Exploiting Model Errors for Exploration in Model-Based Reinforcement Learning

Jared Swift
King's College London
jared.swift@kcl.ac.uk

Matteo Leonetti
King's College London
matteo.leonetti@kcl.ac.uk

Abstract

We address the problem of exploration in model-based reinforcement learning (MBRL). We present **M**odel-**C**orrective e**X**ploration (MCX) a novel approach to exploration in MBRL that is both agnostic to the model representation and scalable to complex environments. MCX learns to generalise model prediction errors in order to make hypotheses about how the model might else be wrong, and uses such hypotheses for performing planning to facilitate exploration. We demonstrate the efficacy of our method in visual control tasks with the state-of-the-art MBRL algorithm, DreamerV3.

1 Introduction

The ability to efficiently explore an environment is critical to the success of a reinforcement learning (RL) agent, and it has been an intensively studied area since the inception of RL. When no knowledge about the environment is available, most methods rely almost entirely on randomness, the prototypical example being ϵ-greedy. In model-based RL (MBRL), the agent learns an explicit representation of the world, in the form of a model, and exploration strategies can therefore leverage such knowledge. The predominant insight underpinning model-based exploration is to use uncertainty within said model to guide the agent toward parts of the state space in which the model is less confident. The prototypical example is Upper Confidence Bounds (Auer et al., 2002) and its adaptations to Markov Decision Processes (MDPs) (Auer and Ortner, 2006).

We explore a third reason to try an action—not randomly, and not because the model has low confidence (but high hope) in its outcome—the agent makes hypotheses based on planning, computing that if the action had a different but *reasonable* outcome, it would lead to higher value, and therefore it is worth trying. Without a definition of a *reasonable* hypothesis, the agent would always hypothesise that every action leads directly to the state of highest reward, and the method would explore similarly to R-MAX (Brafman and Tennenholtz, 2002). To determine what is reasonable to try, in this context, the agent learns a *correction* model, that is, a model that learns ways in which the primary model has been mistaken before, and it uses it to hypothesise that it could be similarly mistaken elsewhere.

We refer to this approach as **M**odel-**C**orrective e**X**ploration (MCX). MCX generalises observed model errors into hypotheses about other potential inaccuracies in the model, which guide exploration towards regions where correcting such errors could yield higher returns. To the best of our knowledge, this is the first exploration method to exploit model errors in this manner.

To illustrate the core ideas behind MCX, consider the grid world in Figure 1a, which also shows the trajectory induced by an optimal policy. The agent starts in the bottom left and must reach the goal state, shaded in green. The gray shaded cells represent obstacles, which the agent cannot enter. The red sinusoidal lines represent an upwind, which allows the agent to move two cells upwards instead of one. The agent's initial model is shown in Figure 1b, along with the trajectory induced by a corresponding optimal policy. Initially, the agent is unaware of the wind, and thus its model is

18th European Workshop on Reinforcement Learning (EWRL 2025).

incorrect and policy suboptimal. Upon executing this policy in the environment, it encounters the upwind on the right side of the grid (Figure 1c), realises that its model is inaccurate, and updates it accordingly (Figure 1d). In the following episode, the agent again finds itself in the bottom left cell (Figure 1a). At this point, its model correctly represents the wind on the right-hand side of the grid, but the trajectory induced by the policy remains mostly unchanged. However, it recalls when previously its model was incorrect about the presence of the wind, and hypothesises a correction to the model that incorporates the wind in the start state. If this correction were true, it would accommodate a shorter path (Figure 1e). The agent acts on this hypothesis and discovers that it is actually true (Figure 1f), it then updates its model accordingly (Figure 1g)).

