$$

where $G(X_n)$ acts as a position-dependent preconditioning matrix. Some possible choices for $G(\cdot)$ based on second derivatives of the target were proposed in Girolami and Calderhead (2011), and sufficient conditions for ergodicity were established in Roy and Zhang (2023). Second derivative information enables the step size to be scaled in a manner commensurate with the curvature of the target, and RMALA can be an effective sampling method. Unfortunately, the computational cost associated with obtaining higher order derivatives is often substantial[1], and as a result RMALA is not widely used. Compromises include taking $G(\cdot) = \epsilon^{-1}G_0$ to be an appropriately scaled constant matrix (Roberts and Stramer, 2002), or $G(\cdot) = \epsilon^{-1}D_0$ where D_0 is a diagonal matrix and the state space of the Markov chain is augmented to include ϵ as a variable that takes values in a discrete set (Biron-Lattes et al., 2024; Liu et al., 2024).

Adaptive MCMC Due to the challenge of tuning MCMC, on-line (or *adaptive*) tuning methods have considerable practical appeal (Haario et al., 2006; Roberts and Rosenthal, 2009). Several creative solutions have been proposed, for instance synthesising ideas from Bayesian optimisation (Mahendran et al., 2012), divergence minimisation (Coullon et al., 2023; Kim et al., 2025), and generative diffusion models (Hunt-Smith et al., 2024). However, altering the Markov chain transition probabilities based on the current sample path breaks the Markov dependency structure and ergodicity can be lost. Theoreticians have sought to discover sufficient criteria under which adaptive MCMC methods can asymptotically sample from the correct target (Atchadé and Rosenthal, 2005;

[1] For instance, in the setting of parameter inference for ordinary differential equations (ODEs), to obtain first order gradients the ODE system can be augmented with an additional $O(d)$ variables, called *sensitivities*, and this augmented system of ODEs can be integrated forward. For second order derivatives, an additional $O(d^2)$ variables must be included. Unfortunately numerical integration of large systems of ODEs can be a challenging task, and fine time steps can be required to properly resolve the dynamics of all the additional variables that are included.

Andrieu and Moulines, 2006; Atchade et al., 2011). The most well-known of these are *containment* and *diminishing adaptation*; the first ensures the Markov chain transition probabilities do not become degenerate, and the second ensures the transition probabilities converge to a limit (Roberts and Rosenthal, 2007). Focussing on Metropolis–Hastings, adaptive algorithms attempt to optimise an explicit performance criterion by iteratively refining parameters of the proposal. Several criteria are available, but the most common[2] choices are the average acceptance rate (AAR) and the expected squared jump distance (ESJD) (Pasarica and Gelman, 2010). Theoretical analysis provides guidance on which AAR to target[3], but since this is limited to idealised settings its practical relevance is debated (Potter and Swendsen, 2015). For ESJD, higher values are preferred, but a high ESJD does not itself guarantee that the posterior is well-approximated (c.f. Section 2.2). For a relatively inflexible parametric proposal, these existing criteria can often assist in the identification of suitable parameter values, but for more flexible classes of proposal the information provided by the AAR and ESJD can be insufficient to identify a suitable proposal.

RL Meets Adaptive MCMC Online decision problems with a Markovian structure are the focus of RL; it therefore appears natural to consider using RL for adaptive MCMC. Recall that a Markov decision process (MDP) consists of a *state* set $\mathcal{S}$, an *action* set $\mathcal{A}$, and an *environment* which determines both how the state s_n is updated to s_{n+1} in response to an action a_n, and the (scalar) *reward* r_n that resulted. RL refers to a broad class of methods that attempt to learn a *policy*[4] $\pi : \mathcal{S} \to \mathcal{A}$, meaning a mechanism to select actions $a_n = \pi(s_n)$, which aims to maximise an expected cumulative (discounted) reward (Kaelbling et al., 1996). A natural strategy is to define the state s_t of the MDP to be the state X_t of the Markov chain, and the action to be parameters of the Markov transition kernel. This approach was called *Reinforcement Learning Accelerated MCMC* in Chung et al. (2023), where a finite set of Markov transition kernels were developed for a particular multi-scale inverse problem, and called *Reinforcement Learning-Aided MCMC* in Wang et al. (2021), where RL was used to learn frequencies for updating the blocks of a Gibbs sampler. The success of these methods relied on limiting the number of transition kernels that were considered; for a flexible class of transition kernels the action would become high-dimensional, a regime to which RL is not well-suited. A conceptual breakthrough occurred in Wang et al. (2025), where it was shown how Metropolis–Hastings algorithms can be cast as MDPs via an augmented state $s_n = [X_n, X^*_{n+1}]$, consisting of the current and proposed state of the Markov chain, and an action $a_n = [\phi(X_n), \phi(X^*_{n+1})]$. The augmented state is necessary because the Metropolis–Hastings acceptance probability requires both the parameters $\phi(X_n)$ of the forward proposal and the parameters $\phi(X^*_{n+1})$ of the reverse proposals to be provided within the MDP. The key insight of RLMH, in contrast to earlier work, is to parametrise the (local) proposal distribution instead of the (global) transition kernel, simultaneously enabling flexible transition kernels while keeping actions low-dimensional. The authors of that work focussed on theoretical analysis, establishing sufficient conditions on RLMH for containment and diminishing adaptation to hold, while their methodological development was limited to a gradient-free random walk Metropolis–Hastings algorithm in which X^*_{n+1} was sampled from a distribution with mean $\phi(X_n)$ and the function $\phi(\cdot)$ was learned using RL. Realising the full potential of RLMH, including for gradient-based algorithms, was left for future work.

1.2 Outline of the Paper

The aim of this paper is to develop efficient gradient-based adaptive MCMC algorithms using the framework of RLMH, which we recall in Section 2.1. Motivated by a trade-off between flexibility of the transition kernel and learnability of the associated RL task (i.e. seeking a low-dimensional action space), we focus on RMALA (3) with $G(\cdot) = \epsilon(\cdot)^{-1}G_0$ where G_0 is a fixed symmetric positive definite matrix and $\epsilon(\cdot)$ is a positive *step-size function* to be learned using RL. Intuitively, such a position-dependent preconditioner G has the potential to overcome the tuning difficulties of MALA, while controlling the difficulty of the RL task and not requiring second order derivative information on the target. A key technical difficulty is that the usual performance criteria of AAR and ESJD are inadequate for RLMH; we explain the issue in Section 2.2 and propose a novel alternative criterion based on contrastive divergence in Section 2.3, which may be of independent interest. A comprehensive empirical assessment based on the `posteriordb` benchmark

[2] Some less widely used criteria include the asymptotic variance of quantities of interest (Andrieu and Robert, 2001), the raw acceptance probabilities (Titsias and Dellaportas, 2019), and divergences between the proposal and the target distributions (Andrieu and Moulines, 2006; Dharamshi et al., 2023).

[3] An optimal AAR of 0.234 was established for a class of random walk Metropolis–Hastings algorithms in Gelman et al. (1997), while the value 0.574 was established for MALA in Roberts and Rosenthal (1998).

[4] Throughout this paper we focus on *deterministic* policies, though stochastic policies are also widely used in RL. A deterministic policy is essential to RLMH, as additional stochasticity in the policy can introduce render the Markov chain no longer P-invariant.

is presented in Section 3 and a discussion is provided in Section 4.

2 METHODS

The notation and set-up for RLMH are recalled in Section 2.1. The inadequacy of standard performance criteria is explained in Section 2.2, and our novel performance criterion, based on contrastive divergence, is presented in Section 2.3. Though our main focus is MALA and RMALA, we briefly discuss opportunities to leverage RL also for other MCMC algorithms in Appendix E.5.

2.1 Reinforcement Learning for RMALA

This section explains how RL can be employed to tune RMALA using the framework of RLMH. Our setting is (3) with the specific choice $G(\cdot) = \epsilon(\cdot)^{-1}G_0$, where $\epsilon : \mathbb{R}^d \to (0, \infty)$ is to be learned using RL. Since we seek flexible deterministic policies $\epsilon(\cdot)$ for continuous action spaces, we employed a deep deterministic policy gradient (DDPG) (Lillicrap et al., 2016) method[5]. That is, the function $\epsilon \equiv \epsilon_\theta$ is taken to be a deep neural network with parameters denoted $\theta \in \mathbb{R}^p$. Since we will be learning θ in an online manner, we write θ_n to denote the value of the parameters at iteration n. Both the initial state X_0 of the Markov chain and the initial parameters θ_0 are randomly initialised. The resulting algorithm is fairly straight-forward and consists of alternating, with some fixed frequency, between the following two steps:

Update X_n via RMALA Given the current state X_n of the Markov chain and the current parameters θ_n of RMALA, propose a new state

$$X^*_{n+1}|X_n \sim Q_{\theta_n}(\cdot|X_n)$$

where $Q_{\theta_n}(\cdot|X_n)$ is a multivariate normal distribution with mean $X_n + \epsilon_{\theta_n}(X_n)G_0^{-1}(\nabla \log p)(X_n)$ and covariance $2\epsilon_{\theta_n}(X_n)G_0^{-1}$. Letting $q_{\theta_n}(\cdot|x)$ denote a density for $Q_{\theta_n}(\cdot|x)$, we then set X_{n+1} equal to X^*_{n+1} with probability

$$\alpha(X^*_{n+1}|X_n) := \min\left\{1, \frac{p(X^*_{n+1})}{p(X_n)}\frac{q_{\theta_n}(X_n|X^*_{n+1})}{q_{\theta_n}(X^*_{n+1}|X_n)}\right\}$$

else set X_{n+1} equal to X_n.

[5]For concreteness we present the methodology using DDPG in the main text, but we note that other RL algorithms compatible with deterministic policies and continuous action spaces could be considered, such as the TD3 method of Fujimoto et al. (2018).

Update θ_n via Policy Gradient Conceptually, a *policy gradient* method attempts to take a step of gradient ascent

$$\theta_{n+1} = \theta_n + \beta_n(\nabla J)(\theta_n),$$
$$J(\theta) = \mathbb{E}\left[\sum_{n=1}^{\infty}\gamma^{n-1}r_n(s_n, a_n, s_{n+1})\right], \qquad (4)$$

where $\beta_n \in (0, \infty)$ is a learning rate, $\gamma \in (0, 1)$ is a discount factor[6], and r_n is a reward, to be specified. Thus, for RLMH, the reward r_n can (only) depend on the current state $s_n = [X_n, X^*_{n+1}]$, action $a_n = [\epsilon_{\theta_n}(X_n), \epsilon_{\theta_n}(X^*_{n+1})]$, and subsequent state $s_{n+1} = [X_{n+1}, X^*_{n+1}]$ of the MDP. Of course, the objective J is intractable in general, and most of the research effort into policy gradient methods is invested in developing approximations to the policy gradient. DDPG achieves this by training a differentiable *critic* network to predict the *value* associated to a state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$; since we use DDPG off-the-shelf, we refer the reader to (Lillicrap et al., 2016) for full detail.

Implemented efficiently, the runtime of RLMH is $O(n)$ and the storage requirement is $O(1)$, as for general MCMC. Despite the apparent complexity of the θ_n update via policy gradient, general sufficient conditions for ergodicity of RLMH were established in Wang et al. (2025). In brief, containment can be enforced through parametrisation of the policy, so that the transition kernel is uniformly ergodic when the parameters θ are constrained to a compact set, while diminishing adaptation can be enforced by employing a summable learning rate $(\beta_n)_{n\in\mathbb{N}}$, in conjunction with gradient clipping to ensure that the sequence $(\theta_n)_{n\in\mathbb{N}}$ remains confined to a bounded set. Fortunately, both a summable learning rate and gradient clipping are relatively standard in the context of DDPG. The technical analysis of uniform ergodicity is much more challenging for gradient-based samplers compared to the gradient-free case studied in Wang et al. (2025), and we do not attempt it in this work. Instead, we simply note that one can in practice set the learning rate β_n to zero after a fixed number of iterations have been performed, so that ergodicity is immediate from ergodicity of RMALA (Roy and Zhang, 2023).

