

# Leveraging Fully Observable Policies for Learning under Partial Observability

Hai Nguyen, Andrea Baisero, Dian Wang, Christopher Amato, Robert Platt  
Khoury College of Computer Sciences  
Northeastern University, Boston, MA, United States  
nguyen.hai1@northeastern.edu

**Abstract:** Reinforcement learning in partially observable domains is challenging due to the lack of observable state information. Thankfully, learning offline in a simulator with such state information is often possible. In particular, we propose a method for partially observable reinforcement learning that uses a fully observable policy (which we call a *state expert*) during offline training to improve online performance. Based on Soft Actor-Critic (SAC), our agent balances performing actions similar to the state expert and getting high returns under partial observability. Our approach can leverage the fully-observable policy for exploration and parts of the domain that are fully observable while still being able to learn under partial observability. On six robotics domains, our method outperforms pure imitation, pure reinforcement learning, the sequential or parallel combination of both types, and a recent state-of-the-art method in the same setting. A successful policy transfer to a physical robot in a manipulation task from pixels shows our approach’s practicality in learning interesting policies under partial observability.

**Keywords:** Partial Observability, Imitation Learning, Fully Observable Experts

## 1 Introduction

Many real-world robotics control problems feature some degree of partial observability, but policy learning in this setting remains a significant challenge in robotics [1]. While there are many reinforcement learning methods that address partial observability in theory [2, 3, 4, 5], their performance remains questionable in practice in interesting partially observable (PO) domains which require long-term memorization and active information gathering [6, 7]. In contrast, the setting of fully observable (FO) control has featured the success of many powerful reinforcement learning (RL) algorithms (e.g., [8, 9, 10, 11]). Unfortunately, full observability only holds for a small portion of realistic robotics problems.

In this work, we attempt to leverage good *fully observable* policies (*state experts*) available only during offline training to help train PO policies that can execute online. We rely on the setting of *offline training and online execution*, a successful RL framework where an agent can use “privileged” information such as the state [12, 13, 14, 15] or the belief about the state [6] during offline training, e.g., from simulators, to efficiently learn PO policies that are later can be deployed without the access to the privileged information anymore. In this work, the privileged information is not just the state itself but also the state expert. Our setting can be illustrated in a navigation task (Figure 1), which requires an agent to navigate to an unknown goal object on the right, identifiable by an object on the left side. While the optimal behavior under partial observability is to first navigate leftwards to identify the goal object, the state expert is able to move to the goal object directly. Despite being sup-optimal from the PO perspective, the state expert can provide experience during training leading to the goal object, which is potentially useful for both exploration and as a part of the policy needed in the PO case after the goal object is identified.

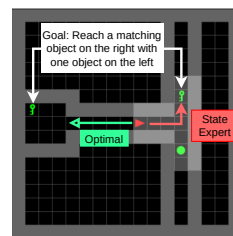


Figure 1: To reach the correct goal object, a state expert takes the red path directly, while a partially observable agent must first take the green path to identify the correct goal object, then take the red path.

In this work, we propose to perform a form of *cross-observability soft imitation learning* (COSIL), *i.e.*, *softly* projecting the behavior of the FO agent into a PO agent which still tries to maximize its own performance. FO experts are generally easier to obtain than PO ones, which often require more impractical computations, *e.g.*, some sufficient statistics of the entire history like the belief states, which requires the true environment dynamics [16]. The resulting COSIL agent balances behaviors that imitate the FO agent with behaviors that are necessary for PO optimality. Experimental results on six robotics domains show that our proposed COSIL method performs significantly better than pure imitation learning, pure reinforcement learning, the sequential or parallel combination of the two types of learning, and other state-of-the-art methods in the same setting. Moreover, a policy with pixel observations learned in simulation is transferred successfully to a physical robot, showing COSIL’s ability to learn policies that can handle the complexities of the real world. Additional details can be seen at our project website <https://sites.google.com/view/cosil-cor122>.

## 2 Related Work

Imitation learning methods such as behavioral cloning (BC) [17] and DAGger [18] perform policy learning by minimizing a supervised loss between the expert’s actions and the agent’s. These agents are trained on a dataset of encountered states and expert actions and are utilized successfully in tasks like autonomous flight [19, 20] or self-driving [21]. Several methods combine reinforcement learning and imitation learning (often from limited expert demonstrations), *e.g.*, DQfD [22], DDPGfD [23], and GAIL [24]. However, these methods are designed for Markov decision processes (MDP), *i.e.*, both the expert and the student can observe the environment state. Closest to our work are ADVISOR [25] and Asymmetric DAGger (A2D) [15], which leverage FO experts *during training* for PO tasks. A2D operates in a setting requiring a differentiable and modifiable state expert, jointly trained from scratch (using the environment states) with the student agent, while *we assume a given, fixed state expert*. In the same setting as ours, ADVISOR adaptively changes the weighting per state between the imitation loss and the reinforcement learning loss (see the appendix for a brief description of ADVISOR). However, it also requires training an additional PO actor to mimic the state expert to compute the weighting. Therefore, both ADVISOR and A2D must wait until the additional actor (ADVISOR) and the state expert (A2D) are properly trained before utilizing them to better train the main agent. In contrast, our method can utilize the state expert immediately.

## 3 Background

In this section, we introduce the background topics required to understand our contributions, *i.e.*, partially observable Markov decision processes, and the Soft Actor-Critic algorithm.

### 3.1 Partially Observable Markov Decision Processes

A partially observable Markov decision process (POMDP) [26] is defined by a tuple  $(\mathcal{S}, \mathcal{A}, \Omega, p_0, T, R, O, \gamma)$ , where  $\mathcal{S}$ ,  $\mathcal{A}$ , and  $\Omega$  are the state space, the action space, and the observation space, respectively. The initial state is sampled according to the starting state distribution  $p_0(s)$ , and the next states are sampled according to the stochastic transition function  $\Pr(s' | s, a) = T(s, a, s')$ . The agent does not observe the states but rather receives observations sampled from the observation function  $\Pr(o | s, a) = O(s, a, o)$ . In order to act optimally, a PO agent must, in general, condition its actions on the entire action-observation history  $h_t = (a_{<t}, o_{\leq t})$ , *i.e.*, all the observable information it has seen so far [27]. Denoting the space of histories as  $\mathcal{H}$ , the goal is to find a *history* policy  $\pi: \mathcal{H} \rightarrow \Delta(\mathcal{A})$  which maximizes the expected  $\gamma$ -discounted return  $J_\pi = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$ .

