A Brief Description of ADVISOR

ADVISOR [25] minimizes a weighted loss, including the standard imitation loss and RL losses (*e.g.*, policy and critic losses from A2C [11] or PPO [9]).

$$\mathcal{L}^{\text{ADV}} = \mathbb{E}_{h,s\sim\mathcal{D}}\left[w(s)D\left(\mu(s),\pi(h)\right) + (1-w(s))\mathcal{L}_{\text{RL}}\right] , \tag{8}$$

where $w(s) : \mathcal{S} \to [0,1]$, μ is an optimal fully-observable policy and $\mathcal{D}$ is a replay buffer of (truncated) episodes. The weight function $w(s)$ is defined per state $s \in \mathcal{S}$ as:

$$w(s) = m_\alpha(D(\mu, \pi^{\text{IL}})) , \tag{9}$$

with $m_\alpha(x) = \exp(-\alpha x)$, and where π^{IL} is an additional history actor that is trained using the imitation loss alone, *i.e.*, by optimizing $\mathcal{L}^{\text{ADV}}$ in Equation (8) without the RL losses and $w(s) = 1$.

B Cross-Observability Imitation Learning and the Optimality Gap

In this section, we define the optimality gap, a theoretical quantity that measures how useful the state expert is in providing supervision to learn a history policy in a behavioral cloning setting. To formalize this gap, we first consider learning a history policy π by imitating a state expert μ, by minimizing the following *cross-observability imitation learning* (COIL) objective,

$$J_\pi = \mathbb{E}_{h,s\sim\mathcal{D}}\left[D(\mu(s),\pi(h))\right] , \tag{10}$$

where $\mathcal{D}$ is a replay buffer containing (truncated) episodes. We denote the optimal solution to COIL as π^{COIL} and denote its performance as $V^{\text{COIL}}(\mu)$, which is a function of the state expert μ. On the other hand, we denote the performance of an optimal partially observable agent as V^*. Then, we can define the *optimality gap* as the difference between the optimal performance achievable by a partially observable policy and the performance of the COIL policy,

$$G(\mu) = V^* - V^{\text{COIL}}(\mu) . \tag{11}$$

The process of optimizing the objective in Equation (10) can be considered as a projection of the state expert μ that assumes full state access into the partially observable setting. Such projection might be useful under the asymmetric learning setting in which we want to leverage privileged information (*i.e.*, having access to states and a state expert) during training to learn a policy that can later act relying on non-privileged information (*i.e.*, the histories).

Our optimality gap is different from the imitation gap [25], which is a measure of the behavior difference between π^{IL} and μ. In contrast, the optimality gap is a measure of the performance difference between π^* and π^{COIL}, which is more relevant for the problem of partially observable control.

C Soft Actor-Critic for Discrete Action

When the action space is discrete, some simplifications become possible [36]. The policy model can output the full distribution vector over actions $\mu_\theta \colon \mathcal{S} \to [0,1]^{|\mathcal{A}|}$, and the value model can also change respectively to $Q_\phi \colon \mathcal{S} \to \mathbb{R}^{|\mathcal{A}|}$. Then, Equations (2), (4) and (5) become

$$J_\mu(\theta) = \mathbb{E}_{s\sim\mathcal{D}}\left[\mu_\theta(s)^\top\left(Q_\phi(s) - \alpha\log\mu_\theta(s)\right)\right] , \tag{12}$$

$$V(s) = \mu_\theta(s)^\top\left(Q_{\bar{\phi}}(s) - \alpha\log\mu_\theta(s)\right) , \tag{13}$$

$$J_\alpha(\alpha) = \alpha\,\mathbb{E}_{s\sim\mathcal{D}}\left[-\mu_\theta(s)^\top\log\mu_\theta(s)\right] - \alpha\bar{H} . \tag{14}$$

D Algorithms

Here, we describe two versions of COSIL for continuous and discrete action spaces. We used the SAC algorithm in [42] as our base algorithm.

