SCIENCE CHINA
Information Sciences

• Supplementary File •

# Cumulative predictive information-regularized reinforcement learning for energy-efficient robot skill learning

Bang YOU[1], Di GUO[2], Fuchun SUN[3] & Huaping LIU[3*]

[1]*School of Information Engineering, Wuhan University of Technology, Wuhan 430070, China*
[2]*School of Artificial Intelligence, Beijing University of Posts and Telecommunications, Beijing 100876, China*
[3]*Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China*

## Appendix A Preliminaries

### Appendix A.1 Mutual Information

Mutual information [1] is a measure of the amount of information obtained about one random variable through knowing another random variable. Consider two random variables $x_1$ and $x_2$, the mutual information $I(x_1, x_2)$ is defined as the Kullback-Leibler (KL) divergence between their joint distribution and the product of their marginal distributions,

$$I(x_1, x_2) = \int_{\mathbf{x_1}, \mathbf{x_2}} p(x_1, x_2) \log \frac{p(x_1, x_2)}{p(x_1)p(x_2)} \mathrm{d}x_1 \mathrm{d}x_2. \tag{A1}$$

Despite the effectiveness in quantifying the statistical dependence between variables, computing mutual information based on its definition is typically intractable.

### Appendix A.2 Soft Actor-Critic

Soft actor-critic is an off-policy maximum entropy reinforcement learning algorithm, which optimizes a policy to maximize the expected cumulative rewards and the entropy of the policy. Unlike standard RL, the soft actor-critic agent receives an additional reward that is related to the entropy of the policy at every time step to encourage stochasticity, $H(\pi(\cdot|s_t) = -\int_{\mathcal{A}} \pi(\cdot|s_t) \log(\pi(\cdot|s_t)))\, \mathrm{d}a$. Soft actor-critic aims to learns an entropy-regularized stochastic policy $\pi_\omega$ parameterized by $\omega$, and two $Q$ functions $Q_{\sigma_1}$ and $Q_{\sigma_2}$ with parameters $\sigma_1$ and $\sigma_2$ respectively, to find a control policy. The soft $Q$ function can be learned by minimizing the Bellman error,

$$L_Q(\sigma_i) = \mathbb{E}_{(s,a,r,s',d)\sim D}\left[\left(Q_{\sigma_i}(s,a) - \mathbb{T}(r,s',d)\right)\right], \tag{A2}$$

where state $s$, action $a$, reward $r$, next state $s'$ and a termination flag $d$ are sampled from the replay buffer $D$. The target value $\mathbb{T}(r, s', d)$ is computed from the $Q$ function

$$\mathbb{T}(r,s',d) = r + \gamma(1-d)\Big(\min_{i=1,2} Q_{\sigma_{\mathrm{targ,i}}}(s',a') - \alpha \log \pi_\omega(a'|s')\Big), \tag{A3}$$

where $a' \sim \pi_\omega(\cdot|s')$, and $Q_{\sigma_{\mathrm{targ,i}}}$ denotes the target $Q$ function, which uses an exponential moving average of the parameters $\sigma_i$ for $i = 1, 2$. This particular parameter update has been shown to stabilize training.

The policy can be optimized by minimizing

$$L_\pi(\omega) = \mathbb{E}_{s,a\sim D,\pi}[\alpha \log \pi_\omega(a|s) - Q(s,a)], \tag{A4}$$

where the states $s$ and actions $a$ are sampled from the replay buffer and the stochastic policy, respectively. $Q(s,a) = \min_{i=1,2} Q_{\sigma_i}$ is the minimum of both $Q$ function approximators.

## Appendix B Cumulative Predictive Information-Regularized Reinforcement Learning

In this section, we present the cumulative predictive information-regularized reinforcement learning problem. We will also introduce the lower bounds of the prediction information for tractable estimation, and derive the information-regularized reward function.

* Corresponding author (email: hpliu@tsinghua.edu.cn)

## Appendix B.1 Problem Statement and Motivation

We aim to build RL agents that use energy-efficient policies to handle control tasks. We formulate the control task as an infinite-horizon MDP, which consists of the state space $s \in \mathcal{S}$, the action space $a \in \mathcal{A}$, the stochastic transition model $p(s_{t+1}|s_t, a_t)$, the reward function $r(s_t, a_t)$. At each time step, the agent receives the current state $s_t$ and determines its action $a_t$ based on its probabilistic policy $\pi(a_t|s_t)$, and then receives the reward signal $r(s_t, a_t)$ as a consequence of its choice. Considering that not all information in states is useful for choosing the optimal action, without loss of generality, we assume that the policy determines the action based on the latent state representation $z_t$ sampled from an encoder $z_t \sim f_\theta(z_t|s_t)$ with parameters $\theta$. The original objective of the RL agent is to maximize the expected cumulative rewards $\mathbb{E}_\tau\left[\sum_{t=1}^{\infty} r(s_t, a_t)\right]$ where $\tau = (s_1, a_1, s_2, a_2, \dots)$ is the agent's trajectory with an infinite horizon.

Predictable behaviors usually contain periodic patterns and avoid unnecessary variations in movements, therefore decreasing the energy consumption required to achieve a control task. We expect to bias RL agents to produce predictable behaviors. To this end, we optimize the policy to maximize the expected cumulative predictive information over the trajectories of the agent, $\mathbb{E}_\pi\big[\sum_{t=1}^{\infty} I(z_{t+1}; z_t, a_t)\big]$, and the rewards:

$$\max_\theta \quad \mathbb{E}_{\pi,f}\left[\sum_{t=1}^{\infty} r(s_t, a_t)\right] \quad \text{s.t.} \quad \mathbb{E}_{\pi,f}\left[\sum_{t=1}^{\infty} I(z_{t+1}; z_t, a_t)\right] \geqslant I_p, \tag{B1}$$

where the agent is encouraged to retain $I_p > 0$ bits of predictive information cumulatively per episode. The per-step predictive information $I(z_{t+1}; z_t, a_t)$ measures the correlation between the next state representation, the current state representation paired with the current action, which has the definition

$$I(z_{t+1}; z_t, a_t) = \mathbb{E}_{p(z_t, a_t, z_{t+1})}\left[\log \frac{p(z_t, a_t, z_{t+1})}{p(z_t, a_t)p(z_{t+1})}\right], \tag{B2}$$

where the expectation is computed over the joint distribution $p(z_t, a_t, z_{t+1})$. By introducing a Lagrange multiplier $\alpha$, we can convert the optimization problem with a single inequality constraint into an unconstrained one:

$$\max_\theta \mathbb{E}_{\pi,f}\left[\sum_{t=1}^{\infty} r(s_t, a_t) + \alpha\big(I(z_{t+1}; z_t, a_t) - I_p\big)\right], \tag{B3}$$

where $\alpha$ trade-offs the objective of maximizing rewards and the objective of preserving the predictive information. Solving the optimization problem in Eq. B1 is equivalent to maximizing the objective function in Eq. B3.