2.2 Inadequacy of 'Natural' Rewards

Designing an informative reward is critical to the success of RL. Recall that, in the framework of RLMH,

[6]If the Markov chain is initialised at stationarity, i.e. $X_0 \sim P$, then the rewards r_n are identically distributed and $J(\theta) = \frac{\gamma}{1-\gamma}\mathbb{E}[r_1(s_1, a_1, s_2)]$, meaning the discount factor γ can be absorbed into the learning rate, or simply ignored.

the reward r_n may depend on the current state $s_n = [X_n, X^*_{n+1}]$, the action $a_n = [\epsilon_{\theta_n}(X_n), \epsilon_{\theta_n}(X^*_{n+1})]$, and the subsequent state $s_{n+1} = [X_{n+1}, X^*_{n+1}]$. This rules out the AAR, which requires knowledge of the full sample path, but permits the apparently natural choice of the squared jump distance $r_n = \|X_n - X_{n+1}\|^2$, motivated by the theoretical connection between ESJD and mixing times (Sherlock and Roberts, 2009), or indeed a Rao–Blackwellised version $r_n = \alpha_n \|X_n - X^*_{n+1}\|^2$ where the randomness due to the accept/reject step is integrated out. However, both choices provide inadequate signal for RLMH; the policy gradient is essentially flat when most samples are rejected. A possible remedy is to increase the dynamic range; to this end, the log ESJD (LESJD) reward $r_n = \log(\alpha_n) + 2\log\|X_n - X^*_{n+1}\|$ was proposed in (Wang et al., 2025). Nevertheless, ESJD and related rewards such as LESJD depend critically upon stationarity of the Markov chain; consider sampling from a standard Gaussian using a Metropolis–Hastings proposal $X^*_{n+1} \sim N(-X_n, \epsilon)$. If the Markov chain is far in the tail, e.g. $X_0 = -100$, then optimising ϵ via policy gradient ascent targeting ESJD leads to $\epsilon = 0$, for which large jumps between ± 100 are always accepted but ergodicity is lost. A technical contribution of this work is to propose a more suitable reward that respects the structure of the MDP and improves stability of training RLMH by avoiding the pathologies of ESJD and LESJD.

2.3 Contrastive Divergence Lower Bound

A natural performance criterion for Metropolis–Hastings would be the Kullback–Leibler (KL) divergence between proposal and target, but this requires a sufficiently tractable proposal and a method through which the KL can be approximated (Arbel et al., 2021; Kim et al., 2025). Instead we seek a general criterion, that is similar in spirit to KL but can be easily estimated using only information available in the MDP formulation of RLMH.

To begin, let P_n denote the law of X_n and let p_n be a density for P_n, assumed to exist. The convergence of P_n to P can be quantified using the KL divergence $D_{\mathrm{KL}}(P_n||P) := \int \log(\mathrm{d}P_n/\mathrm{d}P)\mathrm{d}P_n$. Calculating the KL divergence associated with the law of the Markov chain is impractical, since full knowledge of P is required. However, if convergence occurs, then the *contrastive divergence*

$$\Delta_n := D_{\mathrm{KL}}(P_n||P) - D_{\mathrm{KL}}(P_{n+1}||P) \tag{5}$$

ought to vanish in the $n \to \infty$ limit. Here we derive an explicit lower bound on the contrastive divergence, the maximisation of which we adopt as a heuristic to obtain a novel reward that is suitable for RLMH. In the setting of general Metropolis–Hastings algorithms, let $Q(\cdot|x)$ denote the proposal distribution with density $q(\cdot|x)$ assumed to exist for all $x \in \mathbb{R}^d$. Let also $\alpha(x'|x)$ be the probability of accepting a proposed state x' given the current state of the Markov chain is x. The Markov transition kernel takes the form

$$\mathsf{P}(A|x) = \int_A u(x'|x)\,\lambda(\mathrm{d}x') + r(x)\delta_x(A), \tag{6}$$

where $u(x'|x) = \alpha(x'|x)q(x'|x)$, $r(x) = 1 - \int u(x'|x)\,\mathrm{d}\lambda(x')$, and δ_x is a point mass at $x \in \mathbb{R}^d$. Since Metropolis–Hastings chains have a positive probability to reject, this transition kernel is not absolutely continuous with respect to Lebesgue measure λ. To obtain a density for the transition kernel, we instead use the reference measure $\mu_x = \lambda + \delta_x$, for which

$$\mathsf{p}(x'|x) := \frac{\mathrm{d}\mathsf{P}(\cdot|x)}{\mathrm{d}\mu_x}(x') = \begin{cases} u(x'|x), & \text{for } x' \neq x, \\ r(x), & \text{for } x' = x. \end{cases} \tag{7}$$

The proof of the following result can be found in Appendix A:

Proposition 1 (Lower Bound for Δ_n)**.** *Under the above assumptions, we obtain the following lower bound on the contrastive divergence:*

$$\Delta_n \geq \mathbb{E}\left[\log\left(\frac{p(X_{n+1})}{p(X_n)\mathsf{p}(X_{n+1}|X_n)}\right)\right] + \mathbb{E}\left[\log\left(\frac{p_n(X_n)}{1+p_n(X_{n+1})}\right)\right]. \tag{8}$$

The inequality in Proposition 1 is tight, since it is obtained from a single application of Jensen's inequality, which itself is tight. Further, at stationarity the second term in (8) does not depend on the choice of Markov transition kernel P.

Our heuristic to deriving a novel reward for RLMH is to maximise the relevant first term in the lower bound (8) with respect to the design of P, which amounts to maximising the quantity

$$\mathbb{E}\left[\log\left(\frac{p(X_{n+1})}{p(X_n)\mathsf{p}(X_{n+1}|X_n)}\right)\right] = \underbrace{\mathbb{E}[\log p(X_{n+1}) - \log p(X_n)]}_{\text{exploitation}} + \underbrace{\mathbb{E}[-\log \mathsf{p}(X_{n+1}|X_n)]}_{\text{exploration}}. \tag{9}$$

The first term in (9) encourages exploitation, rewarding moves to regions of higher probability, while the second term in (9) encourages exploration, being the expected entropy of the transition kernel. The exploration term admits the following more explicit lower bound, whose proof can be found in Appendix B:

Proposition 2 (Lower bound for exploration term in (9))**.** *Under the above assumptions we obtain a lower*

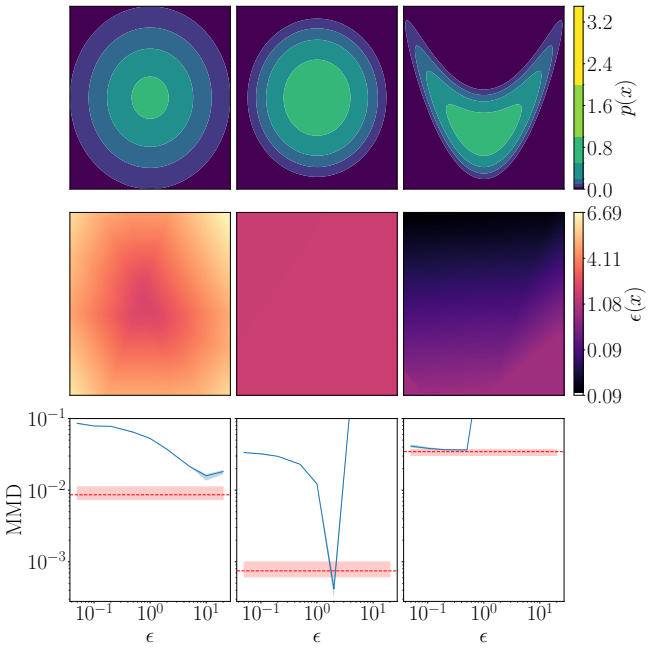
bound

$$\underbrace{\mathbb{E}[-\log \mathsf{p}(X_{n+1}|X_n)]}_{\text{exploration}} \tag{10}$$
$$\geq \mathbb{E}\begin{bmatrix} -\big(1-\alpha(X^*_{n+1}|X_n)\big)\log\big(1-\alpha(X^*_{n+1}|X_n)\big) \\ -\alpha(X^*_{n+1}|X_n)\log q(X^*_{n+1}|X_n) \\ -\alpha(X^*_{n+1}|X_n)\log \alpha(X^*_{n+1}|X_n) \end{bmatrix}.$$

Similarly to Proposition 1, the bound in Proposition 2 is also tight. Let $\alpha_n := \alpha(X^\star_{n+1}|X_n)$ and $\bar{\alpha}_n = 1-\alpha_n$. Combining (9) and (10), we obtain a reward that can be computed using only information available to the MDP formulation of RLMH

$$r_n := \underbrace{\alpha_n[\log p(X^\star_{n+1}) - \log p(X_n)]}_{\text{exploitation}} \tag{11}$$
$$+\underbrace{[-\alpha_n\log\alpha_n - \bar{\alpha}_n\log\bar{\alpha}_n]}_{\text{entropy of accept/reject}} + \underbrace{[-\alpha_n \log q(X^\star_{n+1}|X_n)]}_{\approx \text{ ESJD}}$$

which we call the *contrastive divergence lower bound* (CDLB). The first term in (11) is a Rao–Blackwellised unbiased estimator for the exploitation term in (9), the second term is analogous to the AAR but can be computed in the context of an MDP, and the final term is an affine transformation of the ESJD in the case of a Gaussian proposal. This result may be of independent interest; informally, we have shown that *joint* maximisation of (analogues of) the AAR and ESJD, together with an exploitation incentive, can be sufficient for optimising the performance of a Metropolis–Hastings kernel. In practice the CDLB helps to avoid catastrophic failures that can sometimes occur in RLMH during training with LESJD; empirical support for this claim is provided in Section 3.



(a) Laplace (b) Gaussian (c) Banana

Figure 1: Training the Riemannian Metropolis-adjusted Langevin algorithm (RMALA) using Reinforcement Learning Metropolis–Hastings (RLMH). Top: Target distributions p to be sampled. Middle: Position-dependent step sizes $\epsilon(\cdot)$ learned using RLMH. Bottom: Performance measured using maximum mean discrepancy (MMD). A total of 10 independent simulations were performed and the 25, 50, and 75 percentiles of the MMD are displayed for position-independent RMALA (blue) as a function of the constant step size ϵ, and for RLMH (red) where $\epsilon(\cdot)$ is learned.

3 EMPIRICAL ASSESSMENT

This section presents an empirical assessment of RLMH applied to train RMALA based on the CDLB reward. Full details on the implementation of RMALA are contained in Appendix C, and full details for RLMH are contained in Appendix D. Code to reproduce our experiments is available on GitHub. All computation was performed on a Red Hat Enterprise Linux Server 7.9 (Maipo) system using Intel Xeon E5-2699 v4 CPUs and took 156 CPU hours in total.

Illustration of Reinforcement Learning First we illustrate the behaviour of RMALA trained using RLMH on several examples in dimension $d = 2$, to better understand the nature of the policies that are being learned. The top row of Figure 1 depicts the target distributions p to be sampled. Recall that we implement RMALA with preconditioner of the form $G(\cdot) = \epsilon(\cdot)^{-1}I$, with the step-size function $\epsilon(\cdot)$ learned using RLMH based on the proposed CDLB reward and, for these illustrations, G_0 was the identity matrix. To visualise the learned policy, the middle row of Figure 1 plots the step size function $\epsilon(\cdot)$. For the Laplace target in (a), our algorithm has learned to propose larger steps when the Markov chain is further from the mode; this is actually essential for the Markov chain to rapidly return from an excursion in the tail, since for this target $\nabla \log p$ is bounded. In contrast, for the Gaussian target in (b) a near-constant step size $\epsilon(\cdot)$ has been learned, which we contend is optimal because for a Gaussian the Hessian of $\log p$ is constant. For the banana target (c), the algorithm has learned to take smaller steps when in the narrow tails, which is sensible in light of the larger values of the gradient. The bottom row of Figure 1 contrasts the performance of RLMH to RMALA with a position-independent preconditioner $G(\cdot) = \epsilon^{-1}I$ across differ-

ent values of the constant step size $\epsilon \in (0, \infty)$. Performance was measured using maximum mean discrepancy (MMD) based on the Gaussian kernel with unit lengthscale ($\ell = 1$); we precisely define MMD in Appendix E.2. Small improvements are observed when learning a position-dependent step size in (a) and (c), while this was not the case for (b) since here a constant step size is already optimal.