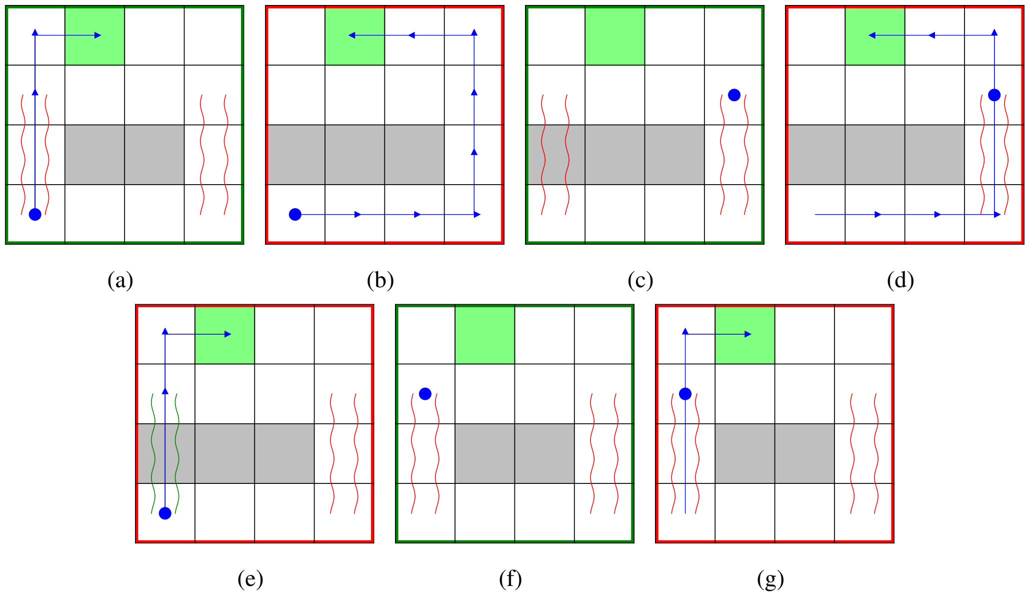


Figure 1: 1a, 1c and 1f, with green borders, are environment representations. 1b, 1d, 1e and 1g, with red borders, are model representations. The agent is represented by a blue circle. The shaded cells are obstacles, which the agent cannot enter. The red sinusoidal lines represent an upwind, which allows the agent to move up two cells instead of one. The goal is represented by the shaded green cell. The reward function encourages the agent to find the shortest path. The blue arrows represent the trajectory induced by the optimal policy. Hypotheses are indicated in green.

The main contribution of this work is a novel method for exploration in model-based RL, which uses observed model errors, rather than uncertainty, to explicitly generate hypotheses about how the model may further be incorrect, and direct exploration. While, in this paper, we evaluate the effect of this strategy in isolation, it is, in principle, possible to combine it with other forms of directed exploration, such as intrinsic motivation and uncertainty-based exploration. Additionally, we provide a formulation for how this method may be implemented with a state-of-the-art model-based RL algorithm, DreamerV3 (Hafner et al., 2025), and provide experimental results demonstrating its effectiveness. We evaluate our method on a set of continuous control tasks from pixel observations; our results show that exploiting corrections in this way improves sample efficiency and, in some cases, asymptotic performance, and in general does not degrade performance.

2 Preliminaries

2.1 Reinforcement learning

We consider the canonical RL setting, modelled as a Markov Decision Process (MDP). An MDP is characterised by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma, \mu_0 \rangle$. Where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces respectively, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function, where $P(s' \mid s, a)$ denotes the probability of transitioning to state s' after taking action a in state s. $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, which returns a scalar reward signal r received after taking action a in state s. $\gamma \in [0, 1)$ represents the discount factor and $\mu_0 : \mathcal{S} \rightarrow [0, 1]$ denotes the start state distribution, that is the probability of the agent starting an episode in a state $s \in \mathcal{S}$. The agent's behaviour is governed by a probabilistic policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, which maps states to a probability distribution over actions. The goal of the agent is to learn an optimal policy π^* that maximises the expected return, $G_t = \mathbb{E}_\pi \left[\sum_{k=t+1}^{T} \gamma^{k-t-1} r_k \right]$. The value function $V^\pi : \mathcal{S} \to \mathbb{R}$ gives the expected return when starting from state s and following policy π. The Q, or action-value, function, $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ gives the expected return when taking action a in state s and following π thereafter.

Within model-based RL (MBRL), the agent learns a model of the environments dynamics and reward function from experience. Formally, it learns a model M, which approximates the transition function and reward function. P_M and R_M represent the transition and reward functions learned under M respectively. M is used for planning, which in the context of MBRL typically involves simulating trajectories under the learned model to evaluate candidate policies (Hafner et al., 2025; Janner et al., 2019).

2.2 DreamerV3

We implement our proposed exploration method within DreamerV3 (Hafner et al., 2025), hereafter referred to as Dreamer, which is a state-of-the-art MBRL algorithm. We now briefly summarise the components of Dreamer that are relevant to our approach. The backbone of Dreamer is a Recurrent State Space Model (RSSM) (Hafner et al., 2019). The RSSM learns a latent state representation $s_t := (h_t, z_t)$ consisting of a deterministic and stochastic component. The deterministic component, h_t, is modelled using a Recurrent Neural Network (RNN), which captures the history:

$$h_{t+1} = f_\phi(h_t, z_t, a_t). \tag{1}$$

The stochastic component, z_t, is modelled by a Multi-Layer Perceptron (MLP) that parametrises a discrete distribution. Dreamer maintains a prior and posterior distribution over z_t. The prior, $\hat{z}_t$ is used for planning within the world model, and is conditioned solely on the history:

$$\hat{z}_{t+1} \sim p_\phi(\hat{z}_{t+1} \mid h_{t+1}), \tag{2}$$

while the posterior, z_t is additionally conditioned on an observation from the environment:

$$z_{t+1} \sim q_\phi(z_{t+1} \mid h_{t+1}, x_{t+1}). \tag{3}$$

Training of the world model encourages the prior to accurately approximate the posterior by minimising the KL divergence between the two. Dreamer uses the prior to learn a policy within the world model and incorporates an entropy regularisation objective to encourage stochasticity in the policy. Dreamer's exploration strategy involves sampling from this learned stochastic policy whilst interacting with the environment.