Intuitively, by additionally maximizing predictive information, the agent is encouraged to learn representations that are easily predictable. A policy that relies on these predictable representations for decision-making is likely to produce temporally consistent behaviors. Furthermore, to obtain representations that are easily predictable, the agent tends to select predictable behaviors, which in turn result in states that are more predictable. The resulting predictable action sequence will improve energy efficiency.

## Appendix B.2 Lower-bounding Predictive Information

Unfortunately, directly computing the per-step mutual information $I(z_{t+1}; z_t, a_t)$ based on its definition is intractable. To effectively estimate the mutual information, we firstly construct a lower bound of the mutual information based on variational inference. Specifically, we introduce a parametric variational distribution $q_\theta(z_{t+1}|z_t, a_t)$ with learnable parameters $\theta$ into the definition of the mutual information:

$$\begin{aligned} I(z_{t+1}; z_t, a_t) &= \mathbb{E}_{p(z_t, a_t, z_{t+1})}\left[\log \frac{p(z_{t+1}|z_t, a_t)}{p(z_{t+1})}\right] \\ &= \mathbb{E}_{p(z_t, a_t, z_{t+1})}\left[\log \frac{q_\theta(z_{t+1}|z_t, a_t)}{p(z_{t+1})}\right] + \mathbb{D}_{\mathrm{KL}}\Big(p(z_{t+1}|z_t, a_t) \parallel q_\theta(z_{t+1}|z_t, a_t)\Big) \\ &\geqslant -\mathbb{E}_{p(z_t, a_t, z_{t+1})}\left[\log \frac{p(z_{t+1})}{q_\theta(z_{t+1}|z_t, a_t)}\right], \end{aligned} \tag{B4}$$

where the inequality is introduced by the non-negativity of the KL divergence. The variational distribution $q_\theta(z_{t+1}|z_t, a_t)$ can be regarded as the dynamic model which predicts the future representation based on the current representation and the action. As the marginal distribution of $z_{t+1}$ is unknown, we use the encoder $f_\theta$ to generate the representation and obtain the lower bound on the mutual information:

$$\begin{aligned} I(z_{t+1}; z_t, a_t) &\geqslant -\mathbb{E}_{p(z_t, a_t, z_{t+1})}\left[\log \frac{f_\theta(z_{t+1}|s_{t+1})}{q_\theta(z_{t+1}|z_t, a_t)}\right] + \mathbb{D}_{\mathrm{KL}}\Big(f_\theta(z_{t+1}|s_{t+1}) \parallel p(z_{t+1})\Big) \\ &\geqslant -\mathbb{E}_{p(z_t, a_t, s_{t+1})}\Big[\mathbb{D}_{\mathrm{KL}}\Big(f_\theta(z_{t+1}|s_{t+1}) \parallel q_\theta(z_{t+1}|z_t, a_t)\Big)\Big] = I_{\mathrm{KL}}, \end{aligned} \tag{B5}$$

where the inequality is introduced by the non-negativity of the KL divergence. The obtained lower bound is the expectation over the KL divergence between the encoder and the dynamic model. Minimizing the lower bound will allow the agent to learn the dynamic model that predicts the future accurately.

However, using the KL-based lower bound alone may cause collapsed solutions, since the generative model $q_\theta(z_{t+1}|z_t, a_t)$ will always output zero for a constant encoding. To solve this problem, we construct an InfoNCE lower bound of the mutual information [2], $I_{\mathrm{NCE}}$, which is based on a discriminative model. On the other hand, using only the InfoNCE bound may fail to capture complex dynamics. Hence, we use the InfoNCE lower bound and the KL-based lower bound both to estimate the predictive information. Specifically, the InfoNCE lower bound is given by

$$I(z_{t+1}; z_t, a_t) \geqslant I_{\mathrm{NCE}} + \log B = \mathop{\mathbb{E}}_{p,N}\left[\log \frac{h_\theta(z_t, a_t, z_{t+1})}{h_\theta(z_t, a_t, z_{t+1}) + U}\right] + \log B, \tag{B6}$$

where $B$ denotes the number of samples containing one positive sample $z_{t+1}$ from the joint distribution $p(z_t, a_t, z_{t+1})$ and $B-1$ negative samples $z^*_{t+1}$ from the marginal distribution $p(z_{t+1})$. $N$ denotes a set of negative samples, and the score function $h_\theta(z_t, a_t, z_{t+1})$ is an exponentiated bilinear similarity function with learnable parameters $\theta$, and $U$ is given by

$$U = \sum_{z^*_{t+1} \in N} h_\theta(z_t, a_t, z^*_{t+1}), \tag{B7}$$

where negative samples $z^*_{t+1}$ sampled from $N$. The score function is optimized to predict the next state embedding $z_{t+1}$ accurately based on the current state embedding $z_t$ and the action $a_t$, as maximizing the InfoNCE lower bound. The term of $\log B$ is a constant independent of learnable parameters $\theta$, and we next omit it for simplicity.