D.1 COSIL (Continuous Action Spaces)

```
 1: Initial history policy $\pi_\theta(h)$ represented as a Gaussian with mean model $m_\theta(h)$ and a standard
    deviation model $s_\theta(h)$
 2: Q-function parameters $\phi_1, \phi_2$
 3: Empty replay buffer $\mathcal{D} \leftarrow \emptyset$
 4: Set target parameters equal to main parameters: $\bar{\phi}_1 \leftarrow \phi_1, \bar{\phi}_2 \leftarrow \phi_2$
 5: *Deterministic* state policy $\mu$
 6: Use the divergence $D(\mu(s), \pi_\theta(h)) = (\mu(s) - m_\theta(h))^2$
 7: Set divergence target $\bar{D}$
 8: **repeat**
 9:     Sample action $a \sim \pi_\theta(h)$ and execute $a$ in the environment
10:     Observe next history $h'$, reward $r$, and done signal $d$
11:     If done, reset the environment and store the episode in $\mathcal{D}$
12:     **if** time to update **then**
13:         **for** $j \leftarrow 1, \ldots, \#$updates **do**
14:             Random sample a batch of $B$ episodes from $\mathcal{D}$
15:             Compute targets for the Q functions:
16:             $y \leftarrow \begin{cases} r & \text{if episode done} \\ r + \gamma \left( \min_{i\in\{1,2\}} Q_{\bar{\phi}_i}(h', \tilde{a}') + \alpha \left(\mu(s') - m_\theta(h')\right)^2 \right) & \text{if episode not done} \end{cases}$
    ,
17:             where $\tilde{a}' \sim \pi_\theta(h')$
18:             Update Q-function by one step of gradient descent using
19:             $\mathcal{L}_{\phi_i} \leftarrow \frac{1}{|B|} \sum_{h,s\in B} \left(Q_{\phi_i}(h,a) - y\right)^2$ for $i \in \{1,2\}$
20:             Update policy by one step of gradient descent using
21:             $\mathcal{L}_\theta \leftarrow \frac{1}{|B|} \sum_{h,s\in B} \left( - \min_{i\in\{1,2\}} Q_{\phi_i}(h, \tilde{a}) - \alpha \left(\mu(s) - m_\theta(h)\right)^2 \right)$,
22:             where $\tilde{a} \sim \pi_\theta(h)$
23:             Adjust $\alpha$ by one step of gradient descent using
24:             $\mathcal{L}_\alpha \leftarrow \frac{1}{|B|} \sum_{h,s\in B} \left( \alpha\bar{D} - \alpha \left(\mu(s) - m_\theta(h)\right)^2 \right)$
25:             Update target networks with
26:             $\bar{\phi}_i \leftarrow \tau\bar{\phi}_i + (1-\tau)\phi_i$ for $i \in \{1,2\}$
27:         **end for**
28:     **end if**
29: **until** convergence
```

D.2 COSIL (Discrete Action Spaces)

```
1: Initial history policy $\pi_\theta$ with parameter $\theta$
2: Q-functions with parameters $\phi_1, \phi_2$
3: Empty replay buffer $\mathcal{D} \leftarrow \emptyset$
4: Set target parameters equal to main parameters: $\bar{\phi}_1 \leftarrow \phi_1$, $\bar{\phi}_2 \leftarrow \phi_2$
5: *Deterministic* state expert $\mu$
6: Use the cross-entropy divergence $D(\mu(s), \pi_\theta(h)) = -\mu(s)^\top \log(\pi_\theta(h))$
7: Set cross-entropy divergence target $\bar{D}$
8: **repeat**
9:     Sample action $a \sim \pi_\theta(h)$ and execute $a$ in the environment
10:    Observe next history $h'$, reward $r$, and done signal $d$
11:    If done, reset the environment and store the episode in $\mathcal{D}$
12:    **if** time to update **then**
13:        **for** $j \leftarrow 1, \dots, \#$updates **do**
14:            Random sample a batch of $B$ episodes from $\mathcal{D}$
15:            Compute targets for the Q functions:
16:            $y \leftarrow \begin{cases} r & \text{if episode done} \\ r + \gamma \pi_\theta(h')^\top \left(\min_{i \in \{1,2\}} Q_{\bar{\phi}_i}(h') + \alpha \mu(s')^\top \log(\pi_\theta(h'))\right) & \text{if episode not done} \end{cases}$
17:            Update Q-function by one step of gradient descent using
18:            $\mathcal{L}_{\phi_i} \leftarrow \frac{1}{|B|} \sum_{h,s \in B} \left(Q_{\phi_i}(h,a) - y\right)^2$ for $i \in \{1,2\}$
19:            Update policy by one step of gradient descent using
20:            $\mathcal{L}_\theta \leftarrow \frac{1}{|B|} \sum_{h,s \in B} \pi_\theta(h)^\top \left(-\min_{i \in \{1,2\}} Q_{\phi_i}(h, \pi_\theta(h)) - \alpha \mu(s)^\top \log(\pi_\theta(h))\right)$
21:            Adjust $\alpha$ by one step of gradient descent using
22:            $\mathcal{L}_\alpha \leftarrow \frac{1}{|B|} \sum_{h,s \in B} \pi_\theta(h)^\top \left(\alpha \bar{D} + \alpha \mu(s)^\top \log(\pi_\theta(h))\right)$
23:            Update target networks with
24:            $\bar{\phi}_i \leftarrow \tau \bar{\phi}_i + (1-\tau)\phi_i$ for $i \in \{1,2\}$
25:        **end for**
26:    **end if**
27: **until** convergence
```