3 Related work

Exploration in model-based RL has been studied extensively, both theoretically and empirically, with a plethora of methods proposed. We briefly review common approaches to exploration in MBRL and relate them to our proposed method.

Random exploration Despite the vast body of research within exploration in model-based RL, in practice, heuristic methods predominantly based on randomness are employed, such as ϵ-greedy (Sutton and Barto, 2018), additive Gaussian noise (Lillicrap et al., 2016), and entropy regularisation (Haarnoja et al., 2018), combined with millions of environment samples. This is even the case in state-of-the-art model-based RL (Hafner et al., 2025; Janner et al., 2019). Random exploration is undirected, and therefore inherently sample-inefficient, often wasting environment interactions in uninformative states. While randomness is a natural by-product of deep RL with function approximation – such as from stochastic gradient descent – our method does not predominantly rely on it for exploration, instead directing exploration through hypothesised model corrections that have the potential to improve performance.

Greedy exploitation The simplest form of performing directed exploration is to act greedily with respect to the current model (Deisenroth and Rasmussen, 2011; Chua et al., 2018). This process naturally induces exploration, since the model can be arbitrarily wrong at any given time. However, this is not an effective exploration method on its own, and can, in many cases, result in the agent getting stuck in local optima. When the model is perfect, exploration is unnecessary and greedy exploitation is optimal. Therefore, in environments where an accurate model is easy to acquire, greedy policies are sufficient. Our method leverages this observation, and utilises greedy exploitation when hypothesised corrections seem to provide no benefit.

Optimistic exploration Optimistic methods are grounded in the principle of *optimism in the face of uncertainty* (OFU), which assumes that uncertain aspects of the environment are better than they currently appear. In RL, this translates to the estimated value function upper bounding the true value function. Optimistic exploration is central to many theoretical results in MBRL, such as those providing PAC guarantees or regret bounds. R-MAX (Brafman and Tennenholtz, 2002) is a classical method for exploration in MBRL. R-MAX assumes that uncertain transitions, measured through visitation counts, lead to a *Garden of Eden* state that returns the maximal possible reward. This often leads to over-optimism with the agent exploring all possible transitions. Moreover, R-MAX is limited to discrete state and action spaces. Whilst extensions to continuous domains have been proposed (Jong and Stone, 2007; Jung and Stone, 2010), they rely on assumptions of discretisation and local smoothness of dynamics, which limits their applicability. Upper-Confidence RL (UCRL) algorithms (Auer and Ortner, 2006; Bartlett and Tewari, 2009; Jaksch et al., 2010; Filippi et al., 2010; Azar et al., 2017; Fruit et al., 2018; Ayoub et al., 2020) instantiate optimism by constructing upper confidence bounds on the approximated transition function. Policies are computed by maximising expected value jointly over the space of policies and statistically plausible models. Whilst theoretically sound, these methods lack scalability due to the need to maintain state-visitation counts or solve a joint maximisation problem, which becomes intractable beyond tabular settings. H-UCRL (Curi et al., 2020) reduces the UCRL objective to greedy exploitation through a reparameterisation trick: hallucinated actions are introduced for each state dimension, allowing the agent to optimistically select one-step transition dynamics within modelled epistemic uncertainty. Whilst this work scales the UCRL paradigm to the deep MBRL setting, it significantly expands the action space, which hinders applicability in high-dimensional domains. DOVE (Seyde et al., 2020) generalises UCRL to deep MBRL by approximating epistemic uncertainty through disagreement among an ensemble of latent dynamics models – this enables scalability as visitation counts are no longer required. NARL (Pacchiano et al., 2021) similarly applies UCRL to deep MBRL by reducing the continuous uncertainty set of models to a discrete set, consisting of ensemble members. OMBRL (Sukhija et al., 2025) shift uncertainty from the extrinsic reward into an intrinsic reward – eliminating the need to sample from or maximise over model uncertainty, and thus accommodating scalability. Our method is optimistic with respect to observed model errors, rather than explicitly modelled uncertainty. Additionally, unlike typical optimistic approaches, we do not incorporate optimism into the learning objective – thus, there is reduced computational overhead. Rather, we apply one-step optimistic updates based on hypotheses about how the model may be wrong. To the best of our knowledge, this is the first use of optimism driven by hypothesised model error rather than uncertainty estimation.