We then combine the InfoNCE lower bound in Eq. B6 with the KL bound in Eq. B5, to estimate the cumulative predictive information:

$$\mathbb{E}_\pi\left[\sum_{t=1}^{\infty} I(z_{t+1}; z_t, a_t)\right] \geqslant \mathop{\mathbb{E}}_{\pi,f,p,N}\left[\sum_{t=1}^{\infty} \log \frac{h_\theta(z_t, a_t, z_{t+1})}{h_\theta(z_t, a_t, z_{t+1}) + U} - \beta \log \frac{f_\theta(z_{t+1}|s_{t+1})}{q_\theta(z_{t+1}|z_t, a_t)}\right], \tag{B8}$$

where the expectation is computed over the policy $\pi(\cdot|\cdot)$, the encoder $f_\theta(\cdot|\cdot)$ and the transition $p(\cdot|\cdot,\cdot)$ and $N$, and the hyperparameter $\beta$ is introduced to balance two bounds.

## Appendix B.3 Information-regularized Reward Function

By plugging Eq. B8 into Eq. B3, we can obtain the optimization objective:

$$\max_\theta \mathbb{E}_{\pi,f,p,N}\left[\sum_{t=1}^{\infty} r(s_t, a_t) - \alpha\beta \log \frac{f_\theta(z_{t+1}|s_{t+1})}{q_\theta(z_{t+1}|z_t, a_t)} + \alpha \log \frac{h_\theta(z_t, a_t, z_{t+1})}{h_\theta(z_t, a_t, z_{t+1}) + U}\right]. \tag{B9}$$

The constant $I_p$ in Eq. B3 is omitted, since it is irrelevant to optimizing parameters $\theta$.

Since the episode may terminate with probability $1-\gamma$ at each time step, there are no bits of information or rewards to preserve after the episode terminates. Hence, we introduce the discount factor $\gamma \in (0, 1)$ into Eq. B9. We then obtain the final objective that jointly optimizes the policy $\pi_\theta(a_t|z_t)$, the encoder $f_\theta(z_t|s_t)$, the dynamic model $q_\theta(z_{t+1}|z_t, a_t)$ and the score function $h_\theta(z_t, a_t, z_{t+1})$ to maximize the rewards and the predictive information:

$$\max_\theta \mathbb{E}_{\pi,f,p,N}\left[\sum_{t=1}^{\infty} \gamma^t \left[r(s_t, a_t) - \alpha\beta \log \frac{f_\theta(z_{t+1}|s_{t+1})}{q_\theta(z_{t+1}|z_t, a_t)} + \alpha \log \frac{h_\theta(z_t, a_t, z_{t+1})}{h_\theta(z_t, a_t, z_{t+1}) + U}\right]\right]. \tag{B10}$$

The discount factor $\gamma$ makes the sum over the infinite horizon finite, which provides the theoretical convergence of the final objective.

Based on the final objective, we can obtain an information-regularized reward function, $r^*(s_t, a_t, s_{t+1})$, which augments the environment rewards with the information gain term:

$$r^*(s_t, a_t, s_{t+1}) = r(s_t, a_t) - \alpha\beta \log \frac{f_\theta(z_{t+1}|s_{t+1})}{q_\theta(z_{t+1}|z_t, a_t)} + \alpha \log \frac{h_\theta(z_t, a_t, z_{t+1})}{h_\theta(z_t, a_t, z_{t+1}) + U}, \tag{B11}$$

where the KL-based lower bound and the InfoNCE lower bound allow the agent to learn predictable representations.

Optimizing the final objective, the agent will receive rewards for selecting behaviors that result in transitions that the agent can predict well. The dynamic model and the score function in the InfoNCE lower bound are optimized to predict the next representation accurately. Notably, the agent improves the consistency of representations by optimizing not only its latent representations but also its behavior: the agent learns predictable representations by the generative dynamic model and the discriminative InfoNCE lower bound, and is incentivized to visit states where its representations are easily predicted by its learned models. The consistent representations will indirectly improve the consistency of actions.

## Appendix C A Practical Algorithm

The objective of our CPRL agent in Eq. B10 can be directly optimized using existing reinforcement learning algorithms. In practice, we implement CPRL on top of the soft actor-critic algorithm (SAC). We first define a parameterized Q function to estimate the expected information-regularized reward at the current state-action pair:

$$Q_\upsilon(s_t, a_t) = \mathbb{E}_{p,f,\pi}\left[\sum_{t=0}^{\infty} \gamma^t r^*(s_t, a_t, s_{t+1}) - \epsilon \sum_{t=1}^{\infty} \gamma^t \pi_\theta(a_t|z_t)\right], \tag{C1}$$

where $\upsilon$ is the learnable parameter. The coefficient $\epsilon$ weights the entropy regularizer of SAC.

The Q function can be optimized by using a standard recursive Bellman equation without any changes:

$$\mathcal{L}(\upsilon) = \mathbb{E}_{p,f,\pi}\left[\left(Q_\upsilon(s_t, a_t) - y(s_t, a_t)\right)^2\right], \tag{C2}$$

where the target is given by

$$y(s_t, a_t) = r^*(s_t, a_t, s_{t+1}) + \gamma(1-d)\Big[Q_v(s_{t+1}, a_{t+1}) - \epsilon\pi_\theta(a_{t+1}|z_{t+1})\Big], \tag{C3}$$

where $\gamma$ is the discounted factor and $d$ is the termination flag. We employ the independent target Q function to compute the target and stop the gradient through it.

We then introduce a value function to estimate the expected information-regularized reward at the current state:

$$\begin{aligned} V^\pi(s_t) &= \mathbb{E}_{p,f,\pi}\Big[Q(s_t, a_t) - \epsilon\pi_\theta(a_t|z_t)\Big] \\ &= \mathbb{E}_{p,f,\pi}\Big[r^*(s_t, a_t, s_{t+1}) - \epsilon\pi_\theta(a_t|z_t) + \gamma(1-d)\Big[Q_v(s_{t+1}, a_{t+1}) - \epsilon\pi_\theta(a_{t+1}|z_{t+1})\Big]\Big]. \end{aligned} \tag{C4}$$