E Domain Details

Below are more details about the domains. We use Pybullet [43] and BulletArm [44] to implement `Block-Picking`, and MuJoCo [45] for `Bumps-1D` and `Bumps-2D`.

E.1 Bumps-1D

- **Action**: Compliant/Stiff Finger $\times$ Left/Right
- **Observation**: The position and the deflected angle of the finger from the vertical axis
- **Reward**: Get a positive reward of 1.0 *only* when the finger can push the right bump to the right more than a threshold
- **Episode Termination**: Either bump is moved more than a threshold or an episode lasts more than 100 timesteps
- **State Expert**: Move to the left side of the rightmost bump using a compliant finger, then push the bump with a stiff finger

E.2 Bumps-2D

- **Action**: Left, Right, Up, Down, Stay
- **Observation**: The position and the deflected angle from the vertical axis of the finger
- **Reward**: Get a positive reward of 1.0 *only* when the finger reaches the bigger bump and then performs Stay
- **Episode Termination**: A Stay action is performed, or an episode lasts more than 100 timesteps
- **State Expert**: Going to the bigger bump using the shortest path

E.3 Minigrid-Memory

- **Action**: Turn-Left, Turn-Right, Move-Forward
- **Observation**: A partially observable view of the environment of size 7x7x3
- **Reward**: step reward: -0.05, reaching the correct matching object: -5.0, reaching the wrong object: 5.0
- **Episode Termination**: Reaching any object on the right or an episode lasts more than 100 timesteps
- **State Expert**: Going towards the matching object on the right using the shortest path

E.4 Car-Flag

- **Action**: Continuous power in [-1.0, 1.0]
- **Observation**: The position and the velocity of the car, the side of the green flag (-1 or 1 if the car is near the blue flag, and 0 otherwise)
- **Reward**: step reward: -0.01, reaching the green flag: 1.0, and reaching the red flag: -1.0
- **Episode Termination**: Reaching either flags or an episode lasts more than 160 timesteps
- **State Expert**: Depending on the side of the green flag, the state expert always goes to that side with maximum power

E.5 LunarLander-P, -V

An observation in `LunarLander` includes 8 information fields: the XY coordinates of the lander (0, 1), its linear XY velocities (2, 3), its angle (4), its angular velocity (5), and two booleans (6, 7) that represent whether each leg is in contact with the ground or not. In `LunarLander-P`, the agent can observe fields (0, 1, 4, 6, 7). In `LunarLander-V`, the agent can observe fields (2, 3, 5, 6, 7). We limit the maximum episode length to 160. The rewards, actions, and termination conditions are described at `https://github.com/openai/gym/blob/master/gym/envs/box2d/lunar_lander.py`. We train the state expert using SAC with full state access using Stable-Baselines3 [46] in 500k timesteps.

E.6 Block-Picking

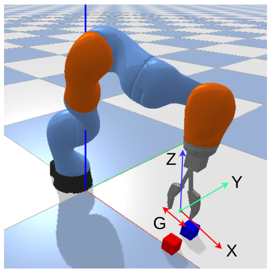


Figure 7: `Block-Picking`.