Although our work focuses on PO control, it also involves FO agents modeled as a *state* policy  $\mu: \mathcal{S} \rightarrow \Delta(\mathcal{A})$ . To differentiate them clearly, we denote history policies as  $\pi$  and state policies as  $\mu$ .

### 3.2 Soft Actor-Critic

Soft Actor-Critic (SAC) [10, 28] addresses the FO maximum-entropy control problem, *i.e.*, the problem of finding a policy  $\mu$  which solves a given FO control problem while maintaining a high action-entropy when possible. SAC does this by extending the standard RL objective with a weighted

entropy term,

$$J_\mu = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t) + \alpha H(\mu(s_t))) \right], \quad (1)$$

where  $H(\mu(s)) = \mathbb{E}_{a \sim \mu(s)} [-\log \mu(a | s)]$  is the entropy of policy  $\mu$  at the given state  $s$ , and  $\alpha > 0$  is a trade-off coefficient which determines the relative importance between the RL objective and the entropy-maximization objective. The entropy term can be interpreted as a supplementary pseudo-reward given to the agent, which is high for high-entropy policies and low for low-entropy policies. Therefore, the agent will seek not only to maximize the entropy of the policy in the visited states but also to visit states associated with a high-entropy policy.

In practice, SAC is an off-policy learning algorithm that employs a replay buffer containing past transitions  $\mathcal{D} = \{(s, a, r, s')_i\}_{i=0}^N$ . SAC trains a parametric policy model  $\mu_\theta: \mathcal{S} \rightarrow \Delta\mathcal{A}$ , and a parametric value model  $Q_\phi: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ . The policy model is trained by maximizing

$$J_\mu(\theta) = \mathbb{E}_{s \sim \mathcal{D}, a \sim \mu_\theta} [Q_\phi(s, a) - \alpha \log \mu_\theta(a | s)], \quad (2)$$

while the value model is trained to minimize

$$J_Q(\phi) = \frac{1}{2} \mathbb{E}_{s, a \sim \mathcal{D}} \left[ (R(s, a) + \gamma \mathbb{E}_{s' | s, a} [V(s')] - Q_\phi(s, a))^2 \right], \quad (3)$$

$$V(s) = \mathbb{E}_{a \sim \mu_\theta(s)} [Q_{\bar{\phi}}(s, a) - \alpha \log \mu_\theta(a | s)], \quad (4)$$

where  $Q_{\bar{\phi}}$  is a frozen target model that is updated at a slower pace than  $Q_\phi$  to improve stability.

Hyperparameter  $\alpha$  plays a central role in SAC, determining how much high-entropy states are preferred to pure rewards. Choosing a reasonable  $\alpha$  can be difficult since it is not directly interpretable, and a good value depends dynamically on the current policy’s expected returns and entropy. Haarnoja et al. [28] proposed to automatically adjust  $\alpha$  by minimizing its own objective,

$$J_\alpha(\alpha) = \alpha \mathbb{E}_{s \sim \mathcal{D}, a \sim \mu_\theta(s)} [-\log \mu_\theta(s)] - \alpha \bar{H}, \quad (5)$$

where  $\bar{H}$  is a given target entropy. In practice, Equation (5) modulates  $\alpha$  such that it is increased if the current policy entropy is lower than the target entropy and vice versa. In contrast to choosing a value of  $\alpha$ , choosing a value of  $\bar{H}$  is much simpler since it is broadly interpretable as the logarithm of the number of actions that we want the max-entropy policy to consider in an average state.

### 3.3 Soft Actor-Critic for Partially Observable Control

Although primarily designed for FO control problems and state policies  $\mu$ , SAC can be easily adapted to handle PO control problems and history policies  $\pi$ , like most (if not all) model-free learning algorithms. Two main changes need to take place for SAC: First, all appearances of a state  $s$  in the equations and models of SAC must be replaced with a respective history  $h$ . This also implies the use of a recurrent neural network component in the overall architecture of policy and value models, *e.g.*, an LSTM [29] or a GRU [30]. Second, the replay buffer must be structured in order to contain and extract (truncated) episodes rather than individual transitions.

## 4 Cross-Observability Soft Imitation Learning

SAC is based on the premise of max-entropy control and is designed to find policies that solve the control task while pertaining to high entropy. This is achieved by extending the standard RL objective with an entropy component which not only pushes the policy model to be more stochastic for any given state but also acts as a pseudo-reward that pushes the policy to visit future states where the policy can be more stochastic. Inspired by this dual effect, we aim to employ a similar technique to exploit FO expert knowledge to train a PO agent.

Consider an offline training scenario where an FO expert  $\mu$  is available, *e.g.*, obtained via a planning procedure, a pre-trained model, or a model trained jointly with the PO agent. We propose formulating a pseudo-reward for the PO agent  $\pi$  based on the expected similarity with the FO agent  $\mu$ , expressed as the following *cross-observability soft imitation learning* (COSIL) objective,

$$J_\pi = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t) - \alpha D(\mu(s_t), \pi(h_t))) \right], \quad (6)$$

where  $D(\mu(s), \pi(h))$  is some divergence measure between the action-distributions of  $\mu$  and  $\pi$ , *e.g.*, the *KL* divergence  $D(\mu(s), \pi(h)) = \mathbb{E}_{a \sim \mu(s)} [\log \mu(a | s) - \log \pi(a | h)]$ , or the *total variation* divergence  $D(\mu(s), \pi(h)) = \max_a |\mu(a|s) - \pi(a|h)|$ , among other options.

Minimizing the divergence  $D(\mu(s), \pi(h))$  can be interpreted as a form of *cross-observability* imitation learning, which tries to project FO behaviors as PO behaviors. On its own, such a form of imitation learning is known to be sub-optimal, *e.g.*, it is not generally possible to replicate the behavior of  $\mu$  with  $\pi$ . Further, FO behaviors are known to be, in general, sub-optimal for PO control, *e.g.*, PO agents might need to engage in information-gathering behaviors which an FO agent would not exhibit. An agent  $\pi$  which strictly minimizes the divergence  $D$  would incur an *optimality gap* between its resulting performance and the true optimal performance (see the appendix).

