A Implementation

In this part, we introduce our detailed implementation of EfficientImitate. Our code is available at `https://github.com/zhaohengyin/EfficientImitate`.

A.1 Model Details

Our model is based on the EfficientZero [10] model. It is composed of several neural networks: representation network f, dynamics network g, value network V, policy network π, BC policy network π_{BC}, discriminator network D, projector network, and predictor network. We use PyTorch [6] to implement these networks. Their detailed structures are as follows.

A.1.1 State-based experiments

Representation Network The representation network is an MLP with one hidden layer of size 256. Its output dimension is 128. It uses LeakyReLU [4] as the hidden layer's activation function. The output activation function is identity.

Dynamics Network The dynamics network is an MLP with one hidden layer of size 256. It concatenates the representation and the action as the input. Its output dimension is 128 (representation's dimension). It uses LeakyReLU as the hidden layer's activation function. The output activation function is identity.

Value Network The value network is an MLP with one hidden layer of size 128 (256 for Humanoid). It uses LeakyReLU as the hidden layer's activation function. The output activation function is identity. We use the categorical value representation introduced in MuZero [7]. The value prediction is discretized into 401 bins to represent the value between $[-100, 100]$. Therefore, the output dimension of value network is 401.

Policy & BC Network The policy and the BC policy networks are both MLPs with one hidden layer of size 128 (256 for Humanoid). These MLPs use LeakyReLU as the hidden layer's activation function. Their output activation functions are identity. They produce a Squarshed-Normal (Tanh-Normal) action distribution [1], which is determined by a predicted mean and a predicted logstd. Therefore, the output dimension is twice of the dimension of the action space.

Discriminator Network The discriminator network is an MLP with one hidden layer of size 128. It uses LeakyReLU as hidden layer's activation function. The output activation function is sigmoid.

Projector Network The projector network is an MLP with one hidden layer of size 512. It uses ReLU [4] as the hidden layer's activation function. The output activation function is identity. The output dimension (projection dimension) is 128.

Predictor Network The predictor network is an MLP with one hidden layer of size 512. It uses ReLU as the hidden layer's activation function. The output activation function is identity. The output dimension is 128.

A.1.2 Image-based experiments

Representation Network The representation network consists of a four layer convolutional neural network and an MLP. Its structure is as follows.

- Convolution layer. Input dim: 12. Output dim: 32. Kernel Size: 5. Stride: 2. Padding: 2.
- ReLU activation.
- Convolution layer. Input dim: 32. Output dim: 32. Kernel Size: 3. Stride: 2. Padding: 1.
- ReLU activation.
- Convolution layer. Input dim: 32. Output dim: 32. Kernel Size: 3. Stride: 2. Padding: 1.
- ReLU activation.

- Convolution layer. Input dim: 32. Output dim: 32. Kernel Size: 3. Stride: 2. Padding: 1.
- Flatten.
- Linear layer. Output dim: 128.
- ReLU activation.

Dynamics Network The dynamics network is an MLP with one hidden layer of size 256. It concatenates the representation and the action as the input. Its output dimension is 128 (representation's dimension). It uses ReLU as the hidden layer's activation function. The output activation function is also ReLU.

Value Network The value network is an MLP with two hidden layers of size 100. It uses ReLU as the hidden layer's activation function. The output activation function is identity. We use the categorical value representation introduced in MuZero. The value prediction is discretized into 401 bins to represent the value between $[-40, 40]$. Therefore, the output dimension of value network is 401.

Policy & BC Network The policy and the BC policy networks are both MLPs with two hidden layers of size 100. These MLPs use ReLU as the hidden layer's activation function. Their output activation functions are identity. They produce a Squarshed-Normal action distribution, which is determined by a predicted mean and a predicted logstd. Therefore, the output dimension of them is twice of the dimension of the action space.

Discriminator Network The discriminator network is an MLP with one hidden layer of size 100. It uses ReLU as hidden layer's activation function. The output activation function is sigmoid.

Projector Network The projector network is an MLP. Its structure is as follows.

- Linear layer. Output dim: 1024. BatchNorm [3] with momentum 0.1. ReLU activation.
- Linear layer. Output dim: 1024. BatchNorm with momentum 0.1. ReLU activation.
- Linear layer. Output dim: 1024. BatchNorm with momentum 0.1.

Predictor Network The predictor network is an MLP. Its structure is as follows.

- Linear layer. Output dim: 512. BatchNorm with momentum 0.1. ReLU activation.
- Linear layer. Output dim: 1024.

A.2 MCTS Details

Our MCTS implementation is mainly based on the Sampled MuZero [2]. We also apply modifications proposed by the EfficientZero. The detailed procedure is as follows.

Expansion For the task having a continuous action space, we can not enumerate all the possible actions at a node as the original MCTS algorithm. To solve this problem, we use the sampling method proposed by the Sampled MuZero. For the expansion of a node s (in the representation space), we sample K actions $\{a_i\}_{i=1}^K$ from current policy $\pi(a|s)$. In this work we propose to integrate BC actions into MCTS, then we actually sample from

$$\tilde{\pi}(a|s) := (1-\alpha)\pi(a|s) + \alpha\pi_{BC}(a|s). \tag{1}$$

Here $\alpha = 0.25$ is a mixture factor.