Framework for the Assessment Magnusson et al. (2025) introduced `posteriordb` to address the lack of a community benchmark for performance assessment in Bayesian computation. By containing a range of realistic posterior distributions arising from genuine data analyses, together with gold-standard samples for many of these tasks, it aims to avoid the practices of hand-picked benchmarks, or classical problems that are insufficiently difficult. Following Wang et al. (2025), we considered the collection of 44 posteriors from `posteriordb` for which 10,000 gold-standard samples are provided. For each posterior and each sampling method, a total of $n = 30,000$ iterations were performed. Assessment was based on the final 5,000 iterations, during which no adaptation was permitted so that in the testing phase all Markov chains were p-invariant. Performance was measured using MMD relative to the gold-standard samples for each task. Here the MMD was based on the Gaussian kernel, whose lengthscale was determined using the *median heuristic* applied to the gold standard samples (Garreau et al., 2017); c.f. Appendix E.2.

All algorithms that we consider require the preconditioner matrix G_0 to be estimated and this is not always straightforward; for example, one could attempt to approximate the Hessian matrix at a mode of the target, or use a simpler off-the-shelf adaptive MCMC method in which a covariance matrix is iteratively learned (Andrieu and Thoms, 2008; Titsias and Dellaportas, 2019). Though it is possible to combine estimation of G_0 with estimation of ϵ or ϕ, we sought to avoid confounding due to the specific choice of how G_0 is estimated; all algorithms that we compare take G_0 to be the inverse of the empirical covariance matrix of the 10^4 gold-standard samples from the target. This choice is justified since we report only a relative comparison of tuning strategies for RMALA. Further, our results are robust to the use of alternative strategies for selecting G_0, as discussed in Appendix E.4.

Performance Assessment Using the `posteriordb` benchmark, we compared the performance of standard RMALA, i.e. based on a preconditioner of the form $G(\cdot) = \epsilon^{-1} G_0$, with constant step-size ϵ tuned either by matching the average acceptance rate (AAR) to 0.574 or by maximising the expected squared jump distance (ESJD), to RMALA with preconditioner of the form $G(\cdot) = \epsilon(\cdot)^{-1} G_0$ with $\epsilon(\cdot)$ learned using RLMH, based either using the LESJD reward of Wang et al. (2025) or the proposed contrastive divergence lower bound (CDLB). The gradient-free RLMH method of Wang et al. (2025) was also included as a benchmark. Full implementation details for RMALA are contained in Appendix C, while full implementation details for RLMH are contained in Appendix D. The computational cost of all gradient-based algorithms was equal, in terms of the number of calls to evaluate either the target p or its gradient.

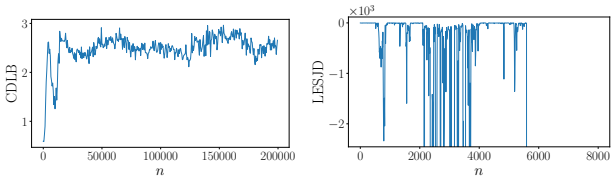


Figure 2: Comparison of training based on CDLB (left panel) and LESJD (right panel). Moving averages of the rewards r_n are displayed. Training with CDLB oscillates during the initial exploration phase, before stabilising once sufficient information has been gained. In contrast, LESJD provides insufficient signal and can lead to catastrophic failure of RLMH.

Summarised results are shown in Table 1, with full results presented in Appendix E.3. The headline is that a position-dependent step size $\epsilon(\cdot)$, learned using RLMH, outperformed a (optimised) constant step size in RMALA, with the smallest average MMD achieved by RLMH in 89% of the tasks considered. All gradient-based samplers outperformed the gradient-free instance of RLMH studied in Wang et al. (2025). On the negative side, for the highest-dimensional tasks no improvement due to the position-dependent step size was observed; this can be attributed to increased difficulty of learning the step size function $\epsilon(\cdot)$ when the domain is high-dimensional. Further, a closer inspection of the sample paths reveals that algorithms based on RLMH converge in a more complex manner compared to classical adaptive MCMC, for which performance typically improves monotonically when a constant step size ϵ is being optimised. This behaviour is illustrated in the left panel of Figure 2.

A secondary, but still important result was that, although the LESJD reward tended to perform slightly better than CDLB as a training objective for RLMH, catastrophic failures were encountered during training for 7% of the tasks when the LESJD reward was used. Such an instance of catastrophic failure is documented in the right panel of Figure 2. These failures were com-

Task	d	Gradient-free RLMH LESJD	RMALA AAR	RMALA ESJD	RMALA-RLMH (proposed) LESJD	RMALA-RLMH (proposed) CDLB
earnings-earn_height	3	1.8(0.1)E-1	5.2(0.3)E-1	5.0(0.0)E-1	**1.4(0.1)E-2**	1.5(0.1)E-2
earnings-log10earn_height	3	1.6(0.0)E-1	1.4(0.0)E-2	1.5(0.1)E-2	1.4(0.1)E-2	**1.3(0.0)E-2**
earnings-logearn_height	3	1.6(0.0)E-1	1.5(0.0)E-2	1.6(0.1)E-2	**1.3(0.1)E-2**	1.4(0.0)E-2
gp_pois_regr-gp_regr	3	1.2(0.0)E-1	**2.2(0.1)E-2**	2.3(0.1)E-2	2.4(0.1)E-2	2.4(0.1)E-2
kidiq-kidscore_momhs	3	1.5(0.0)E-1	1.4(0.1)E-2	1.5(0.1)E-2	**1.2(0.1)E-2**	**1.2(0.0)E-2**
kidiq-kidscore_momiq	3	1.7(0.0)E-1	1.5(0.1)E-2	1.5(0.1)E-2	**1.2(0.0)E-2**	1.4(0.1)E-2
kilpisjarvi_mod-kilpisjarvi	3	1.7(0.0)E-1	5.9(0.3)E-1	6.0(0.3)E-1	**2.5(0.7)E-2**	**2.5(0.6)E-2**
mesquite-logmesquite_logvolume	3	1.3(0.0)E-1	2.2(0.1)E-2	2.4(0.1)E-2	**1.9(0.1)E-2**	2.0(0.1)E-2
arma-arma11	4	1.2(0.0)E-1	2.3(0.1)E-2	2.5(0.1)E-2	2.1(0.1)E-2	**2.0(0.1)E-2**
earnings-logearn_height_male	4	1.6(0.0)E-1	1.4(0.1)E-2	1.6(0.1)E-2	**1.3(0.1)E-2**	**1.3(0.1)E-2**
earnings-logearn_logheight_male	4	1.6(0.0)E-1	1.3(0.1)E-2	1.5(0.1)E-2	**1.1(0.1)E-2**	1.2(0.1)E-2
garch-garch11	4	1.4(0.0)E-1	**1.4(0.1)E-2**	1.7(0.1)E-2	-	8.9(7.2)E-2
hmm_example-hmm_example	4	1.3(0.0)E-1	2.1(0.1)E-2	2.2(0.1)E-2	**1.7(0.1)E-2**	1.9(0.1)E-2
kidiq-kidscore_momhsiq	4	1.4(0.0)E-1	1.6(0.1)E-2	1.8(0.1)E-2	**1.2(0.1)E-2**	1.7(0.1)E-2
⋮	⋮	⋮	⋮	⋮	⋮	⋮
diamonds-diamonds	26	2.0(0.1)E0	6.1(0.8)E-2	**5.3(0.6)E-2**	6.2(0.6)E-2	5.8(0.7)E-2
mcycle_gp-accel_gp	66	1.9(0.0)E0	**7.6(0.0)E-1**	7.7(0.0)E-1	-	7.7(0.0)E-1

Table 1: Benchmarking using `posteriordb`. Here we compared standard RMALA with constant step-size $\epsilon \in (0, \infty)$ tuned either by matching the average acceptance rate (AAR) to 0.574 or by maximising the expected squared jump distance (ESJD), to RMALA with position-dependent step size $\epsilon(\cdot)$ learned using RLMH, based either using the LESJD reward of Wang et al. (2025) or the proposed CDLB. The gradient-free RLMH method of Wang et al. (2025) is also included as a benchmark. Performance was measured using maximum mean discrepancy (MMD) based on the Gaussian kernel with lengthscale selected using the median heuristic applied to $10,000$ gold-standard samples from the target, and d is the dimension of the target. Results are based on an average of 10 replicates, with standard errors (in parentheses) reported. The method with smallest average MMD is highlighted in **bold**, gray rows indicate tasks for which RLMH performed best, and dashes indicate where catastrophic failure occurred during training of RLMH. Full results are in Appendix E.3.

pletely ameliorated when the CDLB reward was used.

Though we focused on RMALA, the prototypical gradient-based sampling method, it is interesting to explore the potential of RLMH for tuning of other gradient-based samplers. To this end, analogous results for the Barker proposal of Livingstone and Zanella (2022) can be found in Appendix E.5. Interestingly we did not find compelling evidence for the use of a flexible step size for the Barker proposal; this makes sense because the Barker proposal was specifically developed to trade-off some efficiency for being relatively insensitive to the step size of the proposal.

4 DISCUSSION

The impressive performance of modern RL, on applications such as autonomous driving (Kiran et al., 2021), gaming (Silver et al., 2016), and natural language processing (Shinn et al., 2023), serves as strong motivation for investigating if and how techniques from RL can be harnessed for adaptive MCMC. This paper is the first to explore the use of RL to tune gradient-based MCMC. Our empirical findings demonstrate that fast-mixing Markov kernels can be effectively learned within the framework of RLMH. The main limitation is that poor sample quality can occur during the initial exploration phase of RL, analogous to an extended warm-up period in MCMC; we therefore recommend the use of gradient-based RLMH primarily in settings where high-quality posterior approximation is required (i.e. for fast posterior approximations at lower quality, simpler methods should be preferred). In addition we presented the novel CDLB reward, which stabilised training and may have applications in adaptive MCMC beyond this work.

For future research, we highlight that learning the proposal covariance structure (i.e. G_0 in the setting of RMALA) is an open challenge for RL; we speculate that reduced-rank covariance matrix approximations may be useful here, enabling the difficulty of the learning task to be reduced, but our attempts to implement this (not shown) were unsuccessful. Since multiple instances of MCMC can be run in parallel, federated Q-learning and related strategies may offer a promising way forward. Alternatively, one could explore the application of RL to tuning other sampling methods, such as selecting the step size and number of integrator 'leaps' in adaptive Hamiltonian Monte Carlo (Wang et al., 2013; Hoffman and Gelman, 2014; Christiansen et al., 2023), or consider the formulation of related algorithms such as multiple-try Metropolis (Liu et al., 2000; Doig and Wang, 2025) and delayed acceptance MCMC (Christen and Fox, 2005) as MDPs amenable to RL.

Acknowledgements

CW was supported by the China Scholarship Council under Grant Number 202208890004. HK and CJO were supported by the Engineering and Physical Sciences Research Council EP/W019590/1. CJO was supported by a Philip Leverhulme Prize PLP-2023-004.

References

C. Andrieu and É. Moulines. On the ergodicity properties of some adaptive MCMC algorithms. *The Annals of Applied Probability*, 16(1):1462–1505, 2006.

C. Andrieu and C. P. Robert. Controlled MCMC for optimal sampling. Technical Report 33, Center for Research in Economics and Statistics, 2001.

C. Andrieu and J. Thoms. A tutorial on adaptive MCMC. *Statistics and Computing*, 18:343–373, 2008.

M. Arbel, A. Matthews, and A. Doucet. Annealed flow transport Monte Carlo. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

Y. Atchade, G. Fort, E. Moulines, and P. Priouret. *Adaptive Markov chain Monte Carlo: Theory and methods*. Bayesian Time Series Models. Cambridge University Press, 2011.

Y. F. Atchadé and J. S. Rosenthal. On adaptive Markov chain Monte Carlo algorithms. *Bernoulli*, 11(5):815–828, 2005.

A. A. Barker. Monte Carlo calculations of the radial distribution functions for a proton-electron plasma. *Australian Journal of Physics*, 18(2):119–134, 1965.