The policy $\pi_\theta(a_t|z_t)$, the encoder $f_\theta(z_t|s_t)$, the dynamic model $q_\theta(z_{t+1}|z_t, a_t)$ and the score function $h_\theta(z_t, a_t, z_{t+1})$ are simultaneously optimized by minimizing the negative expected information-regularized reward:

$$\begin{aligned} \mathcal{L}(\theta) = -\mathbb{E}_{p,f,\pi}\Big[&r(s_t, a_t) - \epsilon\pi_\theta(a_t|z_t) - \alpha\beta\log\frac{f_\theta(z_{t+1}|s_{t+1})}{q_\theta(z_{t+1}|z_t, a_t)} \\ &+ \alpha\log\frac{h_\theta(z_t, a_t, z_{t+1})}{h_\theta(z_t, a_t, z_{t+1}) + U} + \gamma(1-d)\Big[Q_v(s_{t+1}, a_{t+1}) - \epsilon\pi_\theta(a_{t+1}|z_{t+1})\Big]\Big]. \end{aligned} \tag{C5}$$

We optimize this expectation via sampling during training. We apply the reparametrization trick to allow gradients through the outputs of the encoder. In practical implementation, we use expressive neural networks to instantiate the Q function, the policy, the encoder, the dynamic model, and the score function. We use two separate target Q functions and take a minimum of two Q values. To stabilize training, the parameters of target Q functions are updated by an exponentially moving average of the parameters of the corresponding Q function:

$$\upsilon_t = \tau\upsilon + (1-\tau)\upsilon_t, \tag{C6}$$

where $\upsilon_t$ are the parameters of target Q function $Q_{\upsilon_t}(s_t, a_t)$, and $\tau \in [0, 1)$ is the coefficient of the exponential moving average.

In addition to the above actor-critic updates, it is required to determine the introduced Lagrange multiplier $\alpha$ and $\beta$ in Eq. B10. Instead of setting the Lagrange multiplier manually, we can automate this process by constructing the dual objective. Specifically, we use the InfoNCE bound value in a single transition to approximate the expectation of the sum of the InfoNCE lower bound. The Lagrange multiplier $\beta$ can be optimized by minimizing the dual objective

$$L(\alpha) = \alpha\Big(\log\frac{h_\theta(z_t, a_t, z_{t+1})}{h_\theta(z_t, a_t, z_{t+1}) + U} - I_m\Big), \tag{C7}$$

where $I_m$ is the constraint. The parameters of the score function are fixed during optimizing the Lagrange multiplier $\alpha$. Similarly, the Lagrange multiplier $\beta$ can be optimized by minimizing the dual objective

$$L(\beta) = \beta\Big(-\log\frac{f_\theta(z_{t+1}|s_{t+1})}{q_\theta(z_{t+1}|z_t, a_t)} - I_n\Big), \tag{C8}$$

where $I_n$ is the constraint.

The final algorithm is presented in Algorithm C1. The algorithm proceeds by alternating between updating the parameters of our model by using the stochastic gradients from batches sampled from a replay buffer and collecting new experiences from the environment. We apply the Adam optimizer to perform gradient updates.

## Appendix D Experimental Evaluation

In this section, we perform extensive experiments to evaluate our method. We will describe the experimental setup and present the experimental results of our method and leading baselines. We will also conduct several ablation studies and visualize the learned policies.

### Appendix D.1 Experimental Setup

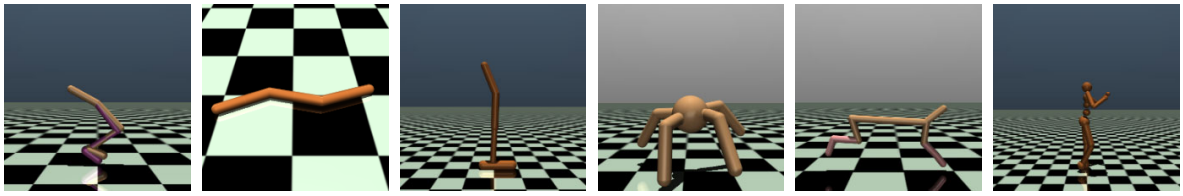
We evaluate our method and the state-of-the-art methods on a range of challenging continuous control tasks from the common locomotion benchmark suite [3], which are implemented using MuJoCo. Specifically, six locomotion tasks are considered (see Fig. D1): Walker2d-v2, Swimmer-v2, Hopper-v2, Ant-v2, HalfCheetah-v2, and Humanoid-v2. In all these tasks, the goal of the agent is to control a robot with complex dynamics and ground contacts to move forward as quickly as possible. The "v2" refers to the version of the used tasks. More detailed task descriptions are available in [3].

We compare our method to the state-of-the-art method, LZSAC [4], which compresses action sequences to improve the predictability of actions, and RPC [5], which compresses a sequence of states to learn predictable policies, and VIB [6] which learns simple policies by compressing individual states. All algorithms use the same implementations of SAC provided by RPC with the same hyperparameters to achieve a fair comparison. We obtain the results for RPC by performing its official implementation, while obtaining the results for VIB and SAC by performing the implementation provided by [6]. While the original implementation of LZSAC uses a different

**Algorithm C1** Cumulative Predictive Information-Regularized Reinforcement Learning for Predictable Control.

**Require:** policy $\pi_\theta(a_t|z_t)$, encoder $f_\theta(z_t|s_t)$, dynamic model $q_\theta(z_{t+1}|z_t, a_t)$, score function $h_\theta(z_t, a_t, z_{t+1})$, Q function $Q_\upsilon(s_t, a_t)$, target Q function $Q_{\upsilon_t}(s_t, a_t)$, replay buffer $\mathcal{D}$, Lagrange multipliers $\alpha$ and $\beta$, batch size $B$, learning rate $\rho$;

**for** each training step **do**
  collect experience $(s_t, a_t, r_t, s_{t+1})$ and add it to replay buffer;
  **for** each gradient step **do**
    Sample a minibatch of transitions $(s_t^i, a_t^i, r_t^i, s_{t+1}^i)_{i=1}^B \sim \mathcal{D}$ from replay buffer;
    Compute information-regularized reward by following Eq. B11;
    Update Q function by following Eq. C2:
$$\upsilon \leftarrow \upsilon - \rho\hat{\nabla}_\upsilon \mathcal{L}(\upsilon);$$
    Update policy, encoder, dynamical model, and score function by following Eq. C5:
$$\theta \leftarrow \theta - \rho\hat{\nabla}_\theta \mathcal{L}(\theta);$$
    Update Lagrange multipliers $\alpha$ and $\beta$ by following Eq. C7 and Eq. C8:
$$\alpha \leftarrow \alpha - \rho\hat{\nabla}_\alpha \mathcal{L}(\alpha)$$
$$\beta \leftarrow \beta - \rho\hat{\nabla}_\beta \mathcal{L}(\beta);$$
    Update target Q functions by following Eq. C6:
$$\upsilon_t = \tau\upsilon + (1-\tau)\upsilon_t;$$
  **end for**
**end for**



(a) Walker2d-v2 (b) Swimmer-v2 (c) Hopper-v2 (d) Ant-v2 (e) HalfCheetah-v2 (d) Humanoid-v2

**Figure D1** Continuous control tasks used in our experiments, namely Walker2d-v2, Swimmer-v2, Hopper-v2, Ant-v2, HalfCheetah-v2, and Humanoid-v2.