Despite these issues, we argue that applying a trade-off between the pure RL objective and the imitation learning objective in a *soft* fashion, *i.e.*, using the divergence  $D$  as a pseudo-reward can be quite beneficial. We are directly inspired by SAC, which applies a similar trade-off between the pure RL and the max-entropy objectives to establish itself as a state-of-the-art model-free RL algorithm. In SAC too, the max-entropy objective alone is insufficient and inadequate when considered alone and only becomes beneficial when combined with task rewards as a pseudo-reward. Additionally, the FO expert  $\mu$  is likely to be a good source of exploratory behavior even for the PO task. The actions chosen by  $\mu$  are undoubtedly related to the overall task, and it is likely beneficial for the PO agent to explore them thoroughly. Second, there are many PO tasks where the agent can and/or must reach low state uncertainty to solve the tasks. Under such low state uncertainty situations, the optimal PO and FO behaviors overlap strongly, if not perfectly (as long as the state uncertainty is maintained low enough). In such situations, the optimality gap becomes low, and the cross-observability imitation learning task becomes beneficial. Finally, because the soft-imitation task is encoded as a pseudo-reward, the PO agent is generally pushed to achieve situations where there is low state uncertainty, which tends to be very beneficial for PO agents.

Similar to Equation (5), we formulate a minimization objective for  $\alpha$ ,

$$J_\alpha(\alpha) = \alpha \bar{D} - \alpha \mathbb{E}_{h, s \sim \mathcal{D}} [D(\mu(s), \pi(h))] , \quad (7)$$

which modulates  $\alpha$  dynamically in order to maintain an expected divergence of  $\bar{D}$ . Like the target entropy  $\bar{H}$  in SAC,  $\bar{D}$  is an important hyperparameter that indicates how different the policies  $\pi$  and  $\mu$  are allowed to be on average. The semantics of  $\bar{D}$  depend on the choice of divergence function  $D(\pi(h), \mu(s))$ , and good values of  $\bar{D}$  are likely to be domain-dependent. In practice, we perform a grid search to find the best value. Please see the appendix for the detailed algorithm.

## 5 Experiments

We perform experiments on a diverse set of robot and navigation domains with discrete (**D**) and continuous action spaces (**C**), proprioceptive and pixel observations. We focus on the setting where the state experts are given *during training*, *i.e.*, the setting of ADVISOR [25].

### 5.1 Domains

Below, we only briefly describe the domains and the behavior of the state experts; see the appendix for more details.

**(D) Bumps-1D.** [6] Two movable homogeneous bumps rest on a table, with a robotics finger moving horizontally above them (Figure 2a). When an episode starts, the positions of all entities are randomized, but the left-right order between the bumps is maintained. The agent must push the rightmost bump (blue) to the right without disturbing the left bump (red). There are four action combinations: move left or right, each with a compliant or stiff finger. The agent does not know which bump is rightmost – it must infer using the history of the finger’s deflected angles and positions while touching both bumps with a compliant finger before pushing right the blue bump with a stiff finger.

**(D) Bumps-2D.** [6] The same finger is now constrained to move in a plane above two bumps of different sizes (Figure 2b). It must reach the bigger bump (red) and stay there to get the only non-zero reward. However, it is unaware of which bump is bigger and, therefore, must make contact with



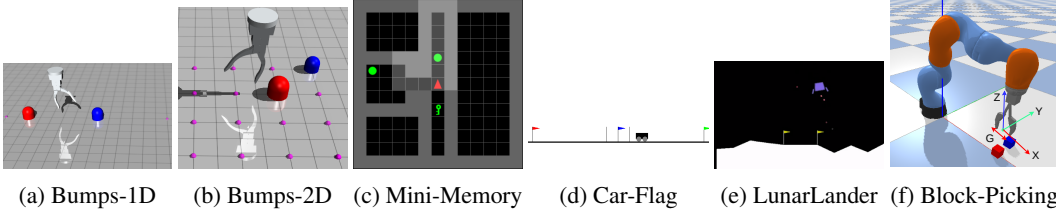


Figure 2: Our experimental domains. The first three domains have discrete actions and the last three feature continuous actions. Block-Picking is the only domain with pixel observations.

both bumps at least once, inferring the bumps’ relative sizes from the angular displacement of the finger. The state expert, however, knows the position of the bigger bump and can go there directly.

**(D) Minigrid-Memory.** [31] An agent (red triangle) must go to a matching object located on the right side with an object (a ball or a key) located in a small room on the left side (Figure 2c). The grid layout, the matching object type, and the agent’s starting position are randomized per episode. The agent might start in a cell where it cannot see the object; therefore, an optimal agent must first gather information by going/turning left into the small room to see the object. We chose the biggest version of this environment [31] with a size of 17.

**(C) Car-Flag.** [32] A car must reach the green flag, which can be on either the left or right side. The agent normally can only observe the car’s position and velocity, but if it is near the blue flag, it can also observe the green flag’s side (left/right). The optimal policy will be navigating to the green flag *only* after visiting the blue flag to find the right side. In contrast, the optimal state expert will always go left or right towards the green flag, whose position is known to the expert.

**(C) LunarLander-P, LunarLander-V.** We consider two versions of the classic LunarLander environment [33] where the agent only observes subsets of the full state (see the appendix for more details). During training, the expert, on the other hand, can observe the full state. Masking parts of the state to turn MDPs into POMDPs is common in previous work [34, 3, 35, 5]. However, these domains do not explicitly require active information gathering because the missing information (*e.g.*, velocities) can be inferred only by memorizing short observation histories (*e.g.*, positions).

**(C) Block-Picking.** Two homogeneous blocks rest on a table. The red block is fixed to the table and cannot be moved. In contrast, the blue one is movable, and the agent will accomplish the task if it picks the blue block. The agent takes in a top-down colorless depth image; therefore, it must gather information by trying to move both blocks to check for movability. The state expert knows the poses of the movable block during training and can always perform a successful pick. This task is particularly challenging if learning from scratch because the exploration is hard, the reward is sparse, the action space is continuous, and it can potentially be long-horizon.

## 5.2 Baselines

We compare COSIL against a diverse set of pure imitation learning agents, pure reinforcement learning agents (general and specialized for POMDPs, on-policy, and off-policy), and the combination of both types in several ways. If not described explicitly, all of these agents are *memory-based*.

