Societal Impact

We do not anticipate immediate negative consequences from conducting this work because our experiments are based on simulation environments designed to conceptually evaluate the capabilities of reinforcement learning (RL) algorithms. Recent studies, however, demonstrate that large-scale RL when integrated with robotics can effectively work on real-world environments (Kalashnikov et al., 2021; Herzog et al., 2023). This makes it crucial to be aware of potential societal harms that RL agents could inadvertently cause. While this work does not aim to address these safety issues, our method might mitigate unintentional harm during the training process. For instance, it might prevent the agent from exhibiting potentially dangerous novelty-seeking behaviors, such as moving robot arms towards low-value, empty regions where a human researcher is likely to be situated.

A Experimental Details

Source code We provide the source code for reproducing our results in the supplementary material.

Compute For MiniGrid experiments, we use a single NVIDIA TITAN Xp GPU and 8 CPU cores for each training run. It takes 15 minutes for training the agent for 1M environment steps. For DeepMind Control Suite and Meta-World experiments, we use a single NVIDIA 2080Ti GPU and 8 CPU cores for each training run. It takes 36 minutes and 90 minutes for training the agent for 100K environment steps on DeepMind Control Suite and Meta-World benchmarks, respectively.

A.1 Implementation Details

Value normalization For normalizing value estimates to stabilize value-conditional state entropy estimation, we compute the mean and standard deviation using the samples within the mini-batch. We empirically find no significant difference to using the running estimate.

Extrinsic critic function For training the extrinsic critic function described in Section 4.2, we introduce another set of critic and target critic functions based on the same hyperparameters used for the main critic the policy aims to maximize. Then we use the target critic for obtaining value estimates. We apply the stop gradient operation to inputs to disable the gradients from updating the extrinsic critic to update other components. For the policy, we use the same policy for training both main and extrinsic critic functions. We empirically find no need for training another policy solely for the extrinsic critic.

A2C implementation details We use the official implementation[5] of RE3 (Seo et al., 2021) and use the same set of hyperparameters unless otherwise specified. Following the setup of RE3, we use a fixed, randomly initialized encoder to extract state representations and use them for computing the intrinsic reward. We use the same hyperparameter of fixed intrinsic scale $\beta = 0.005$ and $k = 5$ for both SE and VCSE following the original implementation. For RE3, we normalize the intrinsic reward with its standard deviation computed using the samples within the mini-batch, following the original implementation. But we do not normalize our VCSE intrinsic reward.

DrQv2 implementation details We use the official implementation[6] of DrQv2 (Yarats et al., 2021a) and use the same set of hyperparameters unless otherwise specified. For both SE and VCSE exploration, we find that using $\beta = 0.1$ achieves the overall best performance. We also use $k = 12$ for both SE and VCSE. For computing the intrinsic reward, we follow the scheme of Laskin et al. (2021) that trains Intrinsic Curiosity Module (ICM; Pathak et al. 2017) upon the representations from a visual encoder and uses ICM features for measuring the distance between states. We note that we detach visual representations used for training ICM to isolate the effect of training additional modules on the evaluation. For both SE and VCSE exploration, we disable the noise scheduling scheme of DrQv2 that decays σ from 1 by following a pre-defined schedule provided by the authors. This is because we find that such a noise scheduling conflicts with the approaches that introduce additional intrinsic rewards. Thus we use the fixed noise of 0.2 for SE and VCSE exploration. For Meta-World

[5] https://github.com/younggyoseo/RE3

[6] https://github.com/facebookresearch/drqv2

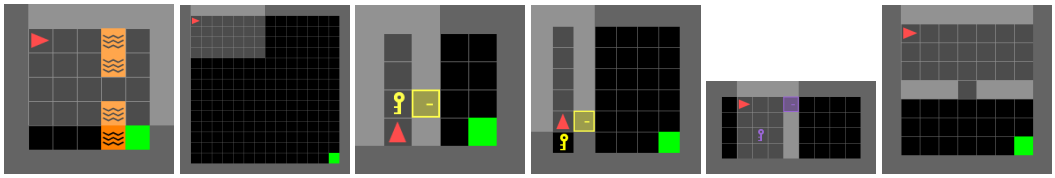


Figure 9: Examples of tasks in our MiniGrid experiments: (a) LavaGapS7, (b) Empty-16×16, (c) DoorKey-6×6, (d) DoorKey-8×8, (e) Unlock, and (f) SimpleCrossingS9N1.