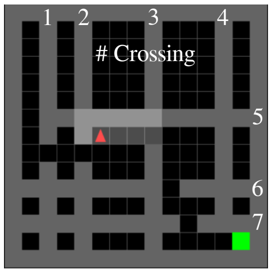


Figure 8: `Minigrid-Crossing`.

- **Action**: $(\delta_g, \delta_x, \delta_y, \delta_z)$, where $\delta_g \in [0, 1]$ is the absolute openness of the gripper, and $\delta_x, \delta_y, \delta_z \in [-0.05, 0.05]$ are the relative movements of the gripper from the previous timestep
- **Observation**: a depth image of size 84x84x1, the status of the gripper (holding an object or not), and the current reward 0 or 1 (to signify whether a box is picked successfully). We broadcast the last two fields and combine them with the depth image to create a unified observation of size 84x84x3
- **Reward**: A reward of 1.0 only when the movable block is picked and brought higher than a certain height
- **Episode Termination**: The movable block is picked successfully, or an episode lasts more than 50 timesteps
- **State Expert**: By following moving and picking waypoints computed using the poses of the movable block and the gripper, the expert always picks the movable block successfully

E.7 Minigrid-Crossing

The domain is visualized in Figure 8. The number of crossings is the total number of columns/rows made of stone.

- **Action**: Turn-Left, Turn-Right, Move-Forward
- **Observation**: A partial ego-centric view of size 7x7x3 around the agent
- **Reward**: A reward of 1 when reaching the goal, 0 otherwise
- **Episode Termination**: Reaching the target or an episode lasts more than 100 timesteps
- **State Expert**: Going towards the goal using the shortest path

F Learned Policies

In this section, we describe the policies learned by COSIL. Please see our project website at `https://sites.google.com/view/cosil-corl22` for videos that visualize these policies.

F.1 Bumps-1D

As shown in Figure 9, COSIL acts differently depending on the relative of the initial position of the gripper to the positions of the two bumps (Left, Middle, or Right). COSIL acts similarly to the state expert only when the gripper is initialized on the left of the two bumps. However, when the gripper starts in the middle or on the right of the two bumps, COSIL needs to perform additional informative actions with a compliant finger, unlike the state expert. This ensures it knows where the rightmost bump is before switching to a stiff finger to perform a push to the right.

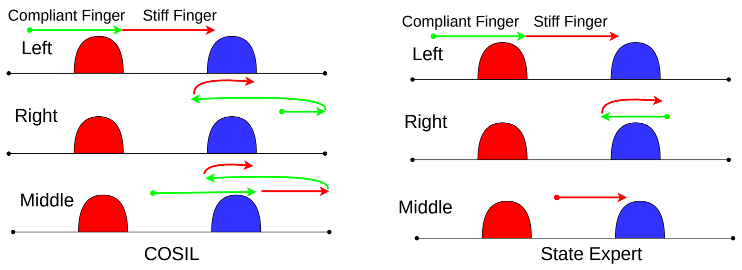


Figure 9: The policy learned by COSIL in `Bumps-1D` [6, 32] behaves differently depending on the relative position of the gripper and the positions of the two bumps.

F.2 Bumps-2D

Figure 10 shows that COSIL behaves differently depending on which bump is found first. If the agent (magenta) finds the bigger bump (blue) first, it will keep looking for the smaller bump (red). When it finds that bump, it will quickly go back to the location of the bigger bump (it memorizes earlier) and perform a Stay action to finish the task. If the smaller bump is found first, the agent will explore to find the bigger bump and immediately perform Stay when it touches the bigger bump.

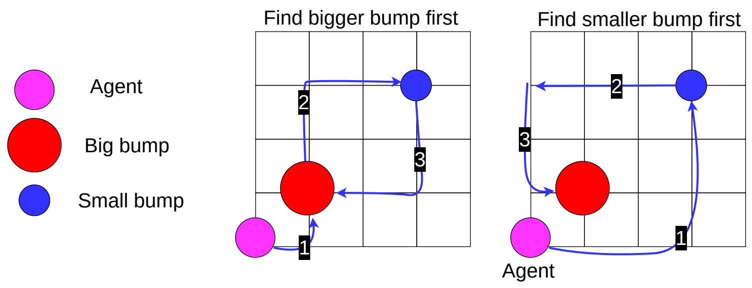


Figure 10: The policy learned by COSIL in `Bumps-2D` [6, 32] acts differently depending on which bump is found first. Numbers denote the order in the action sequences.