M. Biron-Lattes, N. Surjanovic, S. Syed, T. Campbell, and A. Bouchard-Côté. autoMALA: Locally adaptive Metropolis-adjusted Langevin algorithm. In *Proceedings of the 27th International Conference on Artificial Intelligence and Statistics*, 2024.

D. Blessing, J. Berner, L. Richter, and G. Neumann. Underdamped diffusion bridges with applications to sampling. In *Proceedings of the 13th International Conference on Learning Representations*, 2025.

T. Bui-Thanh and O. Ghattas. A scaled stochastic Newton algorithm for Markov chain Monte Carlo simulations. *SIAM Journal on Uncertainty Quantification*, pages 1–25, 2012.

Y. Chen, D. Z. Huang, J. Huang, S. Reich, and A. M. Stuart. Sampling via gradient flows in the space of probability measures. *arXiv preprint arXiv:2310.03597*, 2023.

N. Chopin and O. Papaspiliopoulos. *An Introduction to Sequential Monte Carlo*. Springer, 2020.

J. A. Christen and C. Fox. Markov chain Monte Carlo using an approximation. *Journal of Computational and Graphical statistics*, 14(4):795–810, 2005.

H. Christiansen, F. Errica, and F. Alesiani. Self-tuning Hamiltonian Monte Carlo for accelerated sampling. *The Journal of Chemical Physics*, 159(23), 2023.

E. Chung, W. T. Leung, S.-M. Pun, and Z. Zhang. Multi-agent reinforcement learning accelerated MCMC on multiscale inversion problem. *Journal of Scientific Computing*, 93(2), 2023.

J. Coullon, L. South, and C. Nemeth. Efficient and generalizable tuning strategies for stochastic gradient MCMC. *Statistics and Computing*, 33(3):66, 2023.

A. Dharamshi, V. Ngo, and J. S. Rosenthal. Sampling by divergence minimization. *Communications in Statistics*, pages 1–25, 2023.

R. Doig and L. Wang. A unified framework for multiple-try Metropolis algorithms. *arXiv preprint arXiv:2503.11583*, 2025.

A. Eberle. Reflection couplings and contraction rates for diffusions. *Probability Theory and Related Fields*, 166:851–886, 2016.

C. W. Fox and S. J. Roberts. A tutorial on variational Bayesian inference. *Artificial Intelligence Review*, 38:85–95, 2012.

S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 37th International Conference on Machine Learning*, 2018.

D. Garreau, W. Jitkrittum, and M. Kanagawa. Large sample analysis of the median heuristic. *arXiv preprint arXiv:1707.07269*, 2017.

A. Gelman, W. R. Gilks, and G. O. Roberts. Weak convergence and optimal scaling of random walk Metropolis algorithms. *The Annals of Applied Probability*, 7(1):110–120, 1997.

M. Girolami and B. Calderhead. Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 73(2):123–214, 2011.

H. Haario, M. Laine, A. Mira, and E. Saksman. DRAM: Efficient adaptive MCMC. *Statistics and Computing*, 16:339–354, 2006.

M. D. Hoffman and A. Gelman. The No-U-Turn Sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

N. T. Hunt-Smith, W. Melnitchouk, F. Ringer, N. Sato, A. W. Thomas, and M. J. White. Accelerating Markov chain Monte Carlo sampling with dif-

fusion models. *Computer Physics Communications*, 296:109059, 2024.

L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

K. Kim, Z. Xu, J. R. Gardner, and T. Campbell. Tuning sequential Monte Carlo samplers via greedy incremental divergence minimization. *arXiv preprint arXiv:2503.15704*, 2025.

B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2021.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *Proceedings of the 4th International Conference on Learning Representations*, 2016.

J. S. Liu, F. Liang, and W. H. Wong. The multiple-try method and local optimization in Metropolis sampling. *Journal of the American Statistical Association*, 95(449):121–134, 2000.

T. Liu, N. Surjanovic, M. Biron-Lattes, A. Bouchard-Côté, and T. Campbell. AutoStep: Locally adaptive involutive MCMC. *arXiv preprint arXiv:2410.18929*, 2024.

S. Livingstone and G. Zanella. The Barker proposal: Combining robustness and efficiency in gradient-based MCMC. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 84(2):496–523, 2022.

M. Magnusson, J. Torgander, P.-C. Bürkner, L. Zhang, B. Carpenter, and A. Vehtari. `posteriordb`: Testing, benchmarking and developing bayesian inference algorithms. In *Proceedings of the 28th International Conference on Artificial Intelligence and Statistics*, 2025. URL `https://github.com/stan-dev/posteriordb`.

N. Mahendran, Z. Wang, F. Hamze, and N. De Freitas. Adaptive MCMC with Bayesian optimization. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, pages 751–760, 2012.

A. Naik, Y. Wan, M. Tomar, and R. S. Sutton. Reward centering, 2024. URL `https://arxiv.org/abs/2405.09999`.

C. Pasarica and A. Gelman. Adaptively scaling the Metropolis algorithm using expected squared jumped distance. *Statistica Sinica*, pages 343–364, 2010.

C. C. Potter and R. H. Swendsen. 0.234: The myth of a universal acceptance ratio for Monte Carlo simulations. *Physics Procedia*, 68:120–124, 2015.

D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22 (3):400–407, 1951. doi: 10.1214/aoms/1177729586.

G. O. Roberts and J. S. Rosenthal. Optimal scaling of discrete approximations to Langevin diffusions. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 60(1):255–268, 1998.

G. O. Roberts and J. S. Rosenthal. Coupling and ergodicity of adaptive Markov chain Monte Carlo algorithms. *Journal of Applied Probability*, 44(2):458–475, 2007.

G. O. Roberts and J. S. Rosenthal. Examples of adaptive MCMC. *Journal of Computational and Graphical Statistics*, 18(2):349–367, 2009.

G. O. Roberts and O. Stramer. Langevin diffusions and metropolis-hastings algorithms. *Methodology and Computing in Applied Probability*, 4:337–357, 2002.

G. O. Roberts and R. Tweedie. Exponential convergence of Langevin diffusions and their discrete approximation. *Bernoulli*, 2:341–363, 1996.

V. Roy and L. Zhang. Convergence of position-dependent MALA with application to conditional simulation in GLMMs. *Journal of Computational and Graphical Statistics*, 32(2):501–512, 2023.

C. Sherlock and G. O. Roberts. Optimal scaling of the random walk Metropolis on elliptically symmetric unimodal targets. *Bernoulli*, 15(3):774–798, 2009.

N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning. In *Proceedings of the 36th Conference on Neural Information Processing Systems*, 2023.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, pages 387–395. PMLR, 2014.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach,

K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

M. Titsias and P. Dellaportas. Gradient-based adaptive Markov chain Monte Carlo. In *Proceedings of the 32nd Conference on Neural Information Processing Systems*, 2019.

C. Wang, W. Chen, H. Kanagawa, C. Oates, et al. Reinforcement learning for adaptive MCMC. In *Proceedings of the 28th International Conference on Artificial Intelligence and Statistics*, 2025.

Z. Wang, S. Mohamed, and N. Freitas. Adaptive Hamiltonian and Riemann manifold Monte Carlo. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

Z. Wang, Y. Xia, S. Lyu, and C. Ling. Reinforcement learning-aided Markov chain Monte Carlo for lattice Gaussian sampling. In *Proceedings of the IEEE Information Theory Workshop*. IEEE, 2021.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes/Not Applicable; all algorithms are $O(n)$ in time and storage, which is standard for MCMC.]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes, uploaded as supplement.]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes, written in the pdf supplement.]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes, in captions of figures and tables.]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes, in main text.]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes; we cite the creators of the `posteriordb` benchmark.]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Supplementary Materials

These appendices supplement the paper *Harnessing the Power of Reinforcement Learning for Adaptive MCMC*. Appendix A contains the Proof of Proposition 1. Appendix B contains the proof of Proposition 2. Full implementational details for RMALA and RLMH are contained, respectively, in Appendix C and Appendix D. Full results from the `posteriordb` benchmark are contained in Appendix E.

A Proof of Proposition 1

This appendix contains the proof of Proposition 1 in the main text.

Proof of Proposition 1. First, to clarify the notation from (7), note that the (Lebesgue) density p_{n+1} for the law P_{n+1} of the $(n+1)$th state X_{n+1} of the Markov chain can be expressed recursively as

$$p_{n+1}(x) = \underbrace{r(x)p_n(x)}_{\text{reject}} + \underbrace{\int u(x|x')p_n(x')\,\mathrm{d}\lambda(x')}_{\text{accept}} = \int \mathsf{p}(x|x')p_n(x')\,\mathrm{d}\mu_x(x'). \tag{12}$$

Next, we manipulate the definition of contrastive divergence Δ_n in, stated in (5), to obtain that

$$\begin{aligned}
\Delta_n &= \int \log\left(\frac{\mathrm{d}P_n}{\mathrm{d}P}\right)\mathrm{d}P_n - \int \log\left(\frac{\mathrm{d}P_{n+1}}{\mathrm{d}P}\right)\mathrm{d}P_{n+1} \\
&= \int p_n \log p_n \,\mathrm{d}\lambda - \int p_n \log p \,\mathrm{d}\lambda + \int p_{n+1}\log p\,\mathrm{d}\lambda - \int p_{n+1}\log p_{n+1}\,\mathrm{d}\lambda \\
&= D_{\mathrm{KL}}(P_n||\lambda) + \mathbb{E}\left[\log\left(\frac{p(X_{n+1})}{p(X_n)}\right)\right] - \int p_{n+1}\log p_{n+1}\,\mathrm{d}\lambda.
\end{aligned} \tag{13}$$

Considering the final integral, we use (12) to obtain

$$\begin{aligned}
\int p_{n+1}\log p_{n+1}\,\mathrm{d}\lambda &= \int \left(\int \mathsf{p}(x|x')p_n(x')\,\mathrm{d}\mu_x(x')\right)\log\left(\int \mathsf{p}(x|x')p_n(x')\,\mu_x(x')\right)\,\mathrm{d}\lambda(x) \\
&= \int \underbrace{\psi\left(\int \mathsf{p}(x|x')p_n(x')\,\mathrm{d}\mu_x(x')\right)}_{(\star)}\,\mathrm{d}\lambda(x)
\end{aligned}$$

where $\psi(y) := y\log y$. Next we will use Jensen's inequality on $(\star)$, and to make this clear we introduce an explicit normalisation constant

$$Z_n(x) := \int p_n(x')\mathrm{d}\mu_x(x') = 1 + p_n(x),$$

so

$$\int \mathsf{p}(x|x')p_n(x')\,\mathrm{d}\mu_x(x') = \int [Z_n(x)\mathsf{p}(x|x')]\frac{p_n(x')}{Z_n(x)}\,\mathrm{d}\mu_x(x'), \qquad \int \frac{p_n(x')}{Z_n(x)}\,\mathrm{d}\mu_x(x') = 1.$$

Since ψ is convex, Jensen's inequality gives that

$$\begin{aligned}
(\star) = \psi\left(\int [Z_n(x)\mathsf{p}(x|x')]\frac{p_n(x')}{Z_n(x)}\,\mathrm{d}\mu_x(x')\right) &\leq \int \psi(Z_n(x)\mathsf{p}(x|x'))\frac{p_n(x')}{Z_n(x)}\,\mathrm{d}\mu_x(x') \\
&= \int \mathsf{p}(x|x')p_n(x')\log[Z_n(x)\mathsf{p}(x|x')]\,\mathrm{d}\mu_x(x')
\end{aligned}$$

from which we deduce that

$$\int p_{n+1} \log p_{n+1} \, \mathrm{d}\lambda \leq \mathbb{E}[\log \mathsf{p}(X_{n+1}|X_n)] + \mathbb{E}[\log Z_n(X_{n+1})]. \tag{14}$$

Substituting (14) into (13) gives that

$$\begin{aligned}\Delta_n &\geq D_{\mathrm{KL}}(P_n||\lambda) + \mathbb{E}\left[\log\left(\frac{p(X_{n+1})}{p(X_n)}\right)\right] - \mathbb{E}[\log \mathsf{p}(X_{n+1}|X_n)] - \mathbb{E}[\log Z_n(X_{n+1})] \\ &= \mathbb{E}\left[\log\left(\frac{p(X_{n+1})}{p(X_n)\mathsf{p}(X_{n+1}|X_n)}\right)\right] + \mathbb{E}\left[\log\left(\frac{p_n(X_n)}{1+p_n(X_{n+1})}\right)\right]\end{aligned}$$

and completes the argument. □

B Proof of Proposition 2

This appendix contains the proof of Proposition 2 in the main text.