Selection For the action selection, we selects action a^* from the sampled actions that maximize the probabilistic upper confidence bound

$$a^* = \arg\max_{a\in\{a_i\}} Q(s,a) + c(s)\hat{\pi}(a|s)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}, \tag{2}$$

where $\hat{\pi}(a|s) = \frac{1}{K}\sum_i \delta(a, a_i)$. $Q(s,a)$ is the current Q-estimation of the pair (s,a). $N(s,a)$ denotes the times that this pair is visited in MCTS. $c(s)$ is a weighting coeffcient defined by

$$c(s) = c_1 + \log \frac{1 + c_2 + \sum_b N(s,b)}{c_2}, \tag{3}$$

where $c_1 = 1.25$, $c_2 = 19625$.

To encourage exploration, we also inject Dirichlet noise to $\hat{\pi}(a|s)$ at the root node. So $\hat{\pi}(a|s)$ becomes

$$\hat{\pi}(a|s) := (1-\rho)\hat{\pi}(a|s) + \rho N_{\mathcal{D}}(\xi). \tag{4}$$

Here, $\rho = 0.25$ is a mixture factor. $N_{\mathcal{D}}(\xi)$ is the Dirichlet distribution, and ξ is set to 0.3.

At the root node of MCTS, we use the discriminator network D and value network V to calculate Q-value by its original definition: $Q(s,a) = R(s,a) + \gamma V(g(s,a))$. At the other nodes, we use the mean Q-value calculation used by the EfficientZero.

Simulation & Backup The simulation and the backup process is the same as EfficientZero's implementation, and we refer the readers to EfficientZero for details.

A.3 Training Details and Hyperparameters

Finally, we introduce some important training details and the hyperparameters.

Initialization We initialize the weights and biases of the last layer of policy, BC policy, value, and discriminator network to be zero. The other parameters are initialized by the default initializers in PyTorch.

Discriminator In AIL training, it is very useful to apply the gradient penalty to the discriminator [5]. We also apply gradient penalty to the discriminator network.

Target Update We propose to use a target model for the calculation of policy, value, and AIL reward during reanalyze. The target model is updated periodically during training subject to an update frequency.

The training hyperparameters used in the state-based experiments are in Table 1. The training hyperparameters used in the image-based experiments are in Table 2.

B Environment Details

B.1 State-based experiments

The setup of each task in the state-based experiments is in Table 3.

B.2 Image-based experiments

The setup of of each task in the image-based experiments is in Table 4. We use a 48×48 resolution in the image-based experiments.

C Other Ablations

We also perform ablations on the target discriminator and the multi-step discriminator loss. To evaluate the effect of the target discriminator, we use the latest model to calculate AIL reward in the value target. To evaluate the effect of the multi-step discriminator loss, we replace it with the single-step discriminator loss. We conduct experiments on the state-based and image-based Walker and Cheetah. The results are shown in Figure 2. We find that removing these components will not only lead to instability in training but also harm the performance. Compared with the target discriminator, the multi-step discriminator loss has a larger impact on the image-based tasks.

D Computation Resources

All of our experiments are conducted on a server with 4 NVIDIA RTX 3090 GPUs, 64 CPU cores, and 256GB RAM. For the most of state-based and image-based experiments except Humanoid Walk and image-based Hopper Hop, our experiments require 12-18 hours of training. The main bottleneck is at the Reanalyse [8, 10], where the minibatch cannot be produced and sent to the training loop at a high frequency. We are improving the computation efficiency by using better parallel computation implementation and applying MCTS speed up techniques.

E Visualization

One approach to interpret the learned model is by the t-SNE [9] plot. We use the image-based Walker experiment as an example. We use the trained model at 100k env steps to generate the state embeddings of one expert trajectory, and in environment trajectory at 0k, 25k, 50k, 75k, and 100k steps. Then we use t-SNE to visualize the embeddings on the 2D plane. As is shown in the Figure 1, the agent's trajectory gradually matches expert's trajectory (blue) during training. Moreover, the expert's trajectory has a circle structure, which represents the periodic pattern of the Walker's walking behavior. Therefore, our model can represent the environment in a meaningful way.

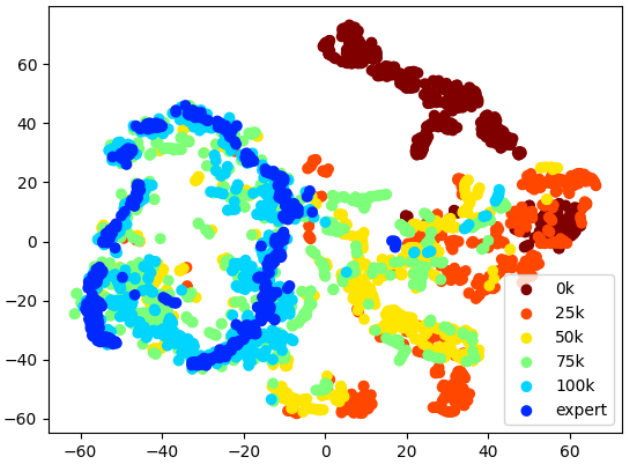

Figure 1: The t-SNE plot of the learned state embeddings.