F.3 Minigrid-Memory

COSIL learns a policy that will first go to the left to see the object in a small room. It then memorizes the object and goes to the right to find the matching object.

F.4 Car-Flag

As shown in Figure 11, COSIL learns a policy that drives the car to the vicinity of the blue flag to observe the side of the green flag. It then memorizes the side before going to the green flag.

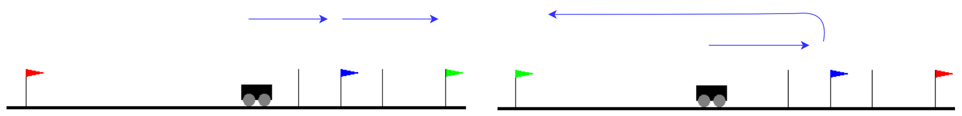


Figure 11: The policy learned by COSIL in `Car-Flag` [32] first goes to the blue flag to check for the side of the green flag before navigating to it. The green flag can be on the left or on the right.

F.5 LunarLander-P and -V

The policy behaves similarly to the state expert but is often less aggressive in driving the agent down on the ground. The partial observability might explain the behavior.

F.6 Block-Picking

COSIL learns a policy that consistently picks one block on the left or on the right. If the block is movable, then the agent will pick that block. If it cannot move that block, it will attempt to pick the other block. During the deployment to the real robot, where making a block immovable is difficult, whenever the agent observes that a block is movable (even if that block is supposed to be immovable), it will keep trying to pick that block. It will eventually succeed due to the powerful gripper that we use. To fix this issue, during deployment, when the agent tries to pick a block that is supposed to be immovable, we disable the $(\delta_x, \delta_y, \delta_g)$ component of the action of the gripper for 20 timesteps (Figure 12). The agent then cannot move the block, and then it "thinks" that block cannot be moved and will move to the other block to pick (after 20 timesteps being partially blocked).

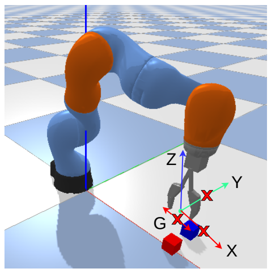


Figure 12: In `Block-Picking`, movements along certain axes are temporarily disabled during policy deployment to the real robot when the gripper picks a block that is supposed to be immovable. Disabling is necessary; otherwise, the powerful gripper will easily move the supposedly immovable block regardless of any taping or gluing.

G Implementation Details

G.1 Non-searched Hyperparameters

Table 2 shows the hyperparameters over which we do not perform a grid search. We do not perform the search over any hyperparameters of VRM because it is very computationally expensive to run. All methods use Adam optimizer [47] with the default decay rates $(\beta_1, \beta_2) = (0.9, 0.999)$.

Method	Values of Hyperparameters
COSIL	`Replay Buffer`: batch-size = 32, buffer-size = 1e6 `Target Network Update`: $\tau = 0.005$ `RNN`: GRU(128)
DPFRL [2]	`A2C`: num-processes = 16, num-steps = maximum episode length, entropy-coef = 0.01, value-loss-coef = 0.5 `Particle Filter`: num-particles = 30, particle-aggregation = MGF `RNN`: GRU(128)
PPO [38]	`PPO`:num-processes = 16, num-step = maximum episode length, $\lambda_{\text{GAE}} = 0.95$ `RNN`: GRU(128)
SAC [34]	`Replay Buffer`: batch-size = 32, buffer-size = 1e6 `Target Network Update`: $\tau = 0.005$ `RNN`: GRU(128)
TD3 [34]	`Replay Buffer`: batch-size = 32, buffer-size = 1e6 `Exploration`: exploration-noise = 0.1, target-noise = 0.2, target-noise-clip = 0.5 `Target Network Update`: $\tau = 0.005$ `RNN`: GRU(128)
VRM [3]	Default hyperparameters from the paper (see Table 1 in [3])
ADV-On [25]	`PPO`: $\lambda_{\text{GAE}} = 1.0$, value loss coefficient = 0.5, discount factor $\gamma = 0.99$ `RNN`: LSTM(128)
BC2SAC	`Replay Buffer`: batch-size = 32, buffer-size = 1e6 `Target Network Update`: $\tau = 0.005$ `RNN`: GRU(128)
BC+SAC	`Replay Buffer`: batch-size = 32, buffer-size = 1e6 `Target Network Update`: $\tau = 0.005$ `RNN`: GRU(128)
Markovian SAC [34]	`Replay Buffer`: batch-size = 32, buffer-size = 1e6 `Target Network Update`: $\tau = 0.005$
DAgger [18]	`Replay Buffer`: batch-size = 32, buffer-size = 1e6 `RNN`: GRU(128)