**ADVISOR.** ADV-On is the original on-policy ADVISOR [25] that is built off PPO [9]. For a fair comparison with COSIL (off-policy), we implemented an off-policy version of ADVISOR (ADV-Off) using SAC for continuous action spaces [10, 28] and discrete action spaces [36].

**DAGGER** [18] is chosen to represent imitation learning methods instead of a naive behavioral cloning agent, because DAGGER often performs better.

**TD3** refers to a recurrent version of TD3 [37] for domains with continuous action spaces, implemented in [34].

**SAC** refers to two recurrent versions of SAC for continuous actions [10, 28] and for discrete actions [36], followed the implementation by [34].

**BC2SAC** is pre-trained with behavioral cloning for  $p_{BC}\%$  of the total training timesteps before being trained with SAC.

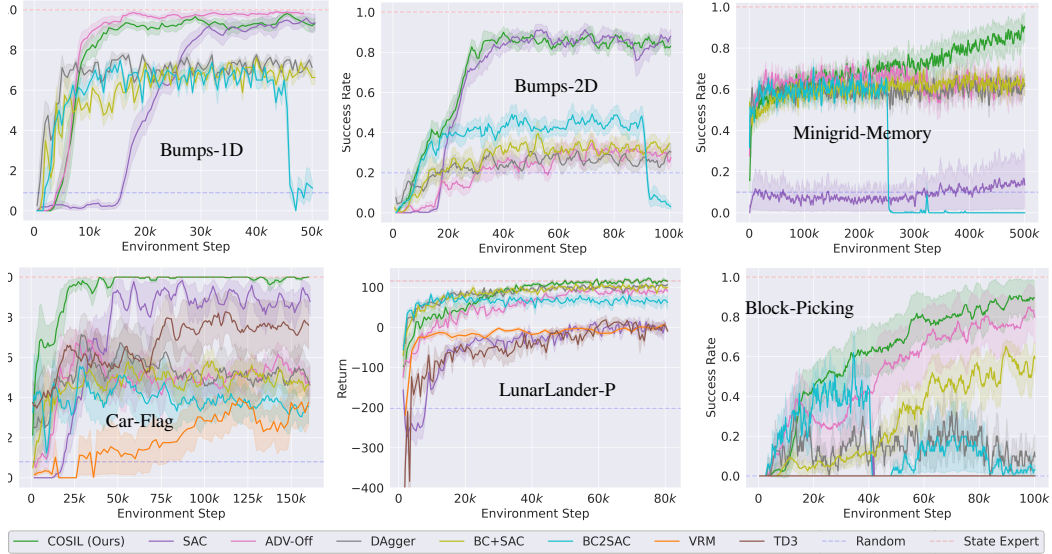


Figure 3: Learning curves of all methods (4 seeds with the shaded areas denoting 1 standard deviation). Note that all agents (except Random and State Expert) are memory-based agents.

**BC+SAC** is trained using a weighted combination of behavioral cloning loss  $\mathcal{L}_{BC}$  and RL losses  $\mathcal{L}_{RL}$ , *i.e.*, the agent is trained by optimizing  $\mathcal{L} = w_{BC}\mathcal{L}_{BC} + (1 - w_{BC})\mathcal{L}_{RL}$ .

**VRM** is a state-of-the-art off-policy model-based agent [3] for POMDPs with continuous action spaces by combining a recurrent variational dynamic model and a SAC agent.

**Random, State Expert.** While a random agent takes actions randomly, a state expert is either trained with full state access or is hand-coded using the domain knowledge.

**Additional Baselines** include recurrent **PPO** [9] implemented in [38], an on-policy POMDP specialized method - **DPFRL** [2], and a non-recurrent SAC (**Markovian SAC** [34]) that uses observations instead of histories. We report the performance of these baselines in the appendix.

### 5.3 Experiments

**Metrics.** Depending on each domain, we record either the success rates or the returns of evaluation agents averaged over ten episodes. For SAC-based agents, we turn off exploration during evaluation. The reported results are averages over four seeds with shaded areas denoting one standard deviation with optional smoothing if needed. For Random and State Expert, their performances are averaged over 100 episodes and visualized as horizontal lines.

**Results.** The performance of agents is shown in Figure 3 (the performance in LunarLander-V is similar to LunarLander-P and is reported in the appendix). Overall, COSIL (green) is comparable to ADV-Off in Bumps-1D, LunarLander-P, Block-Picking but considerably better in the remaining three domains, which we argue have larger optimality gaps. DAgger performs quite well in LunarLander-P, suggesting that this domain’s optimality gap is small. This is expected for this domain because the partially observable angular velocities of the vehicle can be inferred from a few recent fully observable angles. Our agent can also perform well in this domain, signifying the ability to work in domains with a small optimality gap. Other baselines that do not use the state expert struggle to learn efficiently, especially in Block-Picking, where random exploration will have a minuscule chance of picking the correct object. This explains the complete failures of naive baselines such as SAC or TD3, and highlights the advantage of exploiting the state expert for better exploration. BC2SAC performs poorly across all domains. This way, reinforcement learning often unlearns imitation learning policies when optimizing RL losses. The poor performance is illustrated by sudden drops whenever reinforcement learning starts. BC+SAC does not seem to perform well either, often not significantly better than DAgger. A fixed static combination like BC+SAC can be sub-optimal because the optimal PO policy can sometimes act vastly differently from the FO expert.

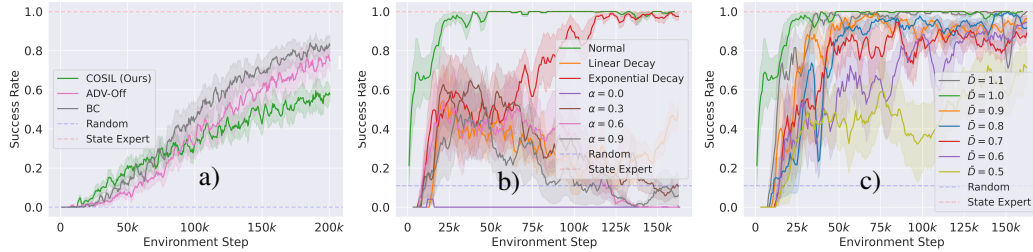


Figure 4: Additional results: a) COSIL in MiniGrid-Crossing; Performance of COSIL in Car-Flag when: b) linearly/exponentially decaying  $\alpha$ , varying  $\alpha \in \{0.0, 0.3, 0.6, 0.9\}$ ; c) varying  $\bar{D} \in \{0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1\}$ .