Proof of Proposition 2. To begin, we condition on the value of X_n and recall the definition of the transition density $\mathsf{p}(X_{n+1}|X_n)$ from the main text:

$$\begin{aligned}\mathbb{E}[-\log \mathsf{p}(X_{n+1}|X_n) \mid X_n] &= -\int \log(\mathsf{p}(x'|X_n))\mathsf{p}(x'|X_n) \, \mathrm{d}\mu_{X_n}(x') \\ &= -r(X_n)\log r(X_n) - \int u(x'|X_n)\log u(x'|X_n)\,\mathrm{d}\lambda(x') \\ &= \underbrace{-\left(1-\int u(x'|X_n)\,\mathrm{d}\lambda(x')\right)\log\left(1-\int u(x'|X_n)\,\mathrm{d}\lambda(x')\right)}_{(15.1)} \\ &\quad \underbrace{-\int u(x'|X_n)\log u(x'|X_n)\,\mathrm{d}\lambda(x')}_{(15.2)}.\end{aligned} \tag{15}$$

where we recall from the main text that $r(x)$ is the probability of rejection if the current state of the Markov chain is x, and $u(x'|x) = \alpha(x'|x)q(x'|x)$ is the acceptance-proposal density. Thus the first term (15.1) in (15) can be expressed as an expectation with respect to the proposal, as

$$(15.1) = -\left(1-\mathbb{E}[\alpha(X^*_{n+1}|X_n)|X_n]\right)\log\left(1-\mathbb{E}[\alpha(X^*_{n+1}|X_n)|X_n]\right).$$

This has the form $-\mathbb{E}[Y]\log\mathbb{E}[Y]$ for $Y = 1-\alpha(X^*_{n+1}|X_n)|X_n$, and thus from convexity of $\psi(y) = y\log y$ we may again use Jensen's inequality to obtain the lower bound $-\mathbb{E}[\psi(Y)]$, which corresponds to

$$(15.1) \geq -\mathbb{E}\left[\left(1-\alpha(X^*_{n+1}|X_n)\right)\log\left(1-\alpha(X^*_{n+1}|X_n)\right) \mid X_n\right]$$

The second term (15.2) in (15) can be expressed as

$$(15.2) = -\mathbb{E}\left[\alpha(X^*_{n+1}|X_n)[\log q(X^*_{n+1}|X_n) + \log\alpha(X^*_{n+1}|X_n)] \mid X_n\right].$$

Thus we have shown the stated inequality holds conditional on X_n, and taking a final expectation with respect to X_n completes the argument. □

C Implementation of RMALA

This appendix precisely describes RMALA, together with the procedure that we used to tune RMALA when the preconditioner is position-independent.

First, and for completeness, full pseudocode for RMALA is contained in Algorithm 1. Here S^+_d denotes the set of symmetric positive definite $d \times d$ matrices and $|M|$ denotes the determinant of a matrix $M \in \mathrm{S}^+_d$. The

Algorithm 1 Riemannian MALA; Girolami and Calderhead, 2011

Require: $x_0 \in \mathbb{R}^d$ (initial state), $G(\cdot) \in \mathrm{S}_d^+$ (position-dependent preconditioner matrix), $n \in \mathbb{N}$ (number of iterations)

1: $X_0 \leftarrow x_0$ ▷ initialise Markov chain

2: **for** $i = 0, 1, 2, \ldots, n-1$ **do**

3: $\quad X_{i+1}^* \leftarrow \underbrace{X_i + G(X_i)^{-1}(\nabla \log p)(X_i)}_{=:\nu(X_i)} + \sqrt{2} G(X_i)^{-1/2} Z_i$ ▷ proposal; $Z_i \sim \mathcal{N}(0, I)$

4: $\quad \alpha_i \leftarrow \min\left(1, \frac{p(X_{i+1}^*)}{p(X_i)} \frac{|G(X_i)|^{1/2}}{|G(X_{i+1}^*)|^{1/2}} \frac{\exp\left(-\|G(X_{i+1}^*)^{-1/2}(X_i - \nu(X_{i+1}^*))\|^2\right)}{\exp\left(-\|G(X_i)^{-1/2}(X_{i+1}^* - \nu(X_i))\|^2\right)}\right)$

5: $\quad X_{i+1} \leftarrow X_{i+1}^*$ with probability α_i, else $X_{i+1} \leftarrow X_i$ ▷ accept/reject

6: **end for**

7: **return** Markov chain $(X_i)_{i=1}^n$

experiments that we report in Section 3 of the main text concern either the case where the preconditioner matrix $G(\cdot) = \epsilon^{-1} G_0$ is position-independent, or $G(\cdot) = \epsilon(\cdot)^{-1} G_0$ with a position-dependent scale $\epsilon(\cdot)$ learned using RLMH. In each case the choice of G_0 was as described in the main text, with sensitivity to this choice explored in Appendix E.4.

In the case of a position-independent preconditioner, we set the step size ϵ either by attempting to maximise the ESJD or tuning the AAR to 0.574, following the recommendation of Roberts and Rosenthal (1998). For tuning based on either ESJD (resp. AAR), at a fixed frequency T we compare the average squared jump distance (resp. absolute deviation of the empirical acceptance rate from 0.574) D_1 for the chain $(X_i)_{i=n-T+1}^n$ to the corresponding quantity D_2 for the chain $(X_i)_{i=n-2T+1}^{n-T}$. Based on D_1 and D_2, the step size is either increased or decreased by a multiplicative factor of 1.05 subject to remaining in an interval $[10^{-4}, 2]$; these values were chosen to ensure reasonable performance over the `posteriordb` benchmark. In our implementation the adaptation frequency $T = 5,000$ was used. The initial step size was set to $\epsilon = 0.1$. Adaptation continues until such a point that adaptation is halted for the purposes of assessment, as described in the main text.

D Implementation of RLMH

This appendix contains the implementational details required to reproduce the experimental results for RLMH reported in Section 3 of the main text. First we recall the main ideas of DDPG in Appendix D.1, before reporting all training details in Appendix D.2.

D.1 Deep Deterministic Policy Gradient

This section describes the DDPG algorithm with reward centring for maximising $J(\theta)$ in (4), based on the deterministic policy gradient theorem of Silver et al. (2014). Since the algorithm itself is quite detailed, we aim simply to summarise the main aspects of DDPG and refer the reader to Lillicrap et al. (2016) for full detail.

The *deterministic policy gradient theorem* states that

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\pi \left[\nabla_\theta \pi(s) \, \nabla_a \mathrm{Q}_\pi(s, a)|_{a=\pi(s)} \right], \tag{16}$$

where the expectation here is taken with respect to the stationary distribution of the MDP when the policy π is fixed. The *action-value function* $\mathrm{Q}_\pi(s, a)$ gives the expected discounted cumulative reward from taking an action a in state s and following policy π thereafter.

Under standard DDPG, the policy π and the action-value function Q are parameterised by neural networks whose parameters are updated via an *actor-critic algorithm* based on Silver et al. (2014). Specifically, an *actor network* $\pi_\theta(s)$ is updated in the direction of the policy gradient in (16), and a *critic network* Q_ϑ that approximates the action-value function, is updated by solving the Bellman equation

$$\mathrm{Q}_\pi(s_n, a_n) = \mathbb{E}_\pi \left[r_n + \gamma \mathrm{Q}_\pi(s_{n+1}, \pi(s_{n+1})) \right],$$

Algorithm 2 Reward-Centred Deep Deterministic Policy Gradient (DDPG)

Require: s_0 (initial state of the MDP), $\gamma \in (0,1)$ (discount factor), $\mathcal{D}$ (replay buffer), $(\mathcal{N}_t)_{t\geq 0}$ (noise process), $\alpha_\pi > 0$ (actor learning rate), $\alpha_{\mathrm{Q}} > 0$ (critic learning rate), N (minibatch size), $\eta > 0$ (centring gain), $\tau \in (0,1)$ (soft-update rate)

initialise actor parameters θ and critic parameters ϑ
$\pi_{\theta'} \leftarrow \pi_\theta$ $\mathrm{Q}_{\vartheta'} \leftarrow \mathrm{Q}_\vartheta$ ▷ initialise target networks
$\bar{R} \leftarrow 0$ ▷ initialise average reward
for each episode **do**
  for $t = 0, 1, 2, \ldots$ until episode ends **do**
    $a_t = \pi_\theta(s_t) + \mathcal{N}_t$ ▷ action with exploration noise
    execute a_t, observe $(r_t,\, s_{t+1})$
    store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    if update step **then**
      sample minibatch $\{(s_i, a_i, r_i, s_i')\}_{i=1}^N \sim \mathcal{D}$
      $r_i^{\mathrm{c}} \leftarrow r_i - \bar{R}$ ▷ reward centring
      $Q_i \leftarrow r_i^{\mathrm{c}} + \gamma\, \mathrm{Q}_{\vartheta'}\big(s_i', \pi_{\theta'}(s_i')\big)$ ▷ TD approximation
      $\vartheta \leftarrow \vartheta - \alpha_{\mathrm{Q}} \nabla_\vartheta \frac{1}{N} \sum_{i=1}^N \big(Q_i - \mathrm{Q}_\vartheta(s_i, a_i)\big)^2$ ▷ critic update
      $\theta \leftarrow \theta + \alpha_\pi \nabla_\theta \frac{1}{N} \sum_{i=1}^N \mathrm{Q}_\vartheta\big(s_i, \pi_\theta(s_i)\big)$ ▷ actor update
      $\bar{R} \leftarrow \bar{R} + \eta\, \alpha_{\mathrm{Q}} \cdot \frac{1}{N} \sum_{i=1}^N (Q_i - \mathrm{Q}_\vartheta(s_i, a_i))$ ▷ update average reward
      $\vartheta' \leftarrow (1-\tau)\, \vartheta' + \tau\, \vartheta$ ▷ soft-update critic target
      $\theta' \leftarrow (1-\tau)\, \theta' + \tau\, \theta$ ▷ soft-update actor target
    end if
  end for
end for

via stochastic approximation with off-policy samples. In DDPG the critic network is trained by solving the optimisation problem

$$\arg\min_{\vartheta} \mathbb{E}_{\tilde{\pi}} \left[(Q_n - \mathrm{Q}_\vartheta(s_n, a_n))^2 \right], \tag{17}$$

where $Q_n = r_n + \gamma \mathrm{Q}_\vartheta(s_{n+1}, \pi(s_{n+1}))$ is called the *one step* temporal difference (TD) approximation, a_n is generated from a stochastic *behaviour policy* $\tilde{\pi}$, and s_{n+1} is resulted from interacting with the environment using $a_n \sim \tilde{\pi}(s_n)$. The expectation in (17) is approximated by sampling a mini-batch from a *replay buffer* that stores the experience tuples $\mathcal{D} := \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^n$. A common choice for the behaviour policy $\tilde{\pi}$ is a noisy version of the deterministic policy π, e.g., $\tilde{a}_n = a_n + \mathcal{N}_n$, where $(\mathcal{N}_n)_{n\geq 0}$ is a *noise process* that must be specified.

There are several aspects of DDPG that are non-trivial, such as the distinction between *target networks* $\mathrm{Q}_{\vartheta'}$ and $\pi_{\theta'}$ and the current actor and critic networks Q_ϑ and π_θ, and how these networks interact via the *taming factor* τ; see Lillicrap et al. (2016) for further detail.