Intrinsic motivation Intrinsically motivated exploration occurs when the agent optimises an *intrinsic* reward that it generates for itself, rather than directly optimising for the *extrinsic* reward, which comes from the environment. Intrinsic motivation is most frequently deployed in the context of unsupervised MBRL, where the agent must learn a model of an unknown environment that provides no extrinsic reward signal. In this setting, intrinsic motivation can provide incentive to explore beyond taking random actions. However, intrinsic rewards may be deployed in the traditional, supervised RL setting, which we consider within this work. VIME (Houthooft et al., 2016) derives an intrinsic reward using variational inference within a Bayesian MBRL setting, aiming to maximise information gain about the agent's belief of the environment's dynamics. Pathak et al. (2017) propose generating intrinsic rewards through prediction errors between a forward and inverse dynamics model in feature space. However, these intrinsic rewards are used to incentivise exploration in a model-free agent. MAX (Shyam et al., 2019) derives intrinsic reward through a measure of novelty from disagreement among an ensemble of forward models. This results in the agent planning to observe novel events. Similarly, Plan2Explore (Sekar et al., 2020) derives an intrinsic reward from disagreement among an ensemble of one-step latent dynamics models; however, this ensemble is not directly used for planning. LEXA (Mendonca et al., 2021) trains two policies: an *explorer* policy that maximises intrinsic reward – derived from ensemble disagreement – and a *goal-conditioned achiever policy* which is trained to reliably reach states discovered by the explorer. LEXA alternates between collecting data with the exploration policy, and sampling a goal, and collecting data with the achiever policy. PEG (Hu et al.,

2023) is an extension of LEXA, which instead selects goals to maximise expected exploration value, and then applying a Go-Explore-style approach (Ecoffet et al., 2021): first reaching the goal and then exploring from there using the explorer policy. These methods aim to learn an accurate model of the environment by exploring areas of the state space with high uncertainty or model error. This often results in exploring *everywhere*, even areas of the state space irrelevant to any downstream reward function. This is the case regardless of whether intrinsic rewards are used alone or as a bonus. Contrary to such methods, our method doesn't necessarily encourage exploration towards where the model is uncertain or wrong, but instead hypothesises plausible model errors which would benefit the agent. Since these hypotheses are local to the policy, our method could result in under-exploration in some scenarios. Combining intrinsic motivation, as a bonus term, with our method could mitigate this by promoting global exploration.

Posterior sampling Posterior Sampling for Reinforcement Learning (PSRL) (Osband et al., 2013) is a Bayesian approach to exploration in tabular MBRL, with strong theoretical guarantees. PSRL maintains a posterior distribution over plausible models. Each episode, a model is sampled from the posterior which is used for computing a policy. The policy is followed for the episode, and the posterior distribution is updated based on the observed trajectory. This process naturally induces exploration, which decreases over time as more models are sampled. PSRL is difficult to scale to beyond small tabular MDPs since a posterior distribution over potentially complex models needs to be maintained and updated. Moreover, each time a model is sampled, the policy needs to be recomputed, which can be computationally expensive. There have been several works which attempt to scale PSRL beyond tabular settings (Tziortziotis et al., 2013; Fan and Ming, 2021), but they rely on the dynamics being linear in a learned feature space; this limits scalability to high-dimensional or non-linear domains. Most recently, Sasso et al. (2023) proposed PSDRL, which attempts to scale PSRL to the deep MBRL setting. PSDRL combines a neural-linear approach with a latent dynamics model to accommodate efficient sampling of models. Additionally, PSDRL uses a learned value network, which is updated from all sampled models, to bootstrap the planning process for new models, reducing the cost of computing a new policy. Our approach is less computationally intensive than posterior sampling methods. Firstly, it does not require maintaining a posterior distribution over plausible models or model parameters. Secondly, our method does not require planning from scratch at each episode; rather, the model and policy are incrementally updated. Planning within our approach is a one-step process that is performed online, and only in select states. Whilst this does introduce an additional computational overhead, it is less expensive than computing a new policy at every episode.

4 Methodology

In this section we introduce our proposed method for exploration in MBRL, **MCX**. MCX is agnostic to the underlying model representation and supplements the standard MBRL loop with a mechanism for performing directed exploration.

We begin with a high-level overview of the general algorithm (Algorithm 1). The agent proceeds in episodes, collecting trajectories in a replay buffer, D, which is used to learn a primary model, M. The primary model is then used to derive a greedy policy, π^*, which is optimal with respect to said model, and corresponding value function, V^*_M. Within this setup, exploration involves two decisions: *when* to explore, and *how* to explore. For simplicity, in this paper, we use an ϵ-greedy strategy to determine when to explore. The main technical contribution of MCX is in how to explore, which forms the remainder of this section.