Table 2: Non-searched hyperparameters of agents.

G.2 Grid-Searched Hyperparameters

For all methods, we perform a grid search over the learning rates. Moreover, we also perform a grid search for certain hyperparameters exclusively for each method.

- ADV-On: $\alpha \in \{4.0, 8.0, 16.0, 32.0\}$ (see Equation (8) for the role of α)
- BC2SAC: The percentage of total timesteps trained with behavioral cloning p_{BC} in $\{10\%, 20\%, \dots, 90\%\}$.
- BC+SAC: The weight for BC loss $w_{\text{BC}} \in \{0.1, 0.2, \dots, 0.9\}$
- COSIL: $\bar{D} \in \{0, 0.1, \dots, 1.0\}$

G.3 Network Structures

The network architecture of our agent is shown in Figure 13, which we developed using the codebase in [34]. The encoders (observation and action) can either be MLP- or CNN-based, depending on the domains (see Table 3). RNNs are 1-layered GRU or LSTM networks of 128 hidden units. The observation-action encoder combines an observation and an action encoder inside.

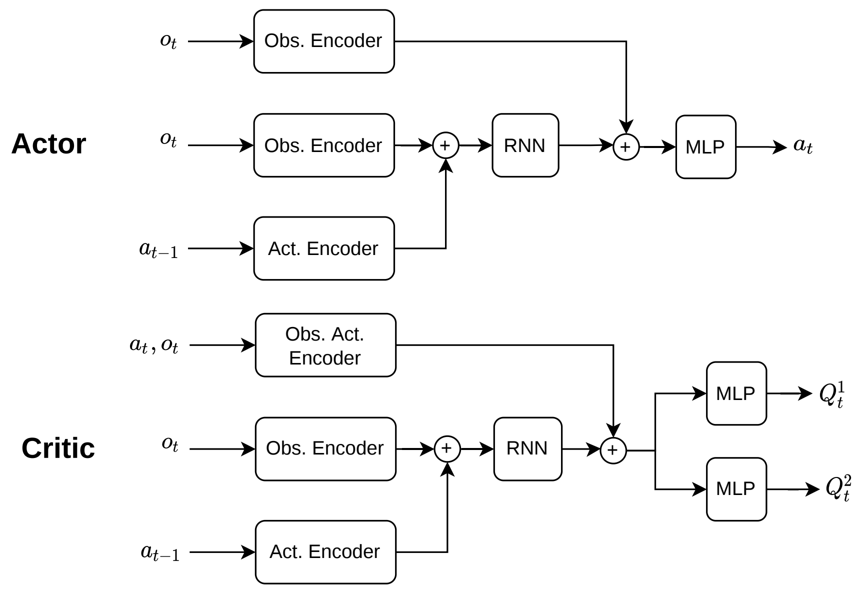


Figure 13: The network structure of COSIL.