Drawing on the reward-centring framework of Naik et al. (2024), we replace the vanilla TD approximation Q_n with the *centralised* TD approximation

$$\bar{Q}_n = (r_n - \bar{R}) + \gamma Q_{\vartheta'}\,(s_{n+1}, \pi_\theta(s_{n+1})) \tag{18}$$

where the running average of past rewards is denoted $\bar{R}$. This centring of the immediate reward term around $\bar{R}$ has the capacity to reduce variance in the TD approximation and stabilises learning (Naik et al., 2024). The reward-centred variant of the DDPG is summarised in Algorithm 2, and includes synchronous updates of the running average reward

$$\bar{R} \leftarrow \bar{R} + \eta \cdot \frac{1}{N} \sum_{i=1}^N r_{(i)} \tag{19}$$

where η is a learning rate and the rewards $r_{(i)}$ are drawn from the replay buffer.

For the purpose of this work we largely relied on default settings for DDPG; full details are contained in Appendix D.2. The specific design of RL methods for use in adaptive MCMC was not explored, but might be an interesting avenue for future work.

D.2 Training Details

Here we describe the settings that were used for RLMH. Recall that $\pi_\theta(s_n) = [\epsilon_\theta(X_n), \epsilon_\theta(X^*_{n+1})]$ where $s_n = [X_n, X^*_{n+1}]$ in RLMH.

Parametrisation of ϵ_θ The function ϵ_θ was parametrised using a fully-connected two-layer neural network with the ReLU activation function and 8 features per hidden layer; a total of $p = (8 + d)(d + 1)$ parameters to be inferred.

Pre-training of ϵ_θ The parameters θ of the neural network ϵ_θ were initialised by pre-training against the loss

$$\theta \mapsto \frac{1}{m} \sum_{j=1}^{m} \left\| \epsilon_\theta(y_j) - \epsilon^\dagger \right\|^2,$$

computed over the gold-standard samples $\{y_j\}_{j=1}^m$ with $m = 10^4$, so that the proposal corresponding to ϵ_θ is initially a position-independent proposal with step size $\epsilon^\dagger$. This was achieved using stochastic gradient descent (Robbins and Monro, 1951), employing a batch size of 16, a learning rate of 0.01, and a training duration of 100 epochs.

Of course, a suitable initial step size $\epsilon^\dagger$ will be unknown in general. Inspired by Bui-Thanh and Ghattas (2012), we first considered the heuristic:

$$\varepsilon_0 = \frac{\ell}{\sqrt{\lambda_{\max}(\Sigma^{-1}), d^{1/3}}}$$

where Σ is the covariance matrix computed based on gold-standard samples from the target. This heuristic was further optimised for experiments on `posteriordb` via the nonlinear regression

$$\epsilon^\dagger = a_1 \cdot \varepsilon_0^3 + a_2 \cdot \varepsilon_0^2 + a_3 \cdot \varepsilon_0 + a_4,$$

with numerical coefficients determined through experimentation as $a_1 = 29$, $a_2 = -26$, $a_3 = 3.0$, and $a_4 = 1.3$. This initialisation is merely to ensure that the difference between the initial policy $\epsilon^\dagger$ and typical learned policies remains within three orders of magnitude over the `posteriordb` benchmark, facilitating efficient and stable training on high-performance computing in our assessment of RLMH.

Training of ϵ_θ RLMH was performed using an implementation of DDPG in `Python`. The settings for training ϵ_θ were:

- 100 episodes (i.e. the number of iterations of MCMC performed between each update based on the policy gradient), each consisting of 500 iterations of MCMC
- the noise process $(\mathcal{N}_n)_{n\geq 0}$ was Gaussian with standard deviation equal to the initial step size
- actor learning rate $\alpha_\pi = 10^{-6}$
- experience buffer length $= 2.5 \times 10^4$,
- minibatch size $N = 48$
- centering gain $\eta = 10^{-3}$

Parametrisation of the Critic Q For all experiments we took $\mathrm{Q} : \mathbb{R}^{2d} \times \mathbb{R}^{2d} \to \mathbb{R}$ to be a fully-connected neural network with the ReLU activation function, two hidden layers, and 8 features in each hidden layer. The parametrised critic is denoted Q_ϑ.

Training of the Critic Q_ϑ The settings for training the critic Q_ϑ were identical to those for the actor, with the exceptions:

- critic parameters randomly initialised by Pytorch's default initilaisation method
- critic learning rate $\alpha_\mathrm{Q} = 10^{-2}$.

E Results for the `posteriordb` Benchmark

This appendix contains additional details and results concerning the `posteriordb` benchmark. Additional details required to reproduce our assessment are contained in Appendix E.1. The MMD performance measure is precisely defined in Appendix E.2. The full version of Table 1 from the main text is presented in Appendix E.3. Sensitivity of our conclusions to the mechanism used to select G_0 is investigated in Appendix E.4. Our investigation extended beyond RMALA and full results for the Barker proposal of Livingstone and Zanella (2022) are contained and discussed in Appendix E.5.

E.1 Implementation Details

The initial state $x_0 \in \mathbb{R}^d$ of all Markov chains was taken to be the arithmetic mean of 10^4 gold-standard samples from the target.

For the experiments reported in Section 3 of the main text, the static component $G_0 \in \mathrm{S}_d^+$ of the preconditioner was set to the inverse of the empirical covariance matrix of 10^4 gold-standard samples from the target to avoid confounding due to the specific choice of how G_0 is estimated. In practice, the matrix G_0 would also need to be estimated and this is not always straightforward; for example, one could attempt to approximate the Hessian matrix at a mode of the target, or use a simpler off-the-shelf adaptive MCMC method in which a covariance matrix is iteratively learned (Andrieu and Thoms, 2008).

E.2 Maximum Mean Discrepancy

Performance was measured using MMD relative to the gold-standard samples $\{y_j\}_{j=1}^m$ for each task. Here the MMD was based on the Gaussian kernel $k(x, y) := \exp\bigl(-\|x-y\|^2/\ell^2\bigr)$, whose length scale ℓ was determined using the *median heuristic*

$$\ell := \frac{1}{2}\mathrm{median}\{\|y_i - y_j\| : 1 \leq i, j \leq m\}$$

following Garreau et al. (2017). The MMD $D(P_n, Q_m)$ between a pair of empirical distributions $P_n = \frac{1}{n}\sum_{i=1}^n \delta_{X_i}$ and $Q_m = \frac{1}{m}\sum_{j=1}^m \delta_{y_j}$ is defined via the formula

$$\mathrm{MMD}(P_n, Q_m)^2 := \frac{1}{n^2}\sum_{i=1}^n\sum_{i'=1}^n k(X_i, X_{i'}) - \frac{2}{nm}\sum_{i=1}^n\sum_{j=1}^m k(X_i, y_j) + \frac{1}{m^2}\sum_{j=1}^m\sum_{j'=1}^m k(y_j, y_{j'}),$$

and for this work P_n represents the approximation to the target $p(\cdot)$ produced using an adaptive MCMC algorithm, and Q_m represents a gold-standard set of $m = 10^4$ samples from the target, which are provided in `posteriordb`.

E.3 Full Results for RMALA-RLMH

Full results (i.e. the expanded version of Table 1) are shown in Table 2. Instances where RMALA failed with AAR or ESJD were because proposal covariance became numerically zero; could be avoided but in principle this could introduce bias and so we simply report the failure in these cases.

E.4 Exploring the Sensitivity to G_0

The results that we report for `posteriordb` in the main text set G_0 based on 10^4 gold-standard samples from the target. However, this is an idealised situation since one will not in general have such details information on the target at the outset of running MCMC. Practical strategies include attempting to approximate the Hessian matrix at a mode of the target, or using a simpler off-the-shelf adaptive MCMC method in which a covariance matrix is iteratively learned (Andrieu and Thoms, 2008). This appendix investigates the performance of RMALA when G_0 is also to be estimated. To this end, we reproduced the experiment from Table 1 in the main text but with G_0 set equal to the negative Hessian of $\log p$ evaluated at the initial state x_0 of the Markov chain. Full results are displayed in Table 3. In summary, the broad conclusions that we draw in the main text, regarding the potential to improve performance using a position-dependent step size $\epsilon(\cdot)$ trained using RLMH and the CDLB