For instance, the PO agent often must act to gather more information about the state. In contrast, the state expert hardly has to do that, given its full knowledge of the state. Therefore, it is difficult for BC+SAC to behave optimally by fixing the weights between BC and RL loss. In fact, this observation has indeed motivated ADVISOR, which adaptively changes the weights between the BC loss and the PPO [9] loss for each state (see the appendix for more details).

**Training & Hyperparameter Tuning.** All agents are trained using complete episodes (truncated if too long). On-policy agents (*i.e.*, DPFRL, PPO, and ADV-On) use 5-10x more samples than other off-policy agents to train. We perform a grid search over relevant hyperparameters of each method (see the appendix for more details), then select the best combinations to report.

#### 5.4 Additional Experiments

**Performance in Domains with Small Optimality Gap.** When the optimality gap is small, *e.g.*, when the state expert is nearly optimal under both full and partial observability, the best learning strategy is to purely mimic the state expert. We demonstrate this point in a variant of MiniGrid-Crossing [31] with a grid size of 25 and 10 crossings (see the appendix for details about the domain). This domain has a very small optimality gap: the paths that lead to a goal cell of an optimal PO agent (which only observes a local area around it) and a state agent (which observes the whole grid world) are quite similar. Figure 4a shows that both COSIL (with  $\bar{D} = 0$ ) and ADV-Off are outperformed by behavioral cloning as expected. In the case of COSIL, by setting  $\bar{D} = 0$ ,  $\alpha$  will be increased to penalize the agent heavily if it behaves differently from the state expert (see Equation (6)), gradually morphing COSIL into behaving like BC.

**Not adapting  $\alpha$  over time.** In Figure 4b, we illustrate the performance of COSIL in Car-Flag without adapting  $\alpha$  to reach the target divergence  $\bar{D}$ . We investigated reducing  $\alpha : 1 \rightarrow 0$  linearly/exponentially over time or keeping it fixed at different values in  $\{0.0, 0.3, 0.6, 0.9\}$ . The figure shows that adapting  $\alpha$  is superior to other ways of modulating  $\alpha$  in this domain. However, similar to SAC, there is no hard constraint that  $\alpha$  must be always adapted.

**Performance with varying  $\bar{D}$ .** We investigate the sensitivity of COSIL in Car-Flag when varying  $\bar{D}$  from 1.1 to 0.5 with a step of 0.1. Figure 4c shows COSIL is stable with half of the tested range.

#### 5.5 Real Robot Experiments

We transfer the policies learned in simulation in Block-Picking to a Universal Robot UR5 arm mounted with a Robotiq 2F-85 gripper (Figure 5). We use two cameras: one Occipital Structure Sensor (Cam 1) and one Microsoft Azure Kinect camera (Cam 2), and combine their point clouds to create a depth image by using an orthographic projection at the gripper’s position. To better transfer, during training, we add uniform noise to the positions and the yaw angle of the two blocks and add Perlin noise [39, 40] to the simulated depth images. We pick the best policies in the simulation of ADV-Off and COSIL, roll out ten episodes for each, and count the number of successful picks.

During the deployment, when the agent manipulates the supposedly immovable block, we temporarily disabled the movements along  $x, y, g$  axes (see coordinates in Figure 2f) so that the agent will manipulate the other block after a while trying to move the earlier block without observing any

movements. The disabling is necessary because when the agent observes that the immovable block is *movable*, it will persistently manipulate that block. The temporal blocking allows us to avoid heavily gluing the immovable block to the table. However, it also leads to cases that the agents never see during training (the agents desire to move the gripper, but nothing moves due to the blocking), but both COSIL and ADV-Off seem to generalize well.

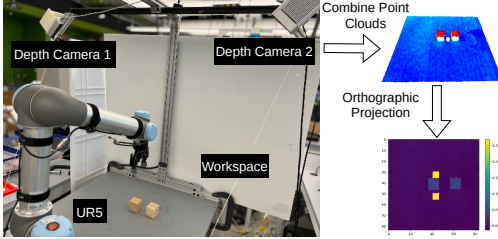


Figure 5: Experimental robot setup.

Method	Simulation	Real-world
ADV-Off	9/10	6/10
COSIL	9/10	8/10

Table 1: Result of transferring policies learned in simulation to the real world in Block-Picking.

Table 1 shows the performance comparison of pick successes between COSIL and ADV-Off. The results show that our COSIL policy learned in simulation can actually perform well in the real world, in a PO task that would be very challenging to learn without the aid from a state expert during training. Empirically, our policies are more robust to rotational and translational misalignment of the two blocks (see videos on our project website), leading to more successful picks. However, similar success rates are expected with a bigger sample size, given their similar performance in simulation.

## 6 Limitations

COSIL can bias exploration negatively if the state expert is heavily sub-optimal under partial observability, being equal to giving very poor demonstrations. In this case, it is doubtful that any method can utilize FO experts to aid in learning optimal PO policies. To illustrate, we modify Bumps-2D such that staying on top of the bigger bump without visiting the smaller bump in the past will be penalized heavily (-100). Previously, such behavior of the state expert did not incur any penalty. Moreover, to maintain Markovian rewards (*e.g.*, the reward only depends on the last state and action), we add a flag indicating whether the smaller bump was visited to the observation. Figure 6 shows that both COSIL and ADV-Off are outperformed by SAC, with ADV-Off failing to learn completely. For COSIL, setting a high value of  $\bar{D}$  might alleviate the situation as COSIL will become closer to a normal SAC agent.

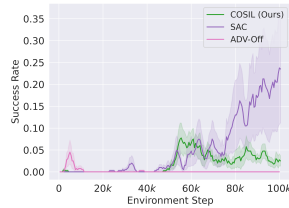


Figure 6: An example when COSIL can be less performant than SAC.

Another limitation is the requirement of constant access to expert actions at every state during offline training, which might be impractical. One potential fix is using our method with a variant of SAC that learns from limited demonstrations, such as one proposed by Liu et al. [41]. Another possible direction for addressing this issue is to use a method proposed by Gangwani et al. [35] with limited demonstration trajectories given by a state expert instead of a POMDP expert.