**Table D1** Hyperparameters used in CPRL

| Parameter | Value |
|---|---|
| Replay buffer capacity | 1000 000 |
| Optimizer | Adam |
| Critic Learning rate | $3 \times 10^{-4}$ |
| Critic Q-function EMA | 0.005 |
| Critic target update freq | 1 |
| Actor learning rate | $3 \times 10^{-4}$ |
| Actor update frequency | 1 |
| Temperature learning rate | $3 \times 10^{-4}$ |
| Initial steps | 10000 |
| Discount factor | 0.99 |
| Batch Size | 256 |
| Log stddev bounds | [0.1 10.0] |
| Mean bounds | [-30 30] |
| Learning rate for multiplier $\alpha$ | $3 \times 10^{-4}$ |
| Learning rate for multiplier $\beta$ | $3 \times 10^{-4}$ |
| Reward scale factor | 0.1 |
| Constraint $I_m$ | -0.14 |
| Constraint $I_n$ | 5.0 |

SAC implementation from RPC, we implement LZSAC on top of the official implementation of RPC to ensure a fair comparison. The comparative experiments are conducted on GeForce GTX 2080 Ti and Intel(R) Xeon(R) E5-2620 CPU. The software environment is configured with Python 3.6, Tensorflow-gpu 2.6.0, Gym 0.13.1, MuJoCo-Py 2.0.2.10 and Tf-agents 0.7.1.

We implement CPRL on top of the official implementation of RPC and use all default hyperparameters from this implementation. The policy $\pi_\theta(a_t|z_t)$ and the Q function are parameterized by using a 2-layer fully-connected network with ReLU activation functions. Each hidden layer has 256 units for the policy and the Q-function. Following RPC, we parameterize the encoder $f_\theta(z_t|s_t)$ and the dynamic model $q_\theta(z_{t+1}|z_t, a_t)$ using forward neural networks. The encoder consists of three fully-connected layers with ReLU activation functions. Each hidden dimension is set to 256 and the output dimension is set to 20. The dynamic model is parameterized by a 3-layer fully-connected network with ReLU activation functions and 20-dimensional output, where each hidden layer has 256 units. The output is divided into the mean and the standard deviation of a diagonal Gaussian distribution for both the encoder and the dynamic model. To improve training stability, we squash the mean to be within $[-30, 30]$ and squash the standard deviation to be within $[0.1, 10.0]$ for the