		Gradient-free RLMH	**RMALA**		**RMALA-RLMH** (proposed)	
Task	d	LESJD	AAR	ESJD	LESJD	CDLB
earnings-earn_height	3	1.8(0.1)E-1	5.2(0.3)E-1	5.0(0.0)E-1	**1.4(0.1)E-2**	1.5(0.1)E-2
earnings-log10earn_height	3	1.6(0.0)E-1	1.4(0.0)E-2	1.5(0.1)E-2	1.4(0.1)E-2	**1.3(0.0)E-2**
earnings-logearn_height	3	1.6(0.0)E-1	1.5(0.0)E-2	1.6(0.1)E-2	**1.3(0.1)E-2**	1.4(0.0)E-2
gp_pois_regr-gp_regr	3	1.2(0.0)E-1	**2.2(0.1)E-2**	2.3(0.1)E-2	2.4(0.1)E-2	2.4(0.1)E-2
kidiq-kidscore_momhs	3	1.5(0.0)E-1	1.4(0.1)E-2	1.5(0.1)E-2	**1.2(0.1)E-2**	**1.2(0.0)E-2**
kidiq-kidscore_momiq	3	1.7(0.0)E-1	1.5(0.1)E-2	1.5(0.1)E-2	**1.2(0.0)E-2**	1.4(0.1)E-2
kilpisjarvi_mod-kilpisjarvi	3	1.7(0.0)E-1	5.9(0.3)E-1	6.0(0.3)E-1	**2.5(0.7)E-2**	**2.5(0.6)E-2**
mesquite-logmesquite_logvolume	3	1.3(0.0)E-1	2.2(0.1)E-2	2.4(0.1)E-2	**1.9(0.1)E-2**	2.0(0.1)E-2
arma-arma11	4	1.2(0.0)E-1	2.3(0.1)E-2	2.5(0.1)E-2	2.1(0.1)E-2	**2.0(0.1)E-2**
earnings-logearn_height_male	4	1.6(0.0)E-1	1.4(0.1)E-2	1.6(0.1)E-2	**1.3(0.1)E-2**	**1.3(0.1)E-2**
earnings-logearn_logheight_male	4	1.6(0.0)E-1	1.3(0.1)E-2	1.5(0.1)E-2	**1.1(0.1)E-2**	1.2(0.1)E-2
garch-garch11	4	1.4(0.0)E-1	**1.4(0.1)E-2**	1.7(0.1)E-2	-	8.9(7.2)E-2
hmm_example-hmm_example	4	1.3(0.0)E-1	2.1(0.1)E-2	2.2(0.1)E-2	**1.7(0.1)E-2**	1.9(0.1)E-2
kidiq-kidscore_momhsiq	4	1.4(0.0)E-1	1.6(0.1)E-2	1.8(0.1)E-2	**1.2(0.1)E-2**	1.7(0.1)E-2
earnings-logearn_interaction	5	1.4(0.0)E-1	1.9(0.1)E-2	2.1(0.1)E-2	**1.7(0.1)E-2**	1.8(0.1)E-2
earnings-logearn_interaction_z	5	1.2(0.0)E-1	2.3(0.1)E-2	2.4(0.1)E-2	**2.1(0.1)E-2**	**2.1(0.1)E-2**
kidiq-kidscore_interaction	5	1.7(0.1)E-1	1.5(0.1)E-2	1.7(0.1)E-2	**1.3(0.1)E-2**	1.5(0.1)E-2
kidiq_with_mom_work-kidscore_interaction_c	5	1.6(0.0)E-1	1.7(0.1)E-2	1.8(0.1)E-2	**1.4(0.1)E-2**	1.5(0.1)E-2
kidiq_with_mom_work-kidscore_interaction_c2	5	1.7(0.0)E-1	1.9(0.1)E-2	2.0(0.1)E-2	**1.6(0.1)E-2**	**1.6(0.1)E-2**
kidiq_with_mom_work-kidscore_interaction_z	5	1.5(0.1)E-1	2.2(0.1)E-2	2.3(0.1)E-2	**1.9(0.1)E-2**	**1.9(0.1)E-2**
kidiq_with_mom_work-kidscore_mom_work	5	1.8(0.1)E-1	2.3(0.1)E-2	2.4(0.1)E-2	**1.9(0.1)E-2**	2.0(0.1)E-2
low_dim_gauss_mix-low_dim_gauss_mix	5	1.1(0.0)E-1	2.6(0.1)E-2	2.8(0.1)E-2	**2.4(0.1)E-2**	2.5(0.1)E-2
mesquite-logmesquite_logva	5	1.2(0.0)E-1	2.4(0.1)E-2	2.6(0.1)E-2	**2.0(0.1)E-2**	2.2(0.1)E-2
bball_drive_event_0-hmm_drive_0	6	1.6(0.3)E-1	2.2(0.1)E-2	2.3(0.1)E-2	**1.8(0.1)E-2**	1.9(0.1)E-2
sblrc-blr	6	1.7(0.0)E-1	1.6(0.1)E-2	1.7(0.1)E-2	**1.4(0.1)E-2**	**1.4(0.1)E-2**
sblri-blr	6	1.7(0.0)E-1	1.6(0.1)E-2	1.8(0.1)E-2	**1.4(0.1)E-2**	1.5(0.1)E-2
arK-arK	7	1.1(0.0)E-1	2.7(0.1)E-2	2.9(0.1)E-2	**2.4(0.1)E-2**	**2.4(0.1)E-2**
mesquite-logmesquite_logvash	7	1.1(0.0)E-1	2.5(0.1)E-2	2.6(0.1)E-2	**2.1(0.1)E-2**	2.3(0.1)E-2
mesquite-logmesquite	8	1.1(0.0)E-1	3.0(0.1)E-2	3.2(0.1)E-2	**2.6(0.1)E-2**	2.8(0.1)E-2
mesquite-logmesquite_logvas	8	1.1(0.0)E-1	3.0(0.1)E-2	3.3(0.1)E-2	**2.4(0.1)E-2**	2.7(0.1)E-2
mesquite-mesquite	8	5.1(1.1)E-1	3.2(0.2)E-2	3.4(0.1)E-2	**2.7(0.0)E-2**	2.9(0.1)E-2
eight_schools-eight_schools_centered	10	7.7(1.2)E-1	6.3(0.7)E-1	5.2(0.8)E-1	**3.0(0.3)E-1**	3.3(0.4)E-1
eight_schools-eight_schools_noncentered	10	1.2(0.0)E0	1.1(0.0)E0	**8.1(0.0)E-1**	8.5(0.0)E-1	8.5(0.0)E-1
nes1972-nes	10	1.1(0.0)E-1	2.7(0.1)E-2	2.9(0.1)E-2	**2.4(0.1)E-2**	2.5(0.1)E-2
nes1976-nes	10	1.1(0.0)E-1	2.9(0.1)E-2	3.0(0.1)E-2	**2.6(0.1)E-2**	2.7(0.1)E-2
nes1980-nes	10	1.1(0.0)E-1	3.0(0.1)E-2	3.1(0.1)E-2	**2.7(0.1)E-2**	2.8(0.1)E-2
nes1984-nes	10	1.2(0.0)E-1	3.0(0.1)E-2	3.1(0.1)E-2	**2.7(0.1)E-2**	**2.7(0.1)E-2**
nes1988-nes	10	1.1(0.0)E-1	2.8(0.1)E-2	2.9(0.1)E-2	**2.6(0.1)E-2**	**2.6(0.1)E-2**
nes1992-nes	10	1.1(0.0)E-1	2.8(0.1)E-2	2.9(0.1)E-2	**2.6(0.1)E-2**	**2.6(0.1)E-2**
nes1996-nes	10	1.1(0.0)E-1	2.7(0.1)E-2	2.8(0.1)E-2	**2.6(0.1)E-2**	2.7(0.1)E-2
nes2000-nes	10	1.2(0.0)E-1	2.8(0.1)E-2	2.8(0.1)E-2	**2.4(0.1)E-2**	2.5(0.1)E-2
gp_pois_regr-gp_pois_regr	13	1.5(0.2)E0	-	-	-	**8.7(0.0)E-1**
diamonds-diamonds	26	2.0(0.1)E0	6.1(0.8)E-2	**5.3(0.6)E-2**	6.2(0.6)E-2	5.8(0.7)E-2
mcycle_gp-accel_gp	66	1.9(0.0)E0	**7.6(0.0)E-1**	7.7(0.0)E-1	-	7.7(0.0)E-1

Table 2: Benchmarking using `posteriordb`. Here we compared standard RMALA with constant step-size $\epsilon \in (0, \infty)$ tuned either by matching the average acceptance rate (AAR) to 0.574 or by maximising the expected squared jump distance (ESJD), to RMALA with position-dependent step size $\epsilon(\cdot)$ learned using RLMH, based either using the LESJD reward of Wang et al. (2025) or the proposed CDLB. The gradient-free RLMH method of Wang et al. (2025) is also included as a benchmark. Performance was measured using maximum mean discrepancy (MMD) based on the Gaussian kernel with lengthscale selected using the median heuristic applied to $10,000$ gold-standard samples from the target, and d is the dimension of the target. Results are based on an average of 10 replicates, with standard errors (in parentheses) reported. The method with smallest average MMD is highlighted in **bold**, gray rows indicate tasks for which RLMH performed best, and dashes indicate where catastrophic failure occurred during training of RLMH.

reward, continue to hold, indicating insensitivity of these conclusions to the precise mechanism used to select G_0.

E.5 Additional Results for the Barker Proposal

The *Barker proposal* was developed in Livingstone and Zanella (2022) as an alternative to MALA that trades efficiency for being more easily tuned. It is based on the accept-reject rule of Barker (1965), stated for d-

		RMALA		RMALA-RLMH (proposed)	
Task	d	AAR	ESJD	LESJD	CDLB
earnings-earn_height	3	5.2(0.3)E-1	1.9(0.2)E-2	1.5(0.1)E-2	**1.4(0.2)E-2**
earnings-log10earn_height	3	1.4(0.0)E-2	1.6(0.1)E-2	1.4(0.1)E-2	**1.3(0.1)E-2**
earnings-logearn_height	3	1.5(0.0)E-2	1.6(0.1)E-2	**1.4(0.1)E-2**	**1.4(0.1)E-2**
gp_pois_regr-gp_regr	3	**2.2(0.1)E-2**	2.3(0.1)E-2	2.4(0.1)E-2	2.4(0.1)E-2
kidiq-kidscore_momhs	3	1.4(0.1)E-2	1.5(0.1)E-2	**1.2(0.1)E-2**	**1.2(0.1)E-2**
kidiq-kidscore_momiq	3	1.4(0.0)E-2	1.6(0.1)E-2	**1.1(0.1)E-2**	1.4(0.1)E-2
kilpisjarvi_mod-kilpisjarvi	3	5.9(0.3)E-1	**2.0(0.4)E-2**	2.5(0.7)E-2	2.5(0.7)E-2
mesquite-logmesquite_logvolume	3	2.3(0.1)E-2	2.4(0.1)E-2	**1.9(0.1)E-2**	2.0(0.1)E-2
arma-arma11	4	2.3(0.1)E-2	2.5(0.1)E-2	**2.1(0.1)E-2**	**2.1(0.1)E-2**
earnings-logearn_height_male	4	1.4(0.1)E-2	1.5(0.1)E-2	**1.2(0.1)E-2**	1.3(0.0)E-2
earnings-logearn_logheight_male	4	1.3(0.1)E-2	1.4(0.0)E-2	**1.1(0.1)E-2**	1.2(0.1)E-2
garch-garch11	4	**1.8(0.1)E-2**	2.0(0.1)E-2	-	1.9(0.2)E-2
hmm_example-hmm_example	4	2.1(0.1)E-2	2.2(0.1)E-2	**1.8(0.1)E-2**	1.9(0.1)E-2
kidiq-kidscore_momhsiq	4	1.6(0.1)E-2	1.8(0.1)E-2	**1.3(0.1)E-2**	1.7(0.1)E-2
one_comp_mm_elim_abs-one_comp_mm_elim_abs	4	2.4(0.1)E-2	2.6(0.1)E-2	**1.2(0.8)E-1**	2.0(0.2)E-2
earnings-logearn_interaction	5	1.9(0.1)E-2	2.0(0.1)E-2	**1.7(0.1)E-2**	**1.7(0.1)E-2**
earnings-logearn_interaction_z	5	2.3(0.1)E-2	2.4(0.1)E-2	**2.1(0.1)E-2**	**2.1(0.1)E-2**
kidiq-kidscore_interaction	5	1.5(0.1)E-2	1.7(0.1)E-2	**1.3(0.1)E-2**	1.5(0.1)E-2
kidiq_with_mom_work-kidscore_interaction_c	5	1.7(0.1)E-2	1.8(0.1)E-2	**1.4(0.1)E-2**	1.5(0.1)E-2
kidiq_with_mom_work-kidscore_interaction_c2	5	1.9(0.1)E-2	2.0(0.1)E-2	**1.6(0.1)E-2**	**1.6(0.1)E-2**
kidiq_with_mom_work-kidscore_interaction_z	5	2.2(0.1)E-2	2.3(0.1)E-2	**1.9(0.1)E-2**	**1.9(0.1)E-2**
kidiq_with_mom_work-kidscore_mom_work	5	2.3(0.1)E-2	2.4(0.1)E-2	**1.9(0.1)E-2**	2.0(0.1)E-2
low_dim_gauss_mix-low_dim_gauss_mix	5	2.7(0.1)E-2	2.8(0.1)E-2	**2.4(0.1)E-2**	**2.4(0.1)E-2**
mesquite-logmesquite_logva	5	2.4(0.1)E-2	2.6(0.1)E-2	**2.1(0.1)E-2**	2.2(0.1)E-2
bball_drive_event_0-hmm_drive_0	6	2.3(0.1)E-2	2.4(0.1)E-2	**1.9(0.1)E-2**	2.0(0.1)E-2
bball_drive_event_1-hmm_drive_1	6	1.9(0.1)E-2	2.0(0.1)E-2	**1.6(0.1)E-2**	1.9(0.1)E-2
sblrc-blr	6	1.5(0.1)E-2	1.8(0.2)E-2	**1.4(0.1)E-2**	**1.4(0.1)E-2**
sblri-blr	6	1.6(0.1)E-2	1.7(0.1)E-2	**1.4(0.1)E-2**	**1.4(0.1)E-2**
arK-arK	7	2.7(0.1)E-2	2.9(0.1)E-2	**2.4(0.1)E-2**	2.5(0.1)E-2
mesquite-logmesquite_logvash	7	2.5(0.1)E-2	2.7(0.1)E-2	**2.2(0.1)E-2**	2.3(0.1)E-2
hudson_lynx_hare-lotka_volterra	8	3.0(0.1)E-2	3.4(0.1)E-2	**2.3(0.1)E-2**	2.8(0.1)E-2
mesquite-logmesquite	8	3.1(0.1)E-2	3.3(0.1)E-2	**2.7(0.1)E-2**	3.0(0.1)E-2
mesquite-logmesquite_logvas	8	3.1(0.1)E-2	3.3(0.1)E-2	**2.6(0.2)E-2**	3.0(0.1)E-2
mesquite-mesquite	8	3.3(0.2)E-2	3.4(0.2)E-2	3.0(0.0)E-2	**2.9(0.2)E-2**
eight_schools-eight_schools_centered	10	6.3(0.7)E-1	**5.2(0.8)E-1**	5.7(0.4)E-1	7.7(0.6)E-1
eight_schools-eight_schools_noncentered	10	**6.2(0.0)E-1**	6.3(0.0)E-1	6.3(0.0)E-1	6.3(0.0)E-1
nes1972-nes	10	2.7(0.1)E-2	2.8(0.1)E-2	**2.4(0.1)E-2**	2.5(0.1)E-2
nes1976-nes	10	2.9(0.1)E-2	2.9(0.1)E-2	**2.5(0.1)E-2**	2.6(0.1)E-2
nes1980-nes	10	3.1(0.1)E-2	3.1(0.1)E-2	**2.7(0.1)E-2**	2.8(0.1)E-2
nes1984-nes	10	3.0(0.1)E-2	3.0(0.1)E-2	**2.7(0.1)E-2**	**2.7(0.1)E-2**
nes1988-nes	10	2.8(0.1)E-2	2.8(0.1)E-2	**2.5(0.1)E-2**	**2.5(0.1)E-2**
nes1992-nes	10	2.8(0.1)E-2	2.9(0.1)E-2	**2.5(0.1)E-2**	2.6(0.1)E-2
nes1996-nes	10	2.8(0.1)E-2	2.8(0.1)E-2	**2.5(0.1)E-2**	2.6(0.1)E-2
nes2000-nes	10	2.8(0.1)E-2	2.9(0.1)E-2	**2.4(0.1)E-2**	**2.4(0.1)E-2**
gp_pois_regr-gp_pois_regr	13	-	-	**1.1(0.0)E0**	8.3(1.1)E-1
diamonds-diamonds	26	5.3(0.6)E-2	**4.7(0.4)E-2**	5.8(0.6)E-2	5.6(0.5)E-2
mcycle_gp-accel_gp	66	**7.6(0.0)E-1**	7.7(0.0)E-1	-	8.2(0.5)E-1

