A BENCHMARKING METHODS

A.1 RE3

Given a trajectory $\{\boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{a}_{T-1}, \boldsymbol{s}_T\}$ collected the agent, RE3 first uses a random encoder to encode the visited states before using a k-NN estimator to compute the state entropy (Seo et al., 2021). Denote by $\mathcal{E} = \{\boldsymbol{e}_i\}_{i=1}^{T}$ the encoding vectors, RE3 proposes to define the intrinsic reward as

$$\hat{r}(\boldsymbol{s}_t) = \log\left(\mu_k(\boldsymbol{e}_t, \mathcal{E}) + 1\right). \tag{8}$$

Then the intrinsic reward can be used to online RL and unsupervised pre-training.

A.2 RIDE

RIDE is built based on the ICM (Pathak et al., 2017), which trains a forward and an inverse dynamics model to learn the representation of the state space. Denote by $\phi(\boldsymbol{s})$ the state representation, RIDE computes the intrinsic reward as the following Euclidean distance:

$$\hat{r}(\boldsymbol{s}_t) = \frac{\|\phi(\boldsymbol{s}_{t+1}) - \phi(\boldsymbol{s}_t)\|_2}{\sqrt{N_{ep}(\boldsymbol{s}_{t+1})}}, \tag{9}$$

where N_{ep} is the state visitation frequency during the current episode. N_{ep} is used to discount the intrinsic reward, which prevents the agent from lingering in a sequence of states with a large difference in their embeddings.

B DETAILS ON ATARI GAMES EXPERIMENTS

B.1 ENVIRONMENT SETTING

To handle the graphic observations, we stacked 4 consecutive frames as one state, and these frames were cropped to the size of (84, 84) to reduce the computational complexity. Moreover, we clip the extrinsic reward using a sign function to accelerate the training process. To simulate the sparse-reward scenario, we modified the original environment by adding a reward constraint, which will randomly set the value of extrinsic reward as zero.

B.2 EXPERIMENTAL SETTING FOR RL

In this work, we used a PyTorch implementation of A2C and PPO that can be found in (`https://github.com/DLR-RM/stable-baselines3`). An identical policy network and value network were employed for all methods to make a fair comparison, and their architectures are illustrated in Table 1 (LeCun et al., 2015). Here, 8×8 Conv. 32 represents a convolutional layer with 32 filters of size 8×8, and each convolutional layer is followed by a batch normalization (BN) layer (Ioffe & Szegedy, 2015; Agarap, 2018).

Table 1: The CNN-based network architectures.

Module	Policy network	Value network	Encoder
Input	States	States	States
Arch.	8×8 Conv 32, ReLU 4×4 Conv 64, ReLU 3×3 Conv 32, ReLU Flatten Dense 512, ReLU Dense $\lvert\mathcal{A}\rvert$ Categorical Distribution	8×8 Conv 32, ReLU 4×4 Conv 64, ReLU 3×3 Conv 32, ReLU Flatten Dense 512, ReLU Dense 1	8×8 Conv 32, ReLU 4×4 Conv 64, ReLU 3×3 Conv 32, ReLU Flatten Dense 512, ReLU Dense d
Output	Actions	Predicted values	Encoding vectors

For each game, we trained the agent for 10 million environment steps, in which the agent was set to interact with 10 parallel environments. Take PPO for instance, the agent sampled 256 steps in each

episode, producing 2560 pieces of transitions. After that, the transitions were used to update the policy network and value network. The agent was trained with a learning rate of 0.0003, an action entropy coefficient of 0.01, a value function coefficient of 0.5, a generalized-advantage-estimation (GAE) parameter of 0.95, and a gradient clipping threshold of 0.1 (Schulman et al., 2015). More detailed parameters of PPO and A2C can be found in Table 2 and Table 3.

Table 2: Hyparameters of PPO+REVD for Atari experiments.

Method	Hyperparameter	Value
PPO	Observation downsampling	(84, 84)
	Stacked frames	4
	Environment steps	10000000
	Number of workers	10
	Episode steps	256
	Optimizer	Adam
	Learning rate	0.0003
	GAE coefficient	0.95
	Action entropy coefficient	0.05
	Value loss coefficient	0.5
	Max gradient norm	0.5
	Value clipping coefficient	0.2
	Batch size	64
	Update epochs	5
	Gamma γ	0.99
REVD	Embedding size d	128
	k	5
	α	0.5
	λ_0	0.1
	κ	0.00001
	ϵ	0.0001

Table 3: Hyparameters of A2C+REVD for Atari experiments.

Method	Hyperparameter	Value
A2C	Observation downsampling	(84, 84)
	Stacked frames	4
	Environment steps	10000000
	Number of workers	10
	Episode steps	32
	Optimizer	Adam
	Learning rate	0.0003
	GAE coefficient	0.95
	Action entropy coefficient	0.01
	Value loss coefficient	0.5
	Max gradient norm	0.5
	Value clipping coefficient	0.2
	Batch size	32
	Gamma γ	0.99
REVD	Embedding size d	128
	k	5
	α	0.5
	λ_0	0.1
	κ	0.00001
	ϵ	0.0001

B.3 Experimental Setting for Exploration Methods

REVD. We employed a randomly-initialized and fixed encoder for encoding the state space, whose architecture is illustrated in Table 1. At the end of each episode, the transitions were used to compute the mixed rewards using Eq. (7), where $k = 5, \alpha = 0.5, \lambda_0 = 0.1, \kappa = 0.00001$ and $\epsilon = 0.0001$. In particular, since we considered the on-policy setting, the k-NN searching operation for a state was only performed within its own trajectory.