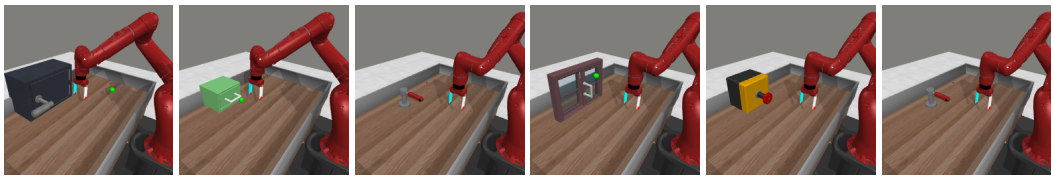


Figure 10: Examples of tasks we used in our Meta-World experiments: (a) Door Open, (b) Drawer Open, (c) Faucet Open, (d) Window Open, (e) Button Press, and (f) Faucet Close.

Figure 11: Examples of tasks we used in our DeepMind Control Suite experiments: (a) Hopper, (b) Walker, (c) Cheetah, (d) Cartpole, (e) Pendulum

experiments, we also disable the scheduling for the DrQv2 baseline as we find it performs better. But we use the original scheduling for the DrQv2 baseline following the official implementation.

Heatmap analysis details For experiments in Figure 8, we use the easy version of SimpleCrossingS9N1 task from MiniGrid benchmark (Chevalier-Boisvert et al., 2018). Specifically, we disable the randomization of map configurations to make it possible to investigate the heatmap over a fixed map. For visualizing the heatmaps, we record x, y position of agents during the initial 100K steps. We train A2C agent with both SE and VCSE exploration as we specified in Section 5.1 and Appendix A.1 without any specific modification for this experiment.

A.2 Environment Details

MiniGrid We conduct our experiments on six navigation tasks from MiniGrid benchmark (Chevalier-Boisvert et al., 2018): LavaGapS7, Empty-16×16, DoorKey-6×6, DoorKey-8×8, Unlock, and SimpleCrossingS9N1. We provide the visualization of the tasks in Figure 9. We use the original tasks without any modification for our experiments in Section 5.1.

Meta-World We conduct our experiments on six manipulation tasks from Meta-World benchmark (Yu et al., 2020): Door Open, Drawer Open, Faucet Open, Window Open, Button Press, and Faucet Close. We provide the visualization of the tasks in Figure 10. We follow the setup of Seo et al. (2022a) that uses a fixed camera location for all tasks.

DeepMind Control Suite We conduct our experiments on six locomotion tasks from DeepMind Control Suite benchmark (Tassa et al., 2020): Hopper Stand, Walker Walk Sparse, Walker Walk, Cheetah Run Sparse, Cartpole Swingup Sparse, and Pendulum Swingup. We use the sparse reward tasks introduced in Seyde et al. (2021), by following RE3. We provide the visualization of the tasks in Figure 10.