Table 3: Benchmarking using `posteriordb`. Here we compared standard RMALA with constant step-size $\epsilon \in (0, \infty)$ tuned either by matching the average acceptance rate (AAR) to 0.574 or by maximising the expected squared jump distance (ESJD), to RMALA with position-dependent step size $\epsilon(\cdot)$ learned using RLMH, based either using the LESJD reward of Wang et al. (2025) or the proposed CDLB. In contrast to the experimental study reported in the main text, here the matrix G_0 was selected as the negative Hessian matrix at the initial state of the Markov chain. Performance was measured using maximum mean discrepancy (MMD) based on the Gaussian kernel with lengthscale selected using the median heuristic applied to $10,000$ gold-standard samples from the target, and d is the dimension of the target. Results are based on an average of 10 replicates, with standard errors (in parentheses) reported. The method with smallest average MMD is highlighted in **bold**, gray rows indicate tasks for which RLMH performed best, and dashes indicate where catastrophic failure occurred during training of RLMH.

dimensional distributions in Algorithm 3.

To investigate the potential of RLMH in the setting of the Barker proposal, we consider allowing the proposal variance σ in Algorithm 3 to be state-dependent. That is, the parameter $\sigma \equiv \sigma(x)$ will depend on the current state x of the Markov chain; we set $\sigma(x) = \sigma_\theta(x)$ where $\sigma_\theta : \mathbb{R}^d \to \mathbb{R}$ is a neural network to be trained using

Algorithm 3 Barker proposal on $\mathbb{R}^d$; Algorithm 1 of Livingstone and Zanella, 2022.

Require: $x \in \mathbb{R}^d$ (current state), scale $\sigma > 0$ (scale).
1: **for** $i \in \{1, \dots, d\}$ **do**
2: Draw $Z_i \sim \mathcal{N}(0, \sigma^2)$.
3: Set $b_i = 1$ with probability

$$\frac{1}{1 + e^{-Z_i \partial_i \log p(x)}}$$

else set $b_i = -1$.
4: Set $X_i^* = x_i + b_i Z_i$.
5: **end for**
6: **return** proposed state $X^* \in \mathbb{R}^d$.

Algorithm 4 Metropolis–Hastings with the state-dependent Barker proposal on $\mathbb{R}^d$. [Here $\phi(x)$ is the probability density function of the standard normal distribution on $\mathbb{R}$.]

Require: $x_0 \in \mathbb{R}^d$ (initial state), $\sigma(\cdot) > 0$ (and state-dependent scale), $n \in \mathbb{N}$ (number of iterations).
1: $X_0 \leftarrow x_0$ ▷ initialise Markov chain
2: **for** $t = 0, 1, 2, \dots, N-1$ **do**
3: Given the current state X_i, sample X_{i+1}^* using Algorithm 3 with scale $\sigma(X_i)$.
4: Set $X_{i+1} \leftarrow X_{i+1}^*$ with probability

$$\min\left(1, \frac{p(X_{i+1}^*)}{p(X_i)} \times \prod_{j=1}^{d} \frac{\frac{1}{\sigma(X_{i+1}^*)}\phi\left(\frac{X_{i,j} - X_{i+1,j}^*}{\sigma(X_{i+1}^*)}\right)}{1 + e^{(X_{i+1,j}^* - X_{i,j})\partial_j \log p(X_{i+1}^*)}} \Bigg/ \frac{\frac{1}{\sigma(X_i)}\phi\left(\frac{X_{i+1,j}^* - X_{i,j}}{\sigma(X_i)}\right)}{1 + e^{(X_{i,j} - X_{i+1,j}^*)\partial_j \log p(X_i)}}\right)$$

else set set $X_{i+1} \leftarrow X_i$
5: **end for**
6: **return** Markov chain $\{X_0, \dots, X_n\}$.

RL. That is, we have an MDP with action of the form $a_n = [\sigma_\theta(X_n), \sigma_\theta(X_{n+1}^*)]$. The full formulation of the Metropolis–Hastings algorithm based on the state-dependent Barker proposal is clarified in Algorithm 4.

For RLMH we employed the same settings as used for RMALA; no fine-tuning was performed. As a baseline we simply considered the constant step size $\epsilon = \epsilon^\dagger$ described in Appendix D.2. Results in Table 4 indicate little benefit from RLMH in this setting; this makes sense and is consistent with the goal of the Barker proposal to be robust to how the step size is selected. However, our results do not rule out a benefit from using RLMH in the context of the Barker proposal, since we did not fine tune settings for RLMH.

Task	d	**Barker** Constant	**Barker-RLMH** LESJD	CDLB
earnings-earn_height	3	5.0(0.0)E-1	5.0(0.0)E-1	5.0(0.0)E-1
earnings-log10earn_height	3	5.5(0.0)E-1	5.5(0.0)E-1	5.5(0.0)E-1
earnings-logearn_height	3	1.1(0.0)E0	1.1(0.0)E0	1.1(0.0)E0
gp_pois_regr-gp_regr	3	1.1(0.1)E-2	**2.5(0.3)E-3**	2.6(0.2)E-3
kidiq-kidscore_momhs	3	4.0(0.8)E-2	**3.6(1.4)E-2**	3.9(0.9)E-2
kidiq-kidscore_momiq	3	4.8(0.2)E-1	**4.6(0.2)E-1**	4.7(0.2)E-1
kilpisjarvi_mod-kilpisjarvi	3	6.3(0.0)E-1	6.3(0.0)E-1	6.3(0.0)E-1
mesquite-logmesquite_logvolume	3	4.9(0.6)E-1	**3.2(0.7)E-1**	3.7(0.6)E-1
arma-arma11	4	10.0(0.0)E-1	10.0(0.0)E-1	10.0(0.0)E-1
earnings-logearn_height_male	4	4.9(0.0)E-1	4.9(0.0)E-1	4.9(0.0)E-1
earnings-logearn_logheight_male	4	6.5(0.0)E-1	6.5(0.0)E-1	6.5(0.0)E-1
garch-garch11	4	3.7(0.5)E-2	**1.6(0.2)E-2**	2.1(0.5)E-2
hmm_example-hmm_example	4	5.6(0.9)E-1	**3.1(0.6)E-1**	4.2(0.7)E-1
kidiq-kidscore_momhsiq	4	1.2(0.0)E0	1.2(0.0)E0	1.2(0.0)E0
one_comp_mm_elim_abs-one_comp_mm_elim_abs	4	**1.1(0.4)E-1**	7.4(0.0)E-1	7.4(0.0)E-1
earnings-logearn_interaction	5	6.7(0.0)E-1	6.7(0.0)E-1	6.7(0.0)E-1
earnings-logearn_interaction_z	5	6.7(0.0)E-1	6.7(0.0)E-1	6.7(0.0)E-1
kidiq-kidscore_interaction	5	1.0(0.0)E0	1.0(0.0)E0	1.0(0.0)E0
kidiq_with_mom_work-kidscore_interaction_c	5	9.1(0.8)E-1	**6.0(0.7)E-1**	6.7(0.7)E-1
kidiq_with_mom_work-kidscore_interaction_c2	5	7.8(0.6)E-1	**5.8(0.7)E-1**	**5.8(0.7)E-1**
kidiq_with_mom_work-kidscore_interaction_z	5	**3.6(0.3)E-2**	4.2(1.1)E-2	3.8(0.8)E-2
kidiq_with_mom_work-kidscore_mom_work	5	1.2(0.3)E-1	**7.4(1.7)E-2**	7.4(1.7)E-2
low_dim_gauss_mix-low_dim_gauss_mix	5	6.9(0.0)E-1	6.9(0.0)E-1	6.9(0.0)E-1
mesquite-logmesquite_logva	5	6.3(0.0)E-1	6.3(0.0)E-1	6.3(0.0)E-1
bball_drive_event_0-hmm_drive_0	6	5.9(0.0)E-1	**5.1(0.4)E-1**	5.4(0.3)E-1
bball_drive_event_1-hmm_drive_1	6	1.1(0.0)E0	1.1(0.0)E0	1.1(0.0)E0
sblrc-blr	6	6.1(0.0)E-1	6.1(0.0)E-1	6.1(0.0)E-1
sblri-blr	6	5.4(0.0)E-1	5.4(0.0)E-1	5.4(0.0)E-1
arK-arK	7	7.2(0.0)E-1	7.2(0.0)E-1	7.2(0.0)E-1
mesquite-logmesquite_logvash	7	8.8(0.0)E-1	8.8(0.0)E-1	8.8(0.0)E-1
hudson_lynx_hare-lotka_volterra	8	9.2(0.0)E-1	9.2(0.0)E-1	9.2(0.0)E-1
mesquite-logmesquite	8	8.7(0.0)E-1	8.7(0.0)E-1	8.7(0.0)E-1
mesquite-logmesquite_logvas	8	7.3(0.0)E-1	7.3(0.0)E-1	7.3(0.0)E-1
mesquite-mesquite	8	8.1(0.1)E-1	8.1(0.1)E-1	8.1(0.1)E-1
eight_schools-eight_schools_centered	10	**4.3(0.5)E-2**	5.4(0.3)E-1	5.4(0.3)E-1
eight_schools-eight_schools_noncentered	10	5.7(0.1)E-1	**5.6(0.0)E-1**	5.7(0.0)E-1
nes1972-nes	10	8.4(0.0)E-1	8.4(0.0)E-1	8.4(0.0)E-1
nes1976-nes	10	7.1(0.0)E-1	7.1(0.0)E-1	7.1(0.0)E-1
nes1980-nes	10	6.4(0.0)E-1	6.4(0.0)E-1	6.4(0.0)E-1
nes1984-nes	10	1.1(0.0)E0	1.1(0.0)E0	1.1(0.0)E0
nes1988-nes	10	6.6(0.0)E-1	6.6(0.0)E-1	6.6(0.0)E-1
nes1992-nes	10	1.2(0.0)E0	1.2(0.0)E0	1.2(0.0)E0
nes1996-nes	10	7.2(0.0)E-1	7.2(0.0)E-1	7.2(0.0)E-1
nes2000-nes	10	1.1(0.0)E0	1.1(0.0)E0	1.1(0.0)E0
gp_pois_regr-gp_pois_regr	13	**8.4(0.6)E-1**	8.7(0.0)E-1	8.7(0.0)E-1
diamonds-diamonds	26	6.8(0.0)E-1	6.8(0.0)E-1	6.8(0.0)E-1
mcycle_gp-accel_gp	66	7.7(0.0)E-1	7.7(0.0)E-1	7.7(0.0)E-1

Table 4: Benchmarking using `posteriordb`. Here we compared standard Barker MCMC with constant step-size ϵ to a position-dependent step size $\epsilon(\cdot)$ learned using RLMH, based either using the LESJD reward of Wang et al. (2025) or the proposed CDLB. Performance was measured using maximum mean discrepancy (MMD) based on the Gaussian kernel with lengthscale selected using the median heuristic applied to $10,000$ gold-standard samples from the target, and d is the dimension of the target. Results are based on an average of 10 replicates, with standard errors (in parentheses) reported. The method with smallest average MMD is highlighted in **bold**, gray rows indicate tasks for which RLMH performed best.