RE3. We followed the implementation of RE3 in a publicly released repository (`https://github.com/younggyoseo/RE3`), which uses intrinsic reward $\hat{r}(\boldsymbol{s}_t) = \mu_k(\boldsymbol{e}_t, \mathcal{E})$ without log exploration. Then the intrinsic reward was combined with the extrinsic reward to make a mixed reward $r_t^{\text{total}} = \check{r}(\boldsymbol{s}_t, \boldsymbol{a}_t) + \lambda_t \cdot \hat{r}(\boldsymbol{s}_t)$, where $\lambda_t = \lambda_0(1-\kappa)^t$. Here, we set $\lambda = 0.05$ and $\kappa \in \{0.001, 0.0001, 0.00001\}$ and $k \in \{5, 10\}$. Similar to REVD, the k-NN searching operation for a state was also performed within its own trajectory.

RIDE. We follow the implementation of RIDE in a publicly released repository (`https://github.com/facebookresearch/impact-driven-exploration`). In practice, we trained a single forward dynamics model g to predict the encoded next-state $\phi(\boldsymbol{s}_{t+1})$ based on the current encoded state and action $(\phi(\boldsymbol{s}_t), \boldsymbol{a}_t)$, whose loss function is $\|g(\phi(\boldsymbol{s}_t), \boldsymbol{a}_t) - \phi(\boldsymbol{s}_{t+1})\|_2$. To compute the state visitation frequency of $\boldsymbol{s}_{t+1}$, we used a pseudo-count method that approximates the frequency using the k-NN distance within episode (Badia et al., 2020).

C Details on PyBullet Experiments

C.1 Environment Setting

We further tested REVD on six tasks from PyBullet Robotics Environments, namely *Ant*, *Cart Pole*, *Half Cheetah*, *Hopper*, *Humanoid*, and *Walker 2D*. The source code of these environments can be found in (`https://github.com/bulletphysics/bullet3`).

C.2 Experimental Setting for RL

Since PyBullet tasks provided low-dimensional features as observations, multilayer perceptron (MLP) (LeCun et al., 2015) was used to build the policy network and value network, whose architectures are illustrated in Table 4.

Table 4: The MLP-based network architectures.

Module	Policy network	Value network	Encoder
Input	States	States	States
Arch.	Dense 64, Tanh Dense 64, Tanh Dense dim($\mathcal{A}$) Gauss Distribution	Dense 64, Tanh Dense 64, Tanh Dense 1	Dense 64, ReLU Dense 64, ReLU Dense d
Output	Actions	Predicted values	Encoding vectors

For each PyBullet task, we trained the agent for 2 million environment steps, in which the agent was also set to interact with 10 parallel environments. Take PPO for instance, the agent sampled 128 steps in each episode, producing 1280 pieces of transitions. After that, the transitions were used to update the policy network and value network. The agent was trained with a learning rate of 0.0003, an action entropy coefficient of 0.01, a value function coefficient of 0.5, a generalized-advantage-estimation (GAE) parameter of 0.95, and a gradient clipping threshold of 0.1 (Schulman et al., 2015). More detailed parameters of PPO and A2C can be found in Table 5 and Table 6.

C.3 Experimental Setting for Exploration Methods

For REVD, We also used a randomly-initialized and fixed encoder to encode the state space, whose architecture is shown in Table 4. In each episode, the visited states were first encoded as vectors

Table 5: Hyparameters of PPO+REVD for PyBullet experiments.

Method	Hyperparameter	Value
PPO	Environment steps	2000000
	Number of workers	10
	Episode steps	128
	Optimizer	Adam
	Learning rate	0.0003
	GAE coefficient	0.95
	Action entropy coefficient	0.01
	Value loss coefficient	0.5
	Max gradient norm	0.5
	Value clipping coefficient	0.2
	Batch size	64
	Update epochs	5
	Gamma γ	0.99
REVD	Embedding size d	64
	k	3
	α	0.5
	λ_0	0.1
	κ	0.00001
	ϵ	0.0001

Table 6: Hyparameters of A2C+REVD for PyBullet experiments.

Method	Hyperparameter	Value
A2C	Environment steps	2000000
	Number of workers	10
	Episode steps	8
	Optimizer	Adam
	Learning rate	0.0003
	GAE coefficient	0.95
	Action entropy coefficient	0.01
	Value loss coefficient	0.5
	Max gradient norm	0.5
	Value clipping coefficient	0.2
	Batch size	32
	Gamma γ	0.99
REVD	Embedding size d	64
	k	3
	α	0.5
	λ_0	0.1
	κ	0.00001
	ϵ	0.0001

with a dimension of 64. Then the encoding vectors were used to compute intrinsic rewards, where $k = 3, \alpha = 0.5, \lambda_0 = 0.1, \kappa = 0.00001$, and $\epsilon = 0.0001$. Finally, the augmentated transitions were used to update the policy network and value network. For RE3 and RIDE, we followed the similar procedures elaborated in Atari experiments and reported the best result.