FTD Implementation

The SAC algorithm used in FTD is implemented based on the DMControl Generalization Benchmark (Hansen, Su, and Wang 2021); the network architecture and learning hyperparameters remain unchanged. For details on hyperparameters and network architecture, please refer to the "Hyperparameters" section and the "Network Architecture" section. The training of FTD is conducted on NVIDIA GeForce RTX 3090 and each seed takes about 3 days.

When applying Mobile SAM to the noisy observations in our experiments, the number of segments usually exceeds 20. Owing to computational resource limitations, we filter the results according to the following criteria. First, if there are more than 95% overlap between two segments, the smaller segment will be discarded. Second, we eliminate segments that exceed a certain range of area, indicating that SAM has erroneously segmented multiple objects together or produced meaningless segments. For finger, cheetah, cartpole, and pendulum, the range is $[100, 300]$, for other environments, the range is $[100, 2000]$. Third, for finger and franka environments, similar to random crop, segments that are entirely within 10 pixels of the image edge will be discarded. Finally, we select up to 9 segments from the current segmentation, choosing them in descending order of size.

For sparse-reward environments (hopper and pendulum), to stabilize the effect of reward predictor, instead of random sampling from the replay buffer, the agent will only sample transitions with rewards in the top 5000.

To save training costs, the update frequency of reward predictor and inverse dynamic predictor will gradually decrease. For every 50k steps of training, the hyperparameter "steps per perdictor update" in Table 4 increases by 1.

Baseline Implementation

DBC, DrQ-v2, and Denoised-MDP are implemented following the open-source code, below are the links of them. MICo and Q^2-learning are implemented according to their original papers, and are included in the code we provide.

Table 3: Links to the open-source code of baseline methods.

Method	Open-Source URL
DrQ-v2	https://github.com/facebookresearch/drqv2
DBC	https://github.com/facebookresearch/deep_bisim4control
Denoised-MDP	https://github.com/facebookresearch/denoised_mdp

SAC Algorithm

SAC is mainly composed of two parts: actor $\pi(a \mid o)$ and critic $Q(o, a)$. The critic parameter is learned by minimizing the Bellman loss using transition (o_t, a_t, o_{t+1}, r_t) sampled from replay buffer $\mathcal{D}$:

$$\mathcal{L}_Q = \mathbb{E}_{\tau\sim\mathcal{D}}\left[\left(Q\left(o_t, a_t\right) - \left(r_t + \gamma V\left(o_{t+1}\right)\right)\right)^2\right]. \tag{8}$$

Here $V\left(o_{t+1}\right)$ is the estimation value of the next observation, which can be written as:

$$V\left(o_{t+1}\right) = \mathbb{E}_{a'\sim\pi}\left[\bar{Q}\left(o_{t+1}, a'\right) - \alpha \log \pi\left(a' \mid o_{t+1}\right)\right], \tag{9}$$

where $\bar{Q}$ is the slowly updated copy of Q. Finally, actor $\pi(a \mid o)$ is trained by approximating the exponential of the soft-Q function, written as the following loss function:

$$\mathcal{L}_\pi = -\mathbb{E}_{a\sim\pi,\tau\sim\mathcal{D}}\left[Q\left(o_t, a\right) - \alpha \log \pi\left(a \mid o_t\right)\right]. \tag{10}$$

Network Architecture

Below are the network architectures of the main components of FTD. Here, `MLP(n)` denotes a fully-connected layer with output size of n; `Conv2D(c, k, s, p)` denotes a 2D convolution layer of output channel c, kernel size k, stride s, and padding p; `Maxpool2D(k, s)` denotes a 2D max-pooling layer of kernel size k and stride s; `Flatten()` denotes a flatten layer; `ReLU()` denotes a rectified linear unit; $\{\cdots\} \times k$ denotes repeating the layers within brace for k times.

Feature extractor ϕ `Conv2D(32,3,2,1)=>{ReLU()=>Conv2D(32,3,1,1)=>Maxpool2D(2,2)}`$\times 5$ `=>Flatten()=>MLP(128)`

Projection W_i^k and W_i^q $(i = 1, 2, 3, 4)$ `MLP(128)`

Feature extractor of o_t^{sel} `Conv2D(32,3,2,0)=>{ReLU()=>Conv2D(32,3,1,0)}`$\times 11$`=>Flatten()=>MLP(100)`

Reward predictor $\hat{\mathcal{R}}$ `MLP(1204)=>MLP(1024)=>MLP(1)`

Inverse dynamic predictor $\hat{\mathcal{P}}$ `MLP(1204)=>MLP(1024)=>MLP(action dim)`

Actor π `MLP(1204)=>MLP(1024)=>MLP(2*action dim)`

Critic Q `MLP(1204)=>MLP(1024)=>1`

Hyperparameters

Below are the hyperparameters for reproducing the experiments.

Table 4: Hyperparameters.