B Additional Experiments

B.1 Experiments with Model-Based RL

Setup As a model-based underlying RL algorithm, we consider Masked World Models (MWM; Seo et al. 2022a) that has shown to be able to solve more challenging, long-horizon tasks compared to DrQv2. We consider four tasks of various difficulties: Box Close, Handle Pull Side, Lever Pull, and Drawer Open. We use the official implementation[7] of MWM (Seo et al., 2022a) and use the same set of hyperparameters unless otherwise specified. For both SE and VCSE exploration, we find that using $\beta = 1$ performs best. Following the idea of Seo et al. (2022b) that introduces an additional reward predictor for intrinsic reward in the world model of DreamerV2 (Hafner et al., 2021), we introduce the reward network that predicts our intrinsic reward $r_t^{\texttt{VCSE}}$. For computing the intrinsic reward, we also follow the idea of Seo et al. (2022b) that uses a random projection (Bingham & Mannila, 2001) to reduce the compute cost of measuring distances between states. Specifically, we project 2048-dimensional model states into 256-dimensional vectors with random projection. Because the original MWM implementation normalizes the extrinsic reward by its running estimate of mean to make its scale 1 throughout training, we find that also normalizing intrinsic rewards with their running estimates of mean stabilizes training. We use $k = 12$ for both SE and VCSE.

Results Figure 12 shows that VCSE consistently accelerates and stabilizes the training of MWM agents on four visual manipulation tasks of different horizons and difficulties, which shows that the effectiveness of our method is consistent across diverse types of RL algorithms. On the other hand, we observe that SE could degrade the performance, similar to our observation from experiments with DrQv2 on Meta-World tasks (see Figure 6). This supports our claim that SE often encourages exploration to be biased towards low-value states especially when high-value states are narrowly-distributed, considering that manipulation tasks have a very narrow task-relevant state space.

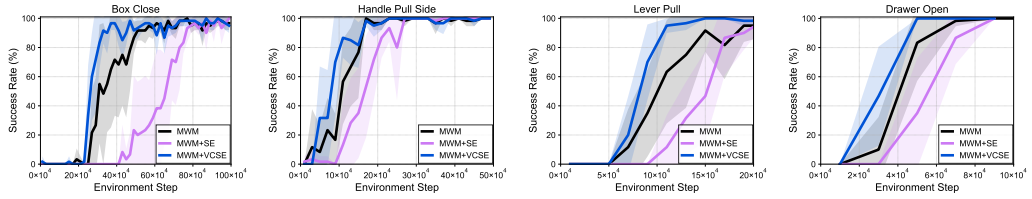
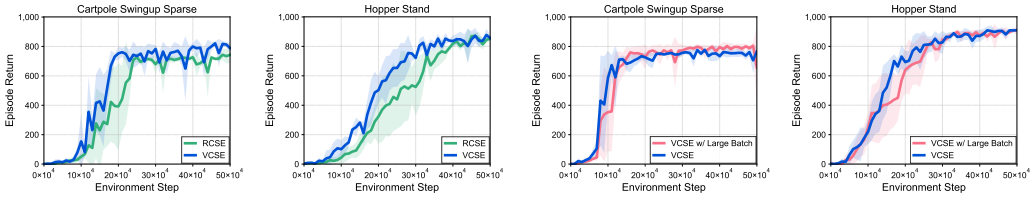


Figure 12: Learning curves on six visual manipulation tasks from Meta-World (Yu et al., 2020) as measured on the success rate. The solid line and shaded regions represent the interquartile mean and standard deviation, respectively, across 16 runs.

B.2 Ablation Study

In Figure 13, we provide the results on individual task used for reporting the aggregate performance that investigate the effect of value conditioning and batch size (see Figure 7b and Figure 7c).



(a) Effect of value conditioning (b) Effect of batch size

Figure 13: Learning curves on two visual locomotion control tasks from DeepMind Control Suite that investigate the effect of (a) value conditioning and (b) batch size. The solid line and shaded regions represent the interquartile mean and standard deviation, respectively, across eight runs.

[7] https://github.com/younggyoseo/MWM

B.3 Experiments with Varying Intrinsic Reward Scale

DrQv2+SE on walker walk with varying β We provide additional experimental results with varying $\beta \in \{0.1, 0.01, 0.001\}$ on Walker Walk task where DrQv2+SE significantly struggles to improve the sample-efficiency of DrQv2. In Figure 14, we find that the performance of DrQv2+SE is consistently worse than the vanilla DrQv2 with different β values. This implies that adding SE intrinsic reward can be sometimes harmful for performance by making it difficult for the agent to exploit the task reward.