## 7 Conclusion

This paper tackles POMDPs in robotics from an unconventional angle: leveraging fully observable policies during offline training to better train partially observable policies that later can be deployed online. We introduce *cross-observability soft imitation learning*, which balances achieving high PO performance with acting similarly to the FO expert. With COSIL, the PO agent can exploit the FO expert for targeted exploration and information-gathering during offline training. COSIL outperforms the state-of-the-art method in the same setting in several robotics domains. COSIL also performs better than other conventional approaches and can learn policies that are successfully transferred to a physical robot.

## Acknowledgments

This material is supported by the Army Research Office under award number W911NF20-1-0265; the U.S. Office of Naval Research under award number N00014-19-1-2131; NSF grants 1816382, 1830425, 1724257, and 1724191.

## References

- [1] H. Kurniawati. Partially observable markov decision processes and robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 5(1):253–277, 2022.
- [2] X. Ma, P. Karkus, D. Hsu, W. S. Lee, and N. Ye. Discriminative particle filter reinforcement learning for complex partial observations. In *International Conference on Learning Representations*, 2019.
- [3] D. Han, K. Doya, and J. Tani. Variational recurrent models for solving partially observable control tasks. In *International Conference on Learning Representations*, 2019.
- [4] M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson. Deep variational reinforcement learning for pomdps. In *International Conference on Machine Learning*, pages 2117–2126. PMLR, 2018.
- [5] L. Meng, R. Gorbet, and D. Kulić. Memory-based deep reinforcement learning for pomdps. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5619–5626. IEEE, 2021.
- [6] H. Nguyen, B. Daley, X. Song, C. Amato, and R. Platt. Belief-grounded networks for accelerated robot learning under partial observability. In *Conference on Robot Learning*, pages 1640–1653. PMLR, 2021.
- [7] H. Nguyen, Z. Yang, A. Baisero, X. Ma, R. Platt, and C. Amato. Hierarchical reinforcement learning under mixed observability. *arXiv preprint arXiv:2204.00898*, 2022.
- [8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.
- [9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [10] 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*, pages 1861–1870. PMLR, 2018.
- [11] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.
- [12] A. Baisero, B. Daley, and C. Amato. Asymmetric dqn for partially observable reinforcement learning. In *The 38th Conference on Uncertainty in Artificial Intelligence*, 2022.
- [13] A. Baisero and C. Amato. Unbiased asymmetric reinforcement learning under partial observability. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pages 44–52, 2022.
- [14] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel. Asymmetric actor critic for image-based robot learning. In *14th Robotics: Science and Systems, RSS 2018*. MIT Press Journals, 2018.
- [15] A. Warrington, J. W. Lavington, A. Scibior, M. Schmidt, and F. Wood. Robust asymmetric learning in pomdps. In *International Conference on Machine Learning*, pages 11013–11023. PMLR, 2021.



- [16] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.
- [17] M. Bain and C. Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129, 1995.
- [18] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.
- [19] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert. Learning monocular reactive uav control in cluttered natural environments. In *2013 IEEE international conference on robotics and automation*, pages 1765–1772. IEEE, 2013.
- [20] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2015.
- [21] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.
- [22] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- [23] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.
- [24] J. Ho and S. Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.
- [25] L. Weihs, U. Jain, I.-J. Liu, J. Salvador, S. Lazebnik, A. Kembhavi, and A. Schwing. Bridging the imitation gap by adaptive insubordination. *Advances in Neural Information Processing Systems*, 34, 2021.
- [26] K. J. Astrom. Optimal control of markov decision processes with incomplete state estimation. *J. Math. Anal. Applic.*, 10:174–205, 1965.
- [27] S. P. Singh, T. Jaakkola, and M. I. Jordan. Learning without state-estimation in partially observable markovian decision processes. In *Machine Learning Proceedings 1994*, pages 284–292. Elsevier, 1994.
- [28] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.
- [29] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [30] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.
- [31] M. Chevalier-Boisvert, L. Willems, and S. Pal. Minimalistic gridworld environment for openai gym. <https://github.com/maximecb/gym-minigrid>, 2018.
- [32] H. Nguyen. Pomdp robot domains. <https://github.com/hai-h-nguyen/pomdp-domains>, 2021.
- [33] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

- [34] T. Ni, B. Eysenbach, and R. Salakhutdinov. Recurrent model-free rl can be a strong baseline for many pomdps. In *International Conference on Machine Learning*, pages 16691–16723. PMLR, 2022.
- [35] T. Gangwani, J. Lehman, Q. Liu, and J. Peng. Learning belief representations for imitation learning in pomdps. In *Uncertainty in Artificial Intelligence*, pages 1061–1071. PMLR, 2020.
- [36] P. Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.
- [37] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.
- [38] I. Kostrikov. Pytorch implementations of reinforcement learning algorithms. <https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail>, 2018.
- [39] S. James, A. J. Davison, and E. Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. In *Conference on Robot Learning*, pages 334–343. PMLR, 2017.
- [40] K. Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3):287–296, 1985.
- [41] G. Liu, L. Zhao, P. Zhang, J. Bian, T. Qin, N. Yu, and T.-Y. Liu. Demonstration actor critic. *Neurocomputing*, 434:194–202, 2021.
- [42] J. Achiam. Spinning Up in Deep Reinforcement Learning. 2018.
- [43] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.
- [44] D. Wang, C. Kohler, X. Zhu, M. Jia, and R. Platt. Bulletarm: An open-source robotic manipulation benchmark and learning framework. In *The International Symposium of Robotics Research*, 2022.
- [45] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.
- [46] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.
- [47] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

## 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]).

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

where  $w(s) : \mathcal{S} \rightarrow [0, 1]$ ,  $\mu$  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:

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

with  $m_\alpha(x) = \exp(-\alpha x)$ , and where  $\pi^{\text{IL}}$  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  $\pi$  by imitating a state expert  $\mu$ , by minimizing the following *cross-observability imitation learning* (COIL) objective,

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

where  $\mathcal{D}$  is a replay buffer containing (truncated) episodes. We denote the optimal solution to COIL as  $\pi^{\text{COIL}}$  and denote its performance as  $V^{\text{COIL}}(\mu)$ , which is a function of the state expert  $\mu$ . 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,