Hyperparameters in environments	
frame size	84×84
frame stack	3
episode length	1000
seeds	0, 1, 2
action repeat	1 (finger-spin), 4 (otherwise)
Hyperparameters of SAC	
γ	0.99
replay buffer size	1×10^5
batch size	128
optimizer	Adam
actor/critic learning rate	3×10^{-4}
steps per actor/critic update	2
initial temperature	0.1
warmup steps	1×10^5 (franka), 1×10^4 (otherwise)
Hyperparameters of self-supervised objectives	
η_1	1
η_2	1
η_3	1
optimizer	Adam
steps per predictor update	1
predictor learning rate	1×10^{-4}
warmup steps	1×10^5 (franka), 1×10^4 (otherwise)
Hyperparameters of MobileSAM	
pred iou thresh	0.5
stability score thresh	0.5
points per side	8
points per batch	64

Visualization Comparison between FTD and Denoised-MDP on Franka-reach

Below is the comparison between the selected frame of FTD and the reconstruction frame of Denoised-MDP on franka-reach.

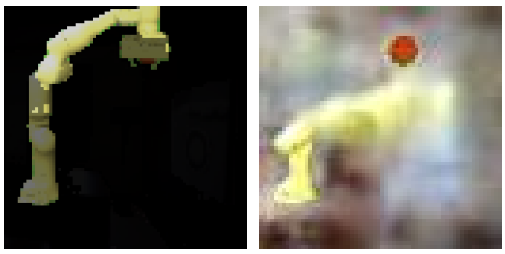
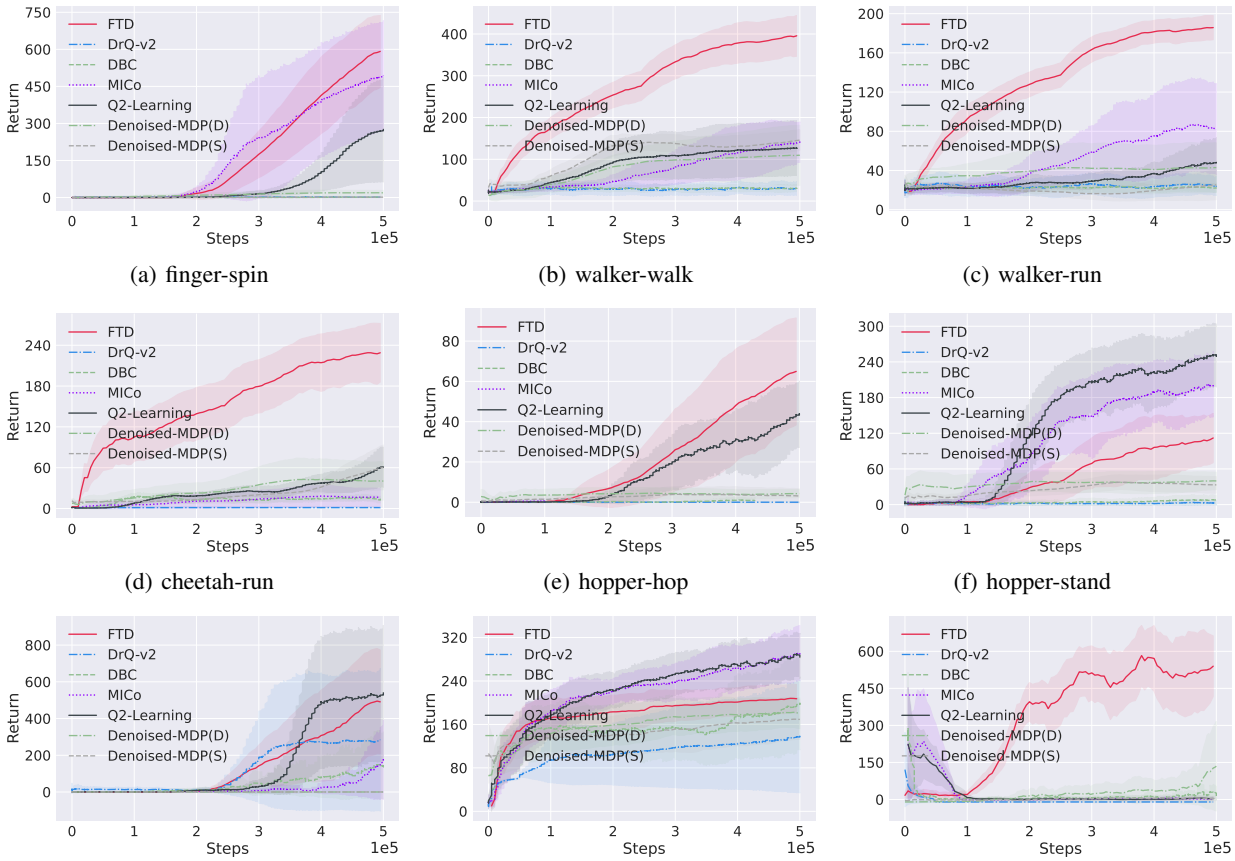


Figure 5: (Left) Selected frame of FTD. (Right) Reconstructed frame of Denoised-MDP(D).

Training Curves of FTD and Baselines



(a) finger-spin (b) walker-walk (c) walker-run

(d) cheetah-run (e) hopper-hop (f) hopper-stand

(g) pendulum-swingup (h) cartpole-swingup (i) franka-reach

Figure 6: Training curves of FTD and baselines.

References

Hansen, N.; Su, H.; Wang, X. 2021. Stabilizing Deep Q-Learning with ConvNets and Vision Transformers under Data Augmentation. In *Advances in Neural Information Processing Systems*, 3680-3693.