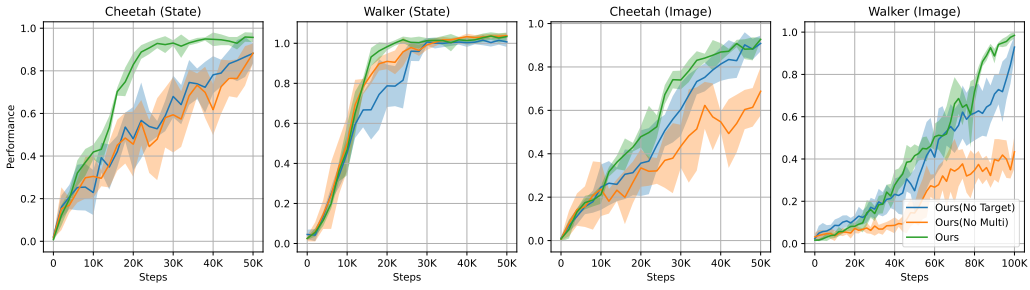

Figure 2: The ablation of target discriminator and multi-step discriminator loss. The results are averaged over three seeds. The shaded area displays the range of one standard deviation.

Table 1: Hyperparameters for the state-based experiments

Discount Factor	0.99
Minibatch Size	256
Optimizer	SGD
Optimizer: Learning Rate	0.01
Optimizer: Momentum	0.9
Optimizer: Weight Decay	1e-4
Maximum Gradient Norm	10
Unroll Steps	5
TD Steps	1
BC Loss Coeff.	0.01
Value Loss Coeff.	1.0
Policy Loss Coeff.	1.0
Discriminator Loss Coeff.	0.1 (1.0 for Humanoid)
Gradient Penalty	1.0
Target Update Interval	200
Consistency Loss Coeff.	2.0
Reanalyze Ratio	1.0
Number of Simulations in MCTS	50
Number of Sampled Actions	16
BC Ratio α	0.25

Table 2: Hyperparameters for the image-based experiments

Discount Factor	0.99
Minibatch Size	128
Stacked Frames	4
Optimizer	SGD
Optimizer: Learning Rate	0.02
Optimizer: Momentum	0.9
Optimizer: Weight Decay	1e-4
Maximum Gradient Norm	10
Unroll Steps	5
TD Steps	1
BC Loss Coeff.	0.01
Value Loss Coeff.	1.0
Policy Loss Coeff.	1.0
Discriminator Loss Coeff.	0.1
Gradient Penalty	1.0
Target Update Interval	200
Consistency Loss Coeff.	20.0
Reanalyze Ratio	1.0
Number of Simulations in MCTS	50
Number of Sampled Actions	16
BC Ratio α	0.25

Table 3: Task setup in the state-based experiments

Task	Action Repeat	Expert Performance
Cartpole Swingup	8	881.3
Ball-in-cup Catch	4	920.1
Reacher Easy	4	911.4
Finger Spin	4	574.2
Walker Walk	4	865.6
Cheetah Run	4	607.3
Hopper Hop	4	300.0
Humanoid Walk	2	782.8

Table 4: Task setup in the state-based experiments

Task	Action Repeat	Expert Performance
Cartpole Swingup	8	881.3
Ball-in-cup Catch	4	920.1
Reacher Easy	4	911.4
Finger Spin	4	574.2
Walker Walk	2	873.5
Cheetah Run	4	607.3
Hopper Hop	4	300.0

References

[1] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 2018.

[2] T. Hubert, J. Schrittwieser, I. Antonoglou, M. Barekatain, S. Schmitt, and D. Silver. Learning and planning in complex action spaces. In *International Conference on Machine Learning*, 2021.

[3] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

[4] A. L. Maas, A. Y. Hannun, A. Y. Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, 2013.

[5] M. Orsini, A. Raichuk, L. Hussenot, D. Vincent, R. Dadashi, S. Girgin, M. Geist, O. Bachem, O. Pietquin, and M. Andrychowicz. What matters for adversarial imitation learning? In *Neural Information Processing Systems*, 2021.

[6] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems*, 2019.

[7] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[8] J. Schrittwieser, T. Hubert, A. Mandhane, M. Barekatain, I. Antonoglou, and D. Silver. Online and offline reinforcement learning by planning with a learned model. In *Neural Information Processing Systems*, 2021.

[9] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(11), 2008.

[10] W. Ye, S. Liu, T. Kurutach, P. Abbeel, and Y. Gao. Mastering atari games with limited data. In *Neural Information Processing Systems*, 2021.