To determine *how* to explore, MCX hypothesises plausible ways in which the primary model may be inaccurate, and could be corrected. To constrain such hypotheses (which can otherwise be arbitrarily optimistic, to the point of being ineffective), we learn a *correction model* (Line 1). The correction model is trained exclusively on transitions that the primary model failed to predict correctly. During exploration, the agent identifies, through planning, actions that would yield greater returns if the hypothesised corrections were to be true.

The correction model, denoted $\tilde{M}$, is a one-step dynamics model. To decide on which samples the correction model should be trained, we introduce an indicator function, f, which determines whether the primary model, M, was *incorrect* in its predictions about a particular transition. Transitions which are incorrectly predicted, where f evaluates to 1, are stored in the *correction buffer* (Lines 1 &

Algorithm 1 Episodic **M**odel-**C**orrective e**X**ploration (MCX)

```
1: initialise primary model, $M$, correction model, $\tilde{M}$, policy, $\pi^*$, value function $V^*_M$, fixed-length
   replay buffer, $D$, fixed-length correction buffer, $\tilde{D}$
2: for each episode do
3:   $s \leftarrow \texttt{env.reset}(), \quad \tau \leftarrow \{\}$
4:   while not done do
5:     $a \leftarrow \begin{cases} \texttt{explore}(s), & \text{with probability } \epsilon \\ \pi^*(s), & \text{otherwise} \end{cases}$   {Explore using Equation 1}
6:     $s', r, \texttt{done} \leftarrow \texttt{env.step}(a)$
7:     $\tau.\texttt{append}((s, a, s', r, \texttt{done}))$
8:     $s \leftarrow s'$
9:   end while
10:  $D.\texttt{extend}(\tau)$
11:  Update $M$ using $D$
12:  Update $\pi^*$, $V^*_M$ via rollouts from $M$
13:  $\tilde{D}.\text{extend}(\{t : t \in \tau | f(t) = 1\})$
14:  Update $\tilde{M}$ on $\tilde{D}$
15: end for
```

13), from which the correction model is trained (Line 14). Let $\langle s, a, s' \rangle$ be an observed transition when acting in the environment, then f evaluates to:

$$f = \begin{cases} 1, & \text{if } M \text{ incorrectly predicts } \langle s, a, s' \rangle \\ 0, & \text{otherwise.} \end{cases}$$

The definition of f depends on the model architecture, as *incorrectness* is model-specific. For example, consider a model that parametrises a Gaussian distribution over possible next states, as in MBPO (Janner et al., 2019), a possible instantiation of f is whether or not the observed next state lies within a standard deviation of the mean of the predicted next state distribution. We provide an additional formulation of f specific to DreamerV3 in Section 4.1.

When given the opportunity to explore, the agent plans using the correction model over a one-step horizon, producing a *corrected action*, $\tilde{a}$. To guide this selection, we introduce a corrected Q-value, $Q_{\tilde{M}}(s, a)$, which denotes the expected value of the next state distribution induced by $\tilde{M}$ under π^* in M, for a given state-action pair:

$$Q_{\tilde{M}}(s, a) := \int_{\mathcal{S}} P_{\tilde{M}}(s'|s, a) V^*_M(s') ds'. \tag{4}$$

In other words, $Q_{\tilde{M}}$ is the expected value of taking action a in state s when the first transition takes place according to $\tilde{M}$ and all the following transitions according to M with actions selected by π^*. In practice, we approximate this expectation by sampling from $P_{\tilde{M}}$. The corrected action, $\tilde{a}$, is chosen to maximise the corrected action value function $Q_{\tilde{M}}$:

$$\tilde{a} = \arg\max_{a' \in \mathcal{A}} Q_{\tilde{M}}(s, a'). \tag{5}$$

If the corrected action is strictly better than following the greedy policy, then it is executed, otherwise the greedy action is executed:

$$a = \begin{cases} \tilde{a}, & \text{if } Q_{\tilde{M}}(s, \tilde{a}) > V^*_M(s) \\ \pi^*(s), & \text{otherwise.} \end{cases} \tag{6}$$

This ensures that corrections which are not beneficial are ignored, and only strict policy improvements are enforced. This forms the action selection procedure of MCX (Line 5).

Computationally, MCX adds the learning of the correction model $\tilde{M}$ and the single step optimisation in Equation 5. Both are standard computations for MBRL, and therefore scale to any domain to which the underlying MBRL system also scales.

In summary, MCX is a simple approach to exploration in MBRL which requires minimal changes to the underlying algorithm, assuming only that it learn a model which may be used for planning. MCX trains a *correction model* to generalise from observed transitions which the primary model failed to predict into hypotheses about other potential inaccuracies. It then reasons, through planning, to identify hypotheses that, if true, would be beneficial, and selects exploratory actions accordingly.