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

The process of optimizing the objective in Equation (10) can be considered as a projection of the state expert  $\mu$  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  $\pi^{\text{IL}}$  and  $\mu$ . In contrast, the optimality gap is a measure of the performance difference between  $\pi^*$  and  $\pi^{\text{COIL}}$ , 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  $\mu_\theta : \mathcal{S} \rightarrow [0, 1]^{|A|}$ , and the value model can also change respectively to  $Q_\phi : \mathcal{S} \rightarrow \mathbb{R}^{|A|}$ . Then, Equations (2), (4) and (5) become

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

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

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

## 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  $j \leftarrow 1, \dots, \#\text{updates}$  do
14:      Random sample a batch of  $B$  episodes from  $\mathcal{D}$ 
15:      Compute targets for the Q functions:
16:      
$$y \leftarrow \begin{cases} r & \text{if episode done} \\ r + \gamma \left( \min_{i \in \{1,2\}} Q_{\bar{\phi}_i}(h', \tilde{a}') + \alpha (\mu(s') - m_\theta(h'))^2 \right) & \text{if episode not done} \end{cases}$$

17:      where  $\tilde{a}' \sim \pi_\theta(h')$ 
18:      Update Q-function by one step of gradient descent using
19:       $\mathcal{L}_{\phi_i} \leftarrow \frac{1}{|B|} \sum_{h,s \in B} (Q_{\phi_i}(h, a) - y)^2$  for  $i \in \{1, 2\}$ 
20:      Update policy by one step of gradient descent using
21:       $\mathcal{L}_\theta \leftarrow \frac{1}{|B|} \sum_{h,s \in B} \left( -\min_{i \in \{1,2\}} Q_{\phi_i}(h, \tilde{a}) - \alpha (\mu(s) - m_\theta(h))^2 \right)$ ,
22:      where  $\tilde{a} \sim \pi_\theta(h)$ 
23:      Adjust  $\alpha$  by one step of gradient descent using
24:       $\mathcal{L}_\alpha \leftarrow \frac{1}{|B|} \sum_{h,s \in B} \left( \alpha \bar{D} - \alpha (\mu(s) - m_\theta(h))^2 \right)$ 
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  $D$
  - 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**  $j \leftarrow 1, \dots, \#\text{updates}$  **do**
  - 14:          Random sample a batch of  $B$  episodes from  $\mathcal{D}$
  - 15:          Compute targets for the Q functions:
  - 16:              $y \leftarrow \begin{cases} r & \text{if episode done} \\ r + \gamma \pi_\theta(h')^\top (\min_{i \in \{1,2\}} Q_{\bar{\phi}_i}(h') + \alpha \mu(s')^\top \log(\pi_\theta(h'))) & \text{if episode not done} \end{cases}$
  - 17:          Update Q-function by one step of gradient descent using
  - 18:              $\mathcal{L}_{\phi_i} \leftarrow \frac{1}{|B|} \sum_{h,s \in B} (Q_{\phi_i}(h, a) - y)^2$  for  $i \in \{1, 2\}$
  - 19:          Update policy by one step of gradient descent using
  - 20:              $\mathcal{L}_\theta \leftarrow \frac{1}{|B|} \sum_{h,s \in B} \pi_\theta(h)^\top (-\min_{i \in \{1,2\}} Q_{\phi_i}(h, \pi_\theta(h)) - \alpha \mu(s)^\top \log(\pi_\theta(h)))$
  - 21:          Adjust  $\alpha$  by one step of gradient descent using
  - 22:              $\mathcal{L}_\alpha \leftarrow \frac{1}{|B|} \sum_{h,s \in B} \pi_\theta(h)^\top (\alpha \bar{D} + \alpha \mu(s)^\top \log(\pi_\theta(h)))$
  - 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  $7 \times 7 \times 3$
- **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](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

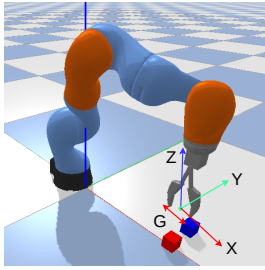


Figure 7: Block-Picking.

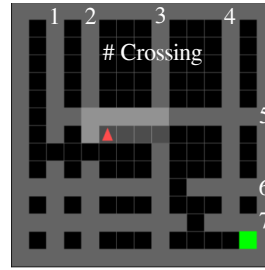


Figure 8: Minigrid-Crossing.

- **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  $84 \times 84 \times 1$ , 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  $84 \times 84 \times 3$
- **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  $7 \times 7 \times 3$  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-cor122> 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.

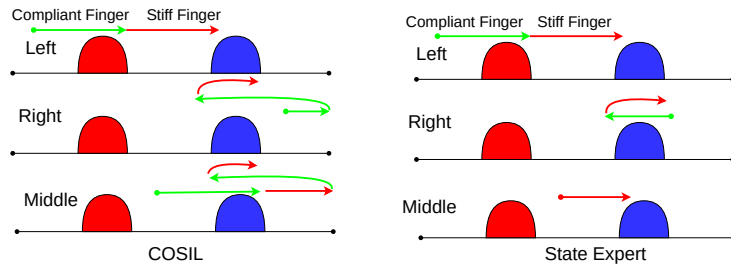


Figure 9: The policy learned by COSIL in Bumps-1D [6, 32] behaves differently depending on the relative position of the gripper and the positions of the two bumps.

### 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.

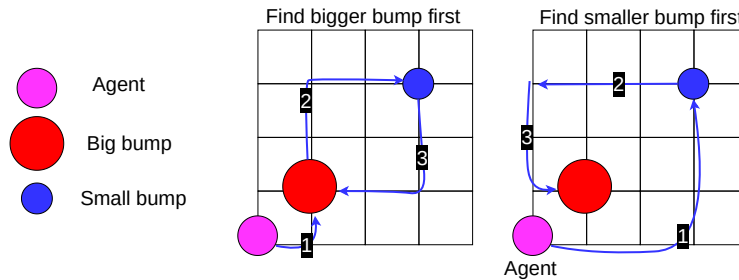


Figure 10: The policy learned by COSIL in Bumps-2D [6, 32] acts differently depending on which bump is found first. Numbers denote the order in the action sequences.

### 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.

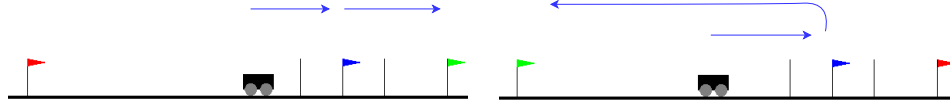


Figure 11: The policy learned by COSIL in Car-Flag [32] first goes to the blue flag to check for the side of the green flag before navigating to it. The green flag can be on the left or on the right.