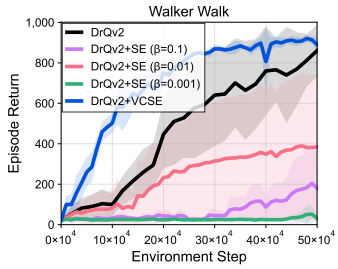


Figure 14: Learning curves as measured on the episode return. The solid line and shaded regions represent the interquartile mean and standard deviation, respectively, across eight runs.

DrQv2+SE with decaying β We conduct additional experiments that compare DrQv2+VCSE with the state entropy baseline that uses decaying schedule for β, similarly to Seo et al. (2021) that uses β schedule for the intrinsic reward. In Figure 15, we find that such a schedule cannot significantly improve the performance of SE, except Hopper Stand where the performance is stabilized. Moreover, we find that the decaying schedule sometimes could degrade the performance, *i.e.,* Walker Walk Sparse. We also note that designing such a decaying schedule is a tedious process that requires researchers to tune the performance, making it less desirable even if it works reasonably well. We indeed find that the performance becomes very sensitive to the magnitude of decaying schedule.[8] On the other hand, DrQv2+VCSE exhibits consistent performance without the decaying schedule, which highlights the benefit of our approach that maximizes the value-conditional state entropy.

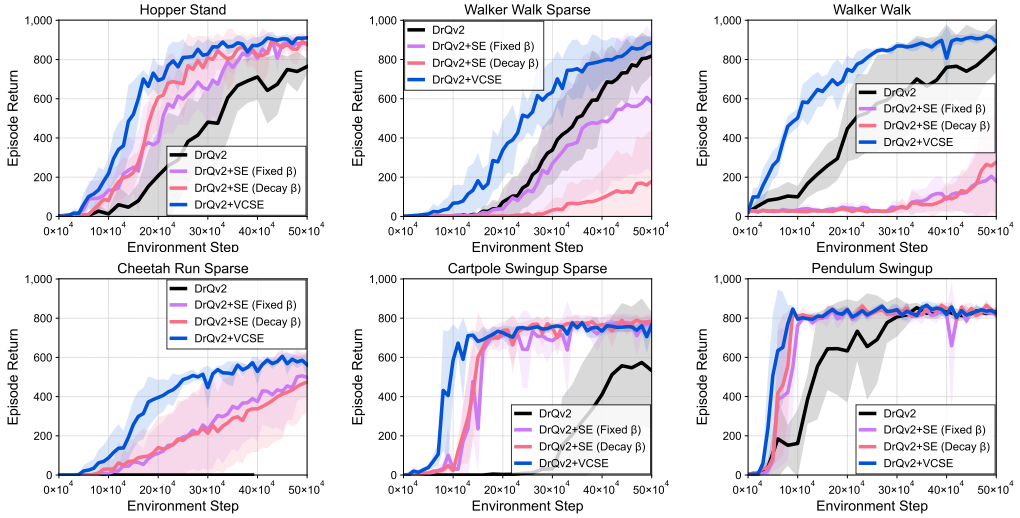


Figure 15: Learning curves on six visual locomotion control tasks from DeepMind Control Suite (Tassa et al., 2020) as measured on the episode return. The solid line and shaded regions represent the interquartile mean and standard deviation, respectively, across eight runs.

[8]Due to the unstable performance from introducing the decay schedule, we run experiments with multiple decaying schedules and report the best performance for each task.

C Experiments with Ground-Truth States

C.1 MiniGrid Experiments

To demonstrate that our method also works in fully-observable MiniGrid where we do not use state encoder for computing the intrinsic bonus, we provide additional experiments that use fully observable states instead of partially observable grid encoding (see Section 5.1). Specifically, we use a set of one-hot vectors that represents a current map as inputs to the agent, and use the location of agent as inputs for computing the intrinsic bonus. Figure 16 shows that VCSE consistently accelerates the training, which highlights the applicability of VCSE to diverse domains with different input types.