4.1 Application to Dreamer

We implement the correction model with Dreamer as a one-step latent dynamics model, which predicts corrections to both the deterministic, h_{t+1}, and stochastic, z_{t+1}, components of the latent state. The deterministic correction model:

$$\tilde{h}_{t+1} = f_\psi(h_t, z_t, a_t),$$

is trained to minimise the mean-squared error between the deterministic correction, $\tilde{h}_{t+1}$, and the next deterministic state, h_{t+1}, produced by the RNN (Equation 1). Additionally, the stochastic correction model:

$$p_\psi(\tilde{z}_{t+1} \mid h_t, z_t, a_t),$$

is trained to minimise the KL divergence between the corrected stochastic distribution, $p_\psi(\tilde{z}_{t+1}|h_t, z_t, a_t)$, and the posterior distribution, $q_\phi(z_{t+1}|h_{t+1}, x_{t+1})$, produced by the encoder (Equation 3). Both the deterministic and stochastic corrections are implemented as MLPs. The full correction loss is:

$$\mathcal{L}_{\text{correction}} = ||\tilde{h}_{t+1} - \text{sg}(h_{t+1})||^2 + D_{KL}(\text{sg}(q_\phi(z_{t+1} \mid h_{t+1}, x_{t+1}))||p_\psi(\tilde{z}_{t+1}|h_t, z_t, a_t)),$$

where $\text{sg}(\cdot)$ is the stop-gradient operator, which prevents backpropagation through the DreamerV3 world model.

Since the dynamics are latent, and there is no ground-truth latent state, the definition of the model being *wrong* is non-trivial to define. However, the dissimilarity between the prior (Equation 2) and the posterior (Equation 3) distributions provide a natural mechanism for evaluating the correctness of predictions made by the model. If the prior produces a distribution that is very different compared to the posterior – which is conditioned on the true observation – this suggests that the model did not capture that transition well.

We consider the set of incorrect transitions to be those that the prior predicts least accurately, that is, those that produce the highest KL divergence. Dreamer trains the world model on batches of size B, which consist of sequences of length T. We select the k transitions with the highest losses in each batch to train the correction model. We consider transition tuples of the form $\nu_{b,t} := \langle s_{b,t}, a_{b,t}, h_{b,t+1}, q_\phi(z_{b,t+1}|h_{b,t+1}, x_{b,t+1})\rangle$. Each tuple comes from the b-th sequence in a batch and contains the latent state and action taken at time step t, as well as the next hidden state and posterior distribution at time $t + 1$. We compute the KL divergence between the prior and posterior for each batch element b and time step t:

$$KL_{b,t} := D_{KL}(q_\phi(z_{b,t}|h_{b,t}, x_{b,t})||p_\phi(z_{b,t}|h_{b,t})).$$

We now define our indicator function as follows:

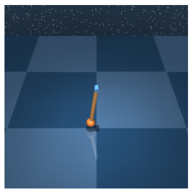
$$f(\nu_{b,t}) = \begin{cases} 1, & \text{if } KL_{b,t} \text{ is in top k } \forall b \in B \text{ and } \forall t \in T \\ 0, & \text{otherwise.} \end{cases}$$

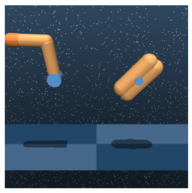
5 Experimental evaluation

We evaluate our method on five domains from the DeepMind control suite. Further information about the experimental setup can be found in Appendix A.1 and A.2.

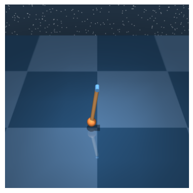
5.1 Environments

The DeepMind control suite (Tunyasuvunakool et al., 2020) consists of visual control tasks of varying difficulties. In these environments, the agent must learn to control robots of various embodiments directly from high-dimensional pixel observations. We now introduce the particular environments that we experiment within. The budget for learning in these environments is 1 million timesteps, with each episode lasting $1,000$ time steps. Additionally, there are no terminal states and the maximal possible cumulative reward per episode is $1,000$.


(a) Acrobot swingup


(b) Finger spin

(c) Hopper hop


(d) Pendulum swingup

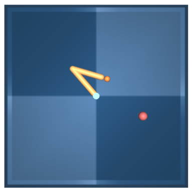

(e) Reacher hard

Figure 2: Five different environments from the DeepMind control suite.

Acrobot swingup This task consists of an underactuated double pendulum, with torques being applied only to the second joint (Firgure 2a). The goal is to swing the pendulum upward and balance vertically. The reward is dense, with the agent receiving a reward of 1 when the system is exactly upright, with the reward decaying smoothly as the pendulum becomes less upright.

Finger spin This task consists of a planar 3-DoF finger, which must spin a rigid body on an unactuated hinge (Figure 2b). The reward is sparse, with the agent only receiving a reward of 1 when the hinge's angular velocity is above a specified threshold.