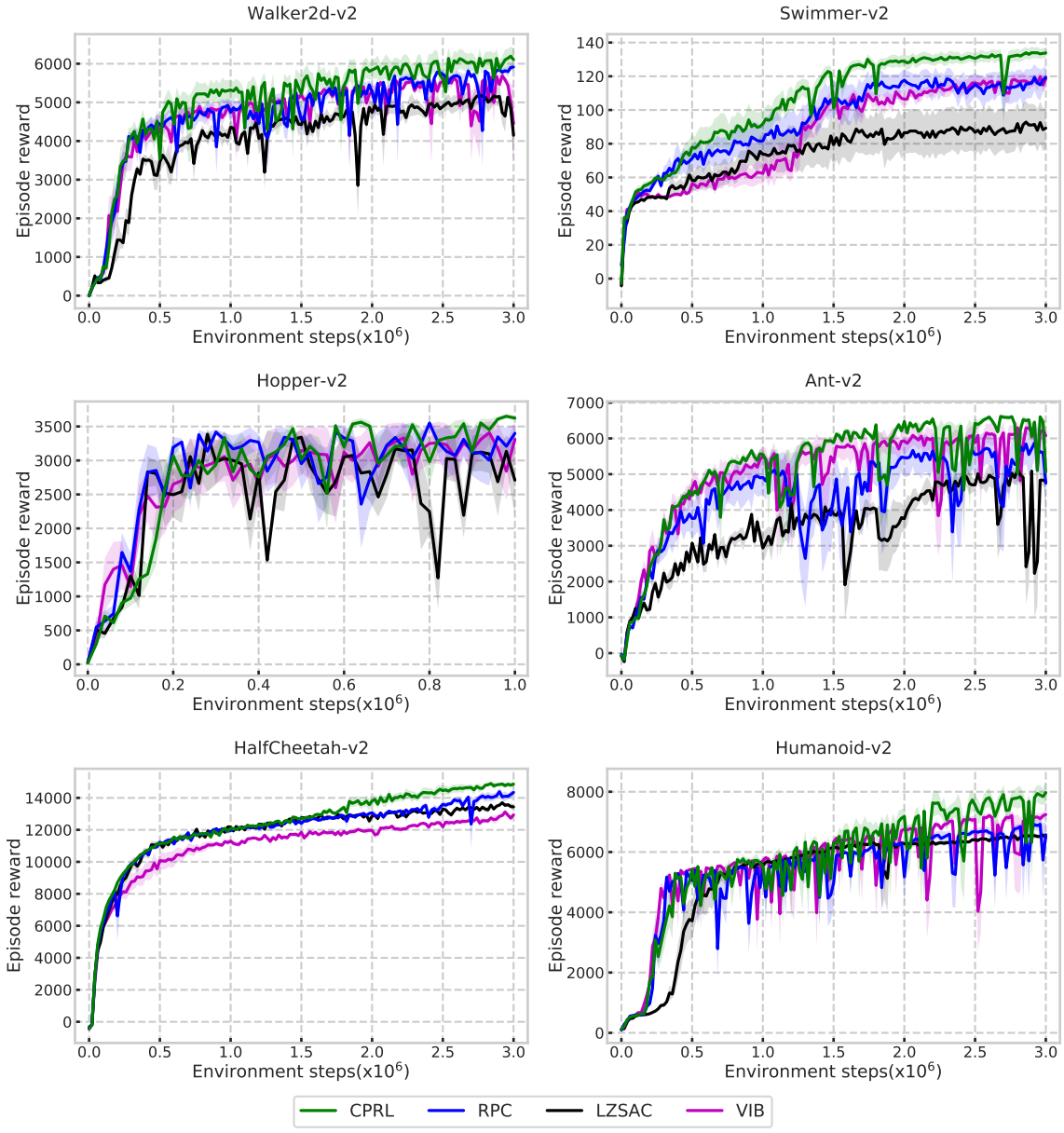



**Figure D2** Learning curves of our method and baselines on six continuous control tasks. Overall, our method achieves better performance than baselines. Lines are the mean episodic reward across five different runs. Shaded regions represent the standard error of the mean. For each run, we evaluate the performance by computing an average reward over 30 test episodes.

encoder and the dynamic model. We employ the score function $h_\theta(z_t, a_t, z_{t+1})$ from [7] without any modifications.

We set the batch size to 256 by following RPC. We determine the hyperparameter $I_m$ and the hyperparameter $I_n$ by performing hyperparameter tuning on the Walker2d-v2 task and fix them across all tasks. Then the hyperparameter $I_p$ can be determined after selecting $I_m$ and $I_n$. Following RPC, we initialize the values of the Lagrange multipliers $\alpha$ and $\beta$ to $10^{-6}$ and update them using stochastic gradient ascent with learning rate $3e^{-4}$. We show an overview of the CPRL hyperparameters in Table. D1.

For each algorithm, we perform one evaluation rollout every 10000 environment steps. Each evaluation is performed by computing the average rewards over 30 episodes to obtain a reliable comparison. Each agent performs one gradient update per environment step to ensure a fair comparison. We train five different instances of each algorithm with different random seeds.

## Appendix D.2 Performance

Fig. D2 compares the performance of our algorithm against the state-of-the-art approaches on six locomotion tasks. The plot shows the mean and the standard error of five independent seeds. CPRL achieves faster learning speed and better performance than the baselines on the majority of tasks. Specifically, On Walker2d-v2, Swimmer2d-v2, HalfCheetah-v2, and Humanoid-v2 tasks, CPRL is significantly better than all baselines by a large margin. On the Ant-v2 task, our algorithm is comparable to VIB, but better than LZSAC

**Table D2** Averaged score (mean and standard error over 5 seeds) achieved by our method and baselines in the last 300K training steps. Our method achieves higher average rewards than all baselines. The bold font indicates that the upper-bound on the reward (based on the given standard error intervals) for the given method, is larger than or equal to the respective lower bound for every other method.

| Scores | Walker2d-v2 | Swimmer-v2 | Hopper-v2 | Ant-v2 | HalfCheetah-v2 | Humanoid-v2 |
|---|---|---|---|---|---|---|
| CPRL (Ours) | **5964 ± 266** | **132 ± 3** | **3382 ± 65** | **6164 ± 136** | **14775 ± 137** | **7486 ± 162** |
| LZSAC | 4930 ± 207 | 89 ± 12 | 2740 ± 303 | 4280 ± 167 | 13424 ± 54 | 6497 ± 72 |
| RPC | 5512 ± 48 | 115 ± 9 | **3127 ± 244** | 5431 ± 497 | 14046 ± 235 | 6495 ± 320 |
| VIB | 5278 ± 85 | 116 ± 3 | **3193 ± 265** | **5953 ± 219** | 12739 ± 141 | 6936 ± 352 |

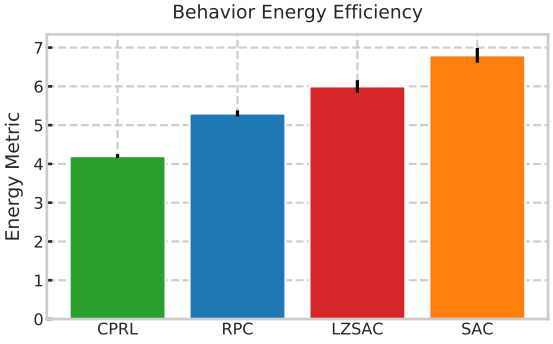



**Figure D3** Comparison of energy efficiency. We collect 20 evaluation trajectories and compute the energy metric on the collected trajectories. The plot shows the mean and standard error over 20 trajectories.

and RPC in terms of performance and sample efficiency. CPRL is competitive to the baselines on the Hopper-v2 task. Maximizing the predictive information improves the consistency of state representations, leading to consistent behaviors. Hence we attribute the performance improvement achieved by CPRL than baselines to the improved predictability of behaviors generated by our policies. In Section Appendix D.4 we will investigate the predictability of behaviors.

We also compare the final performance of all methods in Table D2. Following RPC, to reduce the variations in scores, we compute the final performance by averaging the scores over the last 300K training steps. The results in Table D2 show that our algorithm achieves higher final performance than previous approaches on all tasks. On the Swimmer2d-v2 task, for instance, CPRL achieves a return of 132, higher than a return of 89 achieved by LZSAC, a return of 115 achieved by RPC, and a return of 116 achieved by VIB. CPRL achieves a return of 14775 on the HalfCheetah-v2 task, while RPC achieves a return of 14046 and VIB achieves a return of 12739.

## Appendix D.3 Energy Efficiency

We further evaluate the energy efficiency of our approach and baselines. In our experiment, we use the adapted Cost of Transport (CoT) as the energy metric, which is commonly used in previous practices. The energy metric used in our experiment can be formulated as

$$CoT = \frac{\sum_{t=1}^{T}\sum_{i=1}^{N}\tau_i^t\omega_i^t\Delta t}{d}, \tag{D1}$$

where $\tau_i$ is the torque applied to the $i$th joint of the robot, $\omega_i$ is the angular velocity of the $i$th joint, $\Delta t$ is the specific amount of simulated time per single timestep, $T$ is the horizon of an episode, and $d$ is the forward locomotion distance. We collect 20 evaluation trajectories using learned policies and compute the energy metric for our algorithm and baselines. The experiment results in Fig. D3 demonstrate that our algorithm achieves the highest energy efficiency among all algorithms.