Domain	**Obs. Encoder**	**Act. Encoder**
`Bumps-1D`	**FCR**(dim(Ω), 32)	**FCR**($\lvert\mathcal{A}\rvert$, 8)
`Bumps-2D`	**FCR**(dim(Ω), 32)	**FCR**($\lvert\mathcal{A}\rvert$, 8)
`Minigrid-Memory` `Minigrid-Crossing`	**CNNR**(3, 16, 2, 2) + **CNNR**(32, 64, 4, 2) + **MaxPool**(2, 2) + **CNNR**(16, 32, 2, 2) + **CNNR**(32, 64, 2, 2)	**FCR**($\lvert\mathcal{A}\rvert$, 8)
`Car-Flag`	**FCR**(dim(Ω), 32)	**FCR**(dim($\mathcal{A}$), 8)
`LunarLander-P, -V`	**FCR**(dim(Ω), 32)	**FCR**(dim($\mathcal{A}$), 8)
`Block-Picking`	**CNNR**(3, 32, 8, 4) + **CNNR**(32, 64, 4, 2) + **CNNR**(64, 64, 3, 2) + **FCR**(3136, 128)	**FCR**(dim($\mathcal{A}$), 8)

Table 3: Encoders used for each domain. **FCR** denotes fully connected layers with (input dimension, output dimension) followed by ReLU. **CNNR** denotes convolution neural networks with (input, output, kernel, stride) followed by ReLU.

H Additional Experimental Results

H.1 Performance in LunarLander-V

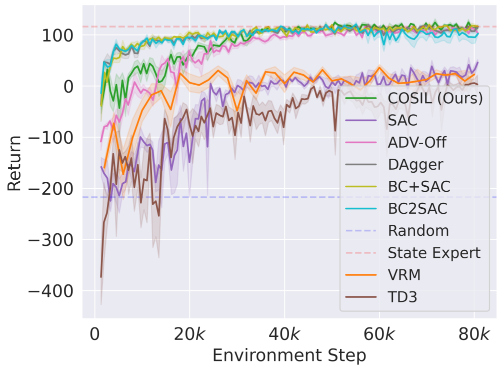
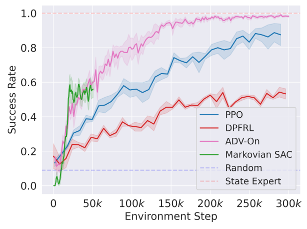


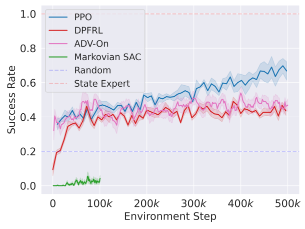
Figure 14: Performance of all methods on `LunarLander-V`.

H.2 Performance of On-Policy Baselines

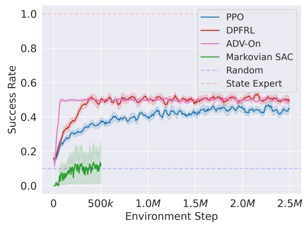
In Figure [15], we report the performance of a recurrent version of PPO, ADV-On (based on PPO), Markovian SAC, and DPFRL for all domains. We only report ADV-On for domains with discrete action spaces as supported by the official codebase at `https://github.com/allenai/advisor`.



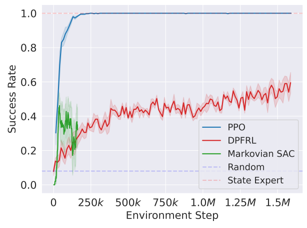
(a) `Bumps-1D`



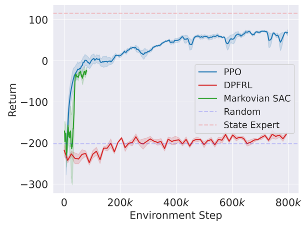
(b) `Bumps-2D`



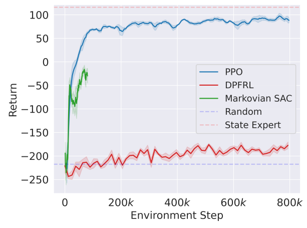
(c) `Minigrid-Memory`



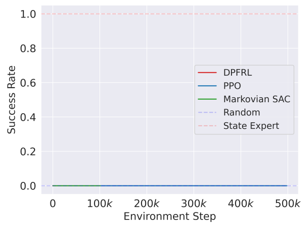
(d) `Car-Flag`



(e) `LunarLander-P`



(f) `LunarLander-V`



(g) `Block-Picking`

Figure 15: Performances of additional baselines (recurrent PPO [9], ADV-On [25], Markovian SAC [34], and DPFRL [2].