## 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).

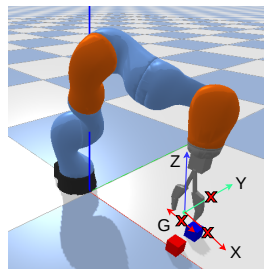


Figure 12: In Block-Picking, movements along certain axes are temporarily disabled during policy deployment to the real robot when the gripper picks a block that is supposed to be immovable. Disabling is necessary; otherwise, the powerful gripper will easily move the supposedly immovable block regardless 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]	PP0: num-processes = 16, num-step = maximum episode length, $\lambda_{GAE} = 0.95$ 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]	PP0: $\lambda_{GAE} = 1.0$ , 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  $\alpha$ )
- BC2SAC: The percentage of total timesteps trained with behavioral cloning  $p_{BC}$  in  $\{10\%, 20\%, \dots, 90\%\}$ .
- BC+SAC: The weight for BC loss  $w_{BC} \in \{0.1, 0.2, \dots, 0.9\}$
- 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.

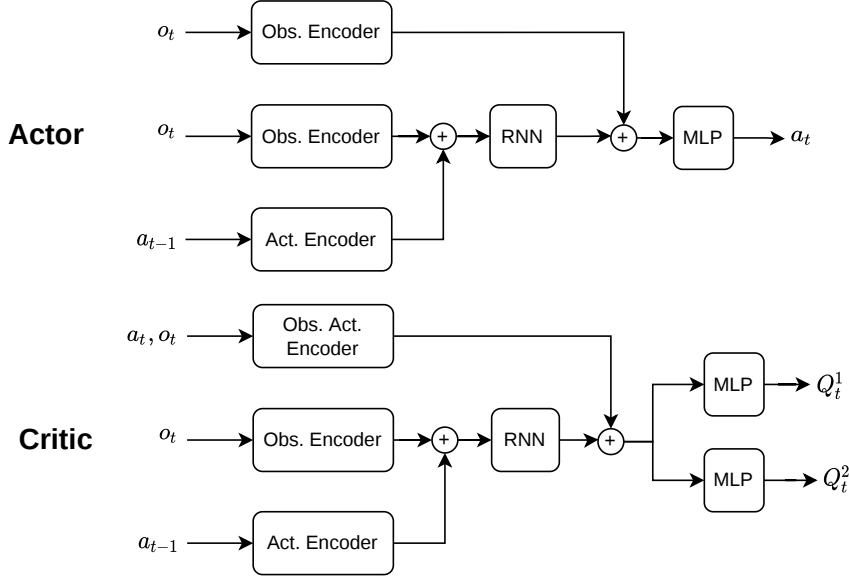


Figure 13: The network structure of COSIL.

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

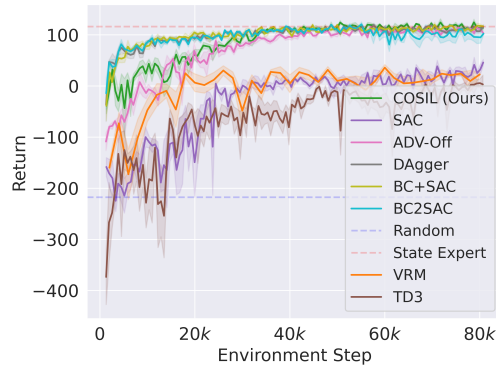


Figure 14: Performance of all methods on LunarLander-V.

### 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>.

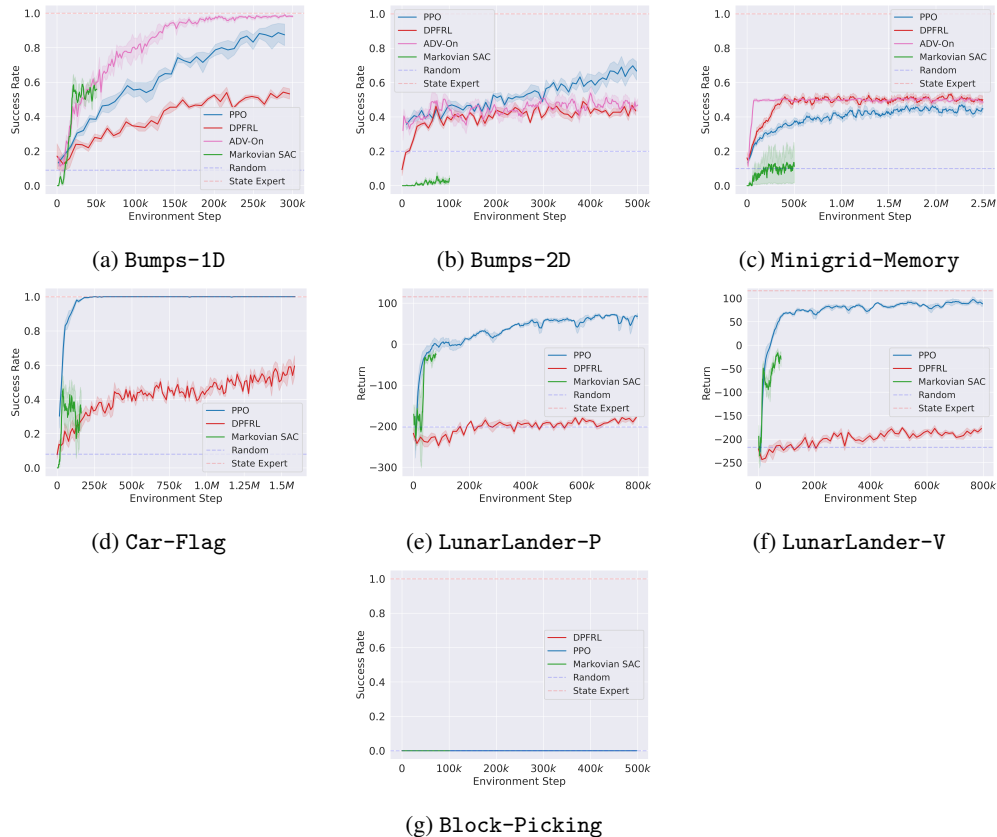


Figure 15: Performances of additional baselines (recurrent PPO [9], ADV-On [25], Markovian SAC [34], and DPFRL [2]).