## Appendix D.4 Predictability of Policies

Our main hypothesis is that preserving predictive information helps induce predictable and consistent behaviors, therefore improving energy efficiency. In this Section, we verify this hypothesis by evaluating the predictability of action sequences. Arguably, the predictability of a behavior can be straightforwardly judged by visualizing it. Therefore, we visualize the action series produced by CPRL. Fig. D4 shows the action sequences produced by the CPRL, RPC, and SAC on the Humanoid task. CPRL generates the most consistent and simplest action trajectory, featuring a high degree of periodicity. In contrast, the trajectories produced by RPC and SAC exhibit numerous erratic changes and fluctuations. Predictable action sequences avoid unnecessary movements, which reduces the energy consumption required to execute a control task. Hence, we conclude that the learned policy by our CPRL agent generates simpler and more predictable action sequences and achieves better energy efficiency than baselines.

Inspired by [4], we further use lossless compression algorithms to *quantify* the predictability of action sequences. Trajectories can be effectively compressed if they contain repetitive or periodic structures. Specifically, for all six locomotion tasks, we collect state-action trajectories using learned policies, round them to one decimal place, save them as .npy file, and then compress them using bzip2, a

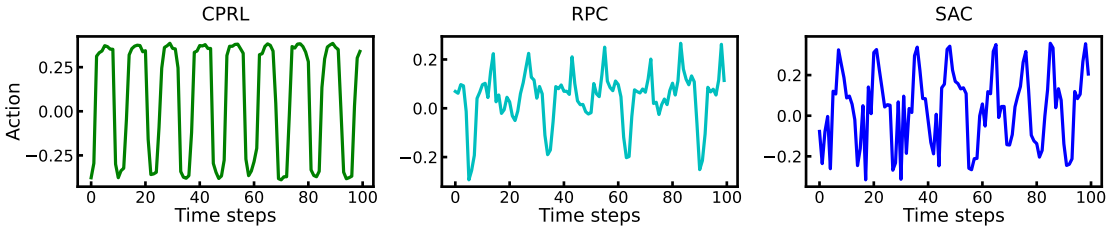



**Figure D4** Visualizations of action sequences generated by CPRL, RPC and SAC on the Humanoid-v2 task. Action sequences produced by the CPRL agent show a simpler periodic pattern than baselines. Actuators were chosen to show qualitatively different behaviors.

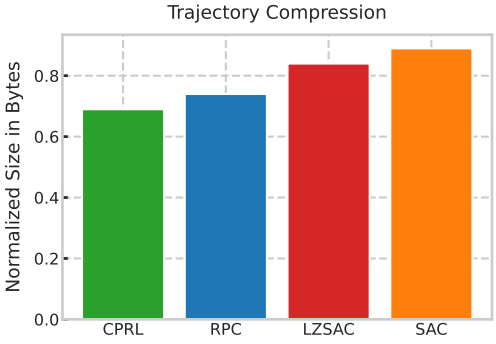



**Figure D5** We compress trajectories produced by CPRL and baselines on all six locomotion tasks by a lossless compression algorithm, bzip2. The plot shows normalized averaged file size in bytes over 5 runs and 6 tasks. Each run includes 30 collected trajectories. CPRL achieves the smallest average size among all methods.

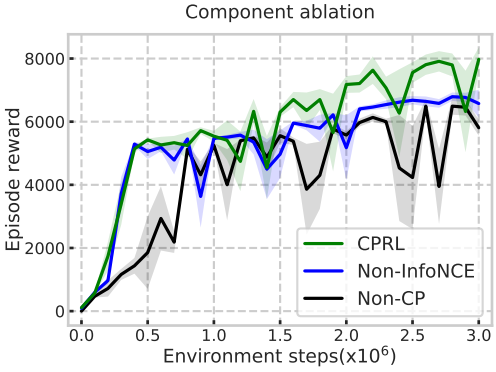



**Figure D6** Performance comparison to two ablations of our CPRL model. CPRL achieves significantly higher rewards than the Non-InfoNCE ablation and the Non-CP ablation.

common lossless compression algorithm. We use the standard bzip2 algorithm, which is available by directly installing the public bz2 Python package. For each trajectory, we call the bz2.compress() method provided by the bz2 package to perform trajectory compression. We show the normalized average file sizes in bytes over 5 runs and 6 tasks for all methods in Figure D5. The normalized file sizes are achieved by scaling the file sizes by the largest size among all methods for each task. Experimental results in Figure D5 show that trajectories of actions and states generated by CPRL are more easily compressed than previous methods. This indicates that our learned policies generate trajectories that show more cyclical and repetitive patterns, therefore improving energy efficiency.

## Appendix D.5 Ablation Studies

We perform ablation studies to investigate the effect of each component of our CPRL model. Specifically, we consider two ablations of our algorithm: Non-InfoNCE, which removes the InfoNCE lower bound on predictive information in Eq. B11, and Non-CP, which removes the InfoNCE lower bound and the KL-based lower bound both in Eq. B11. The Non-CP ablation actually is the SAC algorithm without preserving predictive information. Figure. D6 shows the performance of CPRL and its two ablations on the Humanoid-v2 task. The Figure shows the mean and the standard error of five independent seeds. For each run, we evaluate the performance by computing an average reward over 30 evaluation episodes. The results in Figure. D6 show that in CPRL achieves better performance than its