Hopper hop This task consists of a planar one-legged hopper, which must learn to hop forwards whilst remaining balanced (Figure 2c). The reward is dense, with the agent receiving reward based on it's torso height and forward velocity.

Pendulum swingup This task consists of an inverted pendulum, which must be swung up vertically (Figure 2d). The reward is sparse, with the agent only receiving a reward of 1 when the pole it is within a specified angular distance of the vertical position.

Reacher hard This task consists of a two-link planar robot, which must reach a randomised target in each episode (Figure 2e). The hard variant of the task, which we consider, consists of a smaller target than the easy variant. The reward is sparse, with the agent only receiving a reward of 1 when it comes within a specified distance of the target.

5.2 Results

We compare our approach against Dreamer's built-in exploration mechanism, as detailed in Section 2.2. We adhere to the following evaluation protocol: every $10,000$ environment steps, we freeze the model parameters and perform 10 evaluation episodes, where the agent acts greedily with respect to the learned model's optimal policy. We repeat this process across 10 random seeds. The resultant plots report the mean and standard deviation over these seeds, as shown in Figure 3. Here Dreamer-MCX denotes the implementation of our method with Dreamer. We find that in *Finger spin*, *Reacher hard* and *Pendulum swingup*, Dreamer-MCX outperforms Dreamer. In *Pendulum swingup*, Dreamer-MCX quickly converges to much better performance than Dreamer. In *Hopper hop*, and *Acrobot swingup*, we find that Dreamer-MCX performs on par with Dreamer.

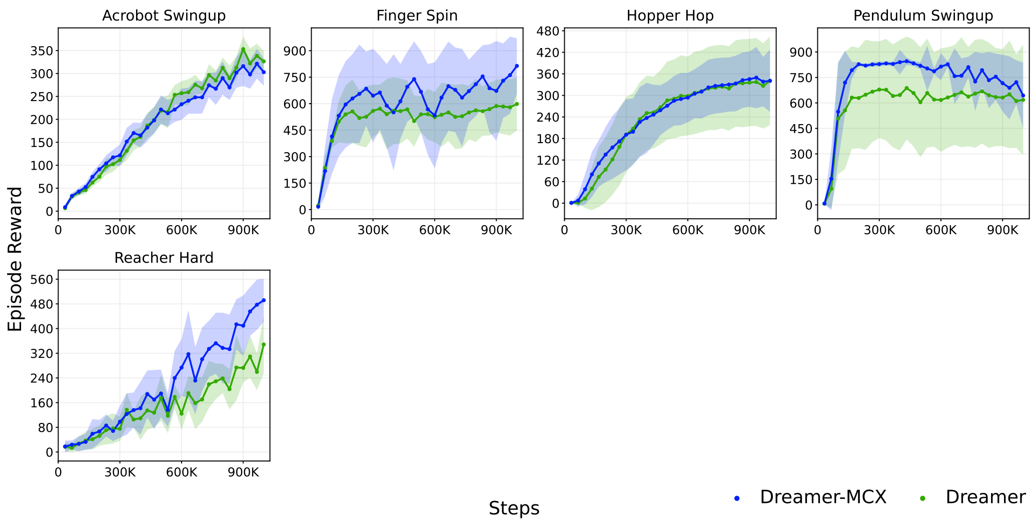


Figure 3: Learning curves for visual control tasks. The x-axis represents environment steps, and the y-axis the mean episode reward over 10 evaluation episodes. Each line is the mean of 10 seeds, and the shaded areas are one standard deviation.

6 Conclusions and future work

We proposed MCX, a novel exploration method for model-based reinforcement learning, which is scalable to complex model architectures and environments. We provided a formulation of MCX within the state-of-the-art model-based RL algorithm DreamerV3, and find that, from our preliminary experiments, it performs on par with or better than DreamerV3's built-in exploration mechanism.

In future work, we will perform further rigorous experiments, across a wider range of domains and model architectures, to further evaluate the benefits of MCX. Additionally, we intend to investigate principled definitions of model correctness, possibly to a wide range of model architectures. We also intend to explore how corrections can be applied to other model components beyond transition dynamics, such as the reward model, as well as principled methods for choosing *when* to explore in this paradigm.

Acknowledgments

This work was supported by UK Research and Innovation [grant number EP/S023356/1], in the UKRI Centre for Doctoral Training in Safe and Trusted Artificial Intelligence (`www.safeandtrustedai.org`). All experiments were performed using King's Computational Research, Engineering and Technology Environment (CREATE). The authors would like to thank Peter Tisnikar for his review of the original manuscript.

References

Peter Auer and Ronald Ortner. Logarithmic online regret bounds for undiscounted reinforcement learning. *Advances in neural information processing systems*, 19, 2006.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256, 2002.

Alex Ayoub, Zeyu Jia, Csaba Szepesvari, Mengdi Wang, and Lin Yang. Model-based reinforcement learning with value-targeted regression. In *International Conference on Machine Learning*, pages 463–474. PMLR, 2020.

Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. Minimax regret bounds for reinforcement learning. In *International conference on machine learning*, pages 263–272. PMLR, 2017.

Peter L Bartlett and Ambuj Tewari. Regal: a regularization based algorithm for reinforcement learning in weakly communicating mdps. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 35–42, 2009.

Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

Sebastian Curi, Felix Berkenkamp, and Andreas Krause. Efficient model-based reinforcement learning through optimistic policy search and planning. *Advances in Neural Information Processing Systems*, 33:14156–14170, 2020.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, 2021.

Ying Fan and Yifei Ming. Model-based reinforcement learning for continuous control with posterior sampling. In *International Conference on Machine Learning*, pages 3078–3087. PMLR, 2021.

Sarah Filippi, Olivier Cappé, and Aurélien Garivier. Optimism in reinforcement learning and kullback-leibler divergence. In *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 115–122. IEEE, 2010.

Ronan Fruit, Matteo Pirotta, Alessandro Lazaric, and Ronald Ortner. Efficient bias-span-constrained exploration-exploitation in reinforcement learning. In *International Conference on Machine Learning*, pages 1578–1586. PMLR, 2018.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, pages 1–7, 2025.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. *Advances in neural information processing systems*, 29, 2016.

Edward S. Hu, Richard Chang, Oleh Rybkin, and Dinesh Jayaraman. Planning goals for exploration. 2023.

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *The Journal of Machine Learning Research*, 11:1563–1600, 2010.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32, 2019.

Nicholas K Jong and Peter Stone. Model-based exploration in continuous state spaces. In *International Symposium on Abstraction, Reformulation, and Approximation*, pages 258–272. Springer, 2007.

Tobias Jung and Peter Stone. Gaussian processes for sample efficient reinforcement learning with rmax-like exploration. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 601–616. Springer, 2010.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

Russell Mendonca, Oleh Rybkin, Kostas Daniilidis, Danijar Hafner, and Deepak Pathak. Discovering and achieving goals via world models. *Advances in Neural Information Processing Systems*, 34: 24379–24391, 2021.

Ian Osband, Daniel Russo, and Benjamin Van Roy. (more) efficient reinforcement learning via posterior sampling. *Advances in Neural Information Processing Systems*, 26, 2013.

Aldo Pacchiano, Philip Ball, Jack Parker-Holder, Krzysztof Choromanski, and Stephen Roberts. Towards tractable optimism in model-based reinforcement learning. In *Uncertainty in Artificial Intelligence*, pages 1413–1423. PMLR, 2021.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

Remo Sasso, Michelangelo Conserva, and Paulo Rauber. Posterior sampling for deep reinforcement learning. In *International Conference on Machine Learning*, pages 30042–30061. PMLR, 2023.

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International conference on machine learning*, pages 8583–8592. PMLR, 2020.

Tim Seyde, Wilko Schwarting, Sertac Karaman, and Daniela Rus. Learning to plan via deep optimistic value exploration. In *Learning for Dynamics and Control*, pages 815–825. PMLR, 2020.

Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *International conference on machine learning*, pages 5779–5788. PMLR, 2019.

Bhavya Sukhija, Lenart Treven, Carmelo Sferrazza, Florian Dorfler, Pieter Abbeel, and Andreas Krause. Optimism via intrinsic rewards: Scalable and principled exploration for model-based reinforcement learning. In *7th Robot Learning Workshop: Towards Robots with Human-Level Abilities*, 2025.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020.

Nikolaos Tziortziotis, Christos Dimitrakakis, and Konstantinos Blekas. Linear bayesian reinforcement learning. In *IJCAI'13: Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1721–1728, 2013.

A Additional Experimental Details

A.1 Dreamer

In our experiments, we use the implementation of DreamerV3 from NM512 (`https://github.com/NM512/dreamerv3-torch`). We use the 12M parameter model size of DreamerV3 – as in the preprint (Hafner et al., 2023) – with an action repeat of 2, replay ratio of 256, 12 parallel environment instances, a batch size of 16 and a batch length of 64.

A.2 Dreamer-MCX

In our experiments, we use $k = 64$ and $\epsilon = 0.5$. Additionally, we use a learning rate of $1e - 3$ for the correction model. The correction model consists of 2 MLPs, each with 2 layers, for the deterministic and stochastic components respectively. Each MLP uses the same number of hidden units as in the 12M parameter model of DreamerV3. Equation 4 is computed through sampling $n = 1000$ latent states, computing their values under the greedy policy, and taking the mean. Equation 5 is computed by $n = 200$ steps of gradient descent, with a learning rate of $1e - 2$.

Our implementation of Dreamer-MCX is open-sourced here: (`https://github.com/jws-1/dreamer-mcx`).