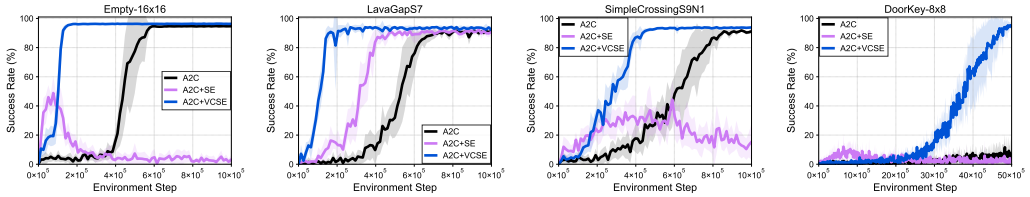


Figure 16: Learning curves on four navigation tasks from fully-observable MiniGrid (Chevalier-Boisvert et al., 2018) as measured on the success rate. The solid line and shaded regions represent the interquartile mean and standard deviation, respectively, across 16 runs.

C.2 DeepMind Control Suite Experiments

We further report experimental results in state-based DeepMind Control Suite experiments where we do not use state encoder for computing the intrinsic bonus. To make the scale of state norms be consistent across diverse tasks with different state dimensions, we divide the state input with its state dimension before computing the intrinsic bonus. As our underlying RL algorithm, we used Soft Actor-Critic (SAC; Haarnoja et al. 2018). For SE and VCSE, we disabled automatic tuning hyperparameter α and used lower value of $\alpha = 0.001$ in SAC, because we find that such an automatic tuning conflicts with introducing the intrinsic reward, similar to noise scheduling in DrQv2. Figure 17 shows that VCSE consistently accelerates the training, which highlights the applicability of VCSE to diverse domains with different input types.

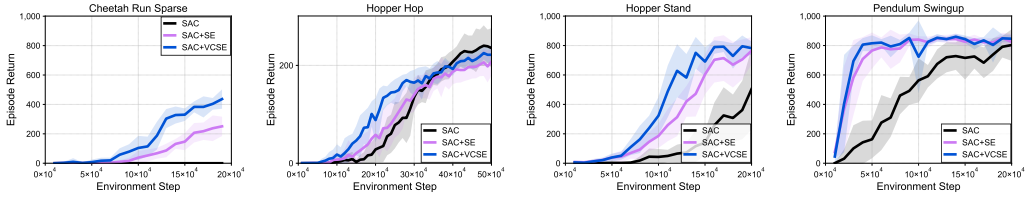


Figure 17: Learning curves on four locomotion tasks from state-based DeepMind Control Suite (Tassa et al., 2020) as measured on the success rate. The solid line and shaded regions represent the interquartile mean and standard deviation, respectively, across 16 runs.

D Additional Illustrations

We provide the additional illustration that helps understanding the procedure of estimating the conditional entropy with KSG estimator, which is explained in Section 3.2.

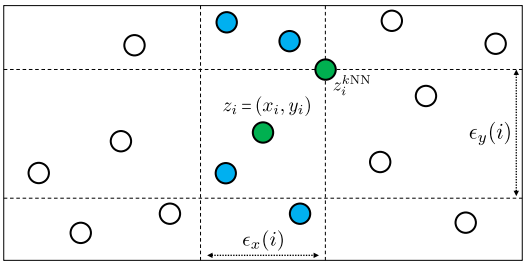


Figure 18: Illustration of a procedure for computing $\epsilon_x(i)$ and $n_x(i)$ when using the KSG estimator with $k = 2$. Given a centered point z_i, we first find k-nearest neighbor state $z_i^{k\text{NN}}$. Then $\epsilon_x(i)$ is twice the distance from x_i to $x_i^{k\text{NN}}$ and n_x can be computed as 5 by counting all the points located within $(x_i - \epsilon_x(i)/2, x_i + \epsilon_x(i)/2)$. We note that $\epsilon_y(i)$ and $n_y(i)$ can be also similarly computed.