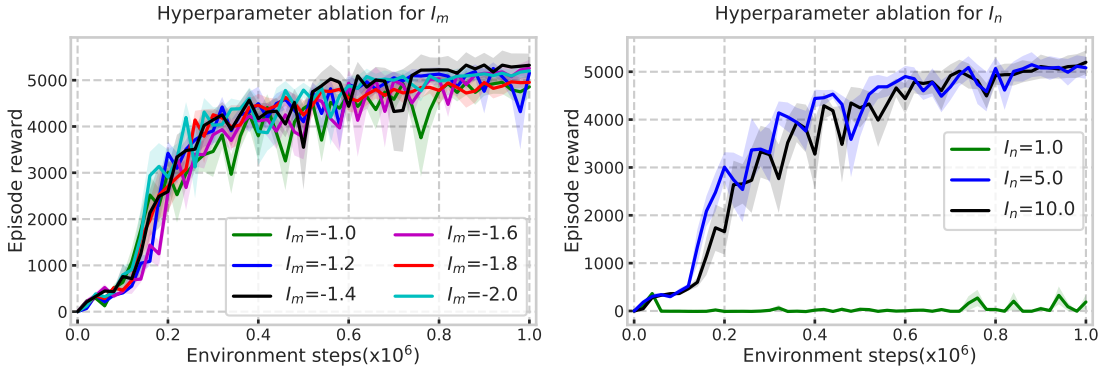



**Figure D7** An ablation on the choice of the specific hyperparameters. For each run, we evaluate the performance by computing an average reward over 30 test episodes. Our algorithm is robust to small variations in the information constraints $I_m$ and $I_n$.

ablations. The Non-CP ablation obtains lower scores than CPRL, indicating that preserving predictive information is helpful in achieving good performance. The performance improved by CPRL over Non-InfoNCE suggests that our InfoNCE lower bound on the mutual information helps learn good policies. Moreover, we observe that Non-InfoNCE outperforms Non-CP in terms of performance. This indicates that maximizing the KL-based lower bound improves performance.

We also conduct an ablation to understand how the choices of hyperparameters affect the performance of our algorithm. The hyperparameters of our algorithm are selected based on common practices [7], except for the constraints $I_m$ and $I_n$. The choice of the hyperparameter $I_m$ and $I_n$ determines the value of the hyperparameter $I_p$ that controls how many bits of predictive information should be preserved per episode (see Eq. B1). We thus focus on the choice of the constraint $I_m$ and $I_n$ in our ablation. We set the search space to $[-1.0, -1.2, -1.4, -1.6, -1.8, -2.0]$ for $I_m$, while setting the search space to $[-1.0, -5.0, -10.0]$ for $I_n$. The training step is set to $1 \times 10^6$ due to the computational cost. We implement the CPRL agent with each configuration across five different seeds to obtain reliable results. Fig. D7 shows the performance curves of our algorithm for different $I_m$ and $I_n$ on the Walker2d-v2 task. Our algorithm is robust to small changes in the hyperparameter $I_m$. When changing $I_n$ from -10.0 to -5.0, our algorithm also achieves consistent performance. However, the performance decreases to around 0 when setting $I_n$ to -1.0. We conjecture that the constraint of $I_n = -1.0$ is too tight and the agent is overly biased to preserve the predictive information, leading to poor performance.

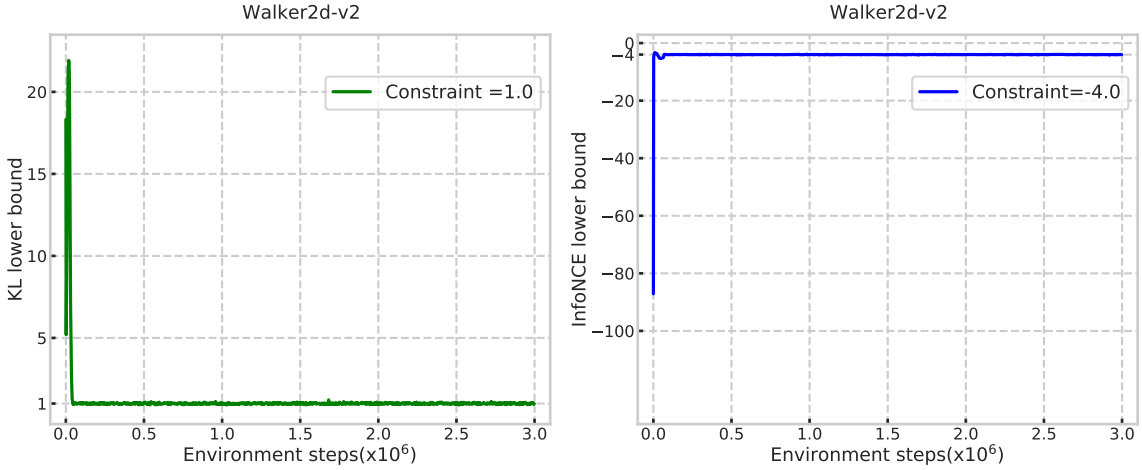



**Figure D8** Training curves of the KL lower bound and the InfoNCE lower bound on the predictive information. Two lower bounds on the predictive information both converge to the target constraint values (1.0 and -4.0).

## Appendix D.6 Training Curves of Lower Bounds

We conduct experiments to evaluate if the imposed constraints are satisfied during training. Specifically, we set the constraint $I_n$ to 1.0 for the KL lower bound, and set the constraint $I_m$ to -4.0 for the InfoNCE lower bound. Training curves in the Fig. D8 show that the lower bounds of the predictive information stably converge to the target constraint values. This result verifies that the imposed information constraints are strictly satisfied during training.

**References**

1 Shang Y, Yuan Z, Yan Y. Mim4dd: Mutual information maximization for dataset distillation. In: Proceedings of Advances in Neural Information Processing Systems, 2024

2 Oord A, Li Y, Vinyals O. Representation learning with contrastive predictive coding. 2018. ArXiv:1807.03748
3 Duan Y, Chen X, Houthooft R, et al. Benchmarking deep reinforcement learning for continuous control. In: Proceedings of International Conference on machine learning, 2016
4 Saanum T, Elteto N, Dayan P, et al. Reinforcement learning with simple sequence priors. In: Proceedings of Advances in Neural Information Processing Systems, 2024
5 Eysenbach B, Salakhutdinov R R, Levine S. Robust predictable control. In: Proceedings of Advances in Neural Information Processing Systems, 2021
6 Igl M, Ciosek K, Li Y, et al. Generalization in reinforcement learning with selective noise injection and information bottleneck. In: Proceedings of Advances in Neural Information Processing Systems, 2019
7 You B, Liu H, Peters J, et al. Rollout Total Correlation for Deep Reinforcement Learning. Trans Mach Learn Res, 2025