Appendix: Bridging the Imitation Gap by Adaptive Insubordination

The appendix includes theoretical extensions of ideas presented in the main paper and details of empirical analysis. We structure the appendix into the following subsections:

A Additional Information

A.1 Formal treatment of Example 2

Let $N \geq 1$ and consider a 1-dimensional grid-world with states $\mathcal{S} = \{-N, N\} \times \{0, \ldots, T\} \times \{-N, \ldots, N\}^T$. Here $g \in \{-N, N\}$ are possible goal positions, elements $t \in \{0, \ldots, T\}$ correspond to the episode's current timestep, and $(p_i)_{i=1}^T \in \{-N, \ldots, N\}^T$ correspond to possible agent trajectories of length T. Taking action $a \in \mathcal{A} = \{\text{left}, \text{right}\} = \{-1, 1\}$ in state $(g, t, (p_i)_{i=1}^T) \in \mathcal{S}$ results in the deterministic transition to state $(g, t+1, (p_1, \ldots, p_t, \text{clip}(p_t + a, -N, N), 0, \ldots, 0))$. An episode start state is chosen uniformly at random from the set $\{(\pm N, 0, (0, \ldots, 0))\}$ and the goal of the agent is to reach some state $(g, t, (p_i)_{i=1}^T)$ with $p_t = g$ in the fewest steps possible. We now consider a collection of filtration functions f^i, that allow the agent to see spaces up to i steps left/right of its current position but otherwise has perfect memory of its actions. See Figs. 2c, 2d for examples of f^1- and f^2-partial observations. For $0 \leq i \leq N$ we define f^i so that

$$f^i(g, t, (p_i)_{i=1}^T) = ((\ell_0, \ldots, \ell_t), (p_1 - p_0, \ldots, p_t - p_{t-1})) \quad \text{and} \tag{5}$$

$$\ell_j = (1_{[p_j + k = N]} - 1_{[p_j + k = -N]} \mid k \in \{-i, \ldots, i\}) \quad \text{for } 0 \leq j \leq t. \tag{6}$$

Here ℓ_j is a tuple of length $2 \cdot i + 1$ and corresponds to the agent's view at timestep j while $p_{k+1} - p_k$ uniquely identifies the action taken by the agent at timestep k. Let π^{exp} be the optimal policy given full state information so that $\pi^{\text{exp}}(g, t, (p_i)_{i=1}^T) = (1_{[g=-N]}, 1_{[g=N]})$ and let μ be a uniform distribution over states in $\mathcal{S}$. It is straightforward to show that an agent following policy $\pi^{\text{IL}}_{f^i}$ will take random actions until it is within a distance of i from one of the corners $\{-N, N\}$ after which it will head directly to the goal, see the policies highlighted in Figs. 2c, 2d. The intuition for this result is straightforward: until the agent observes one of the corners it cannot know if the goal is to the right or left and, conditional on its observations, each of these events is equally likely under μ. Hence in half of these events the expert will instruct the agent to go right and in the other half to go left. The cross entropy loss will thus force $\pi^{\text{IL}}_{f^i}$ to be uniform in all such states. Formally, we will have, for $s = (g, t, (p_i)_{i=1}^T)$, $\pi^{\text{IL}}_{f^i}(s) = \pi^{\text{exp}}(s)$ if and only if $\min_{0 \leq q \leq t}(p_q) - i \leq -N$ or $\max_{0 \leq q \leq t}(p_q) + i \geq N$ and, for all other s, we have $\pi^{\text{IL}}_{f^i}(s) = (1/2, 1/2)$. In Sec. 4, see also Fig. 5, we train f^i-partial policies with f^j-optimal experts for a 2D variant of this example. ■

[4]We overload main paper's notation $d^0(\pi, \pi_f)(s)$ with $d^0_\pi(\pi_f)(s)$

A.2 Proof of Proposition 1

We wish to show that the minimizer of $\mathbb{E}_\mu[-\pi^{\text{exp}}_{f^e}(S) \odot \log \pi_f(S)]$ among all f-partial policies π_f is the policy $\overline{\pi} = \mathbb{E}_\mu[\pi^{\text{exp}}(S) \mid f(S)]$. This is straightforward, by the law of iterated expectations and as $\pi_f(s) = \pi_f(f(s))$ by definition. We obtain

$$\begin{aligned}
\mathbb{E}_\mu[-\pi^{\text{exp}}_{f^e}(S) \odot \log \pi_f(S)] &= -\mathbb{E}_\mu[E_\mu[\pi^{\text{exp}}_{f^e}(S) \odot \log \pi_f(S) \mid f(S)]] \\
&= -\mathbb{E}_\mu[E_\mu[\pi^{\text{exp}}_{f^e}(S) \odot \log \pi_f(f(S)) \mid f(S)]] \\
&= -\mathbb{E}_\mu[E_\mu[\pi^{\text{exp}}_{f^e}(S) \mid f(S)] \odot \log \pi_f(f(S))] \\
&= \mathbb{E}_\mu[-\overline{\pi}(f(S)) \odot \log \pi_f(f(S))]\ . \qquad (7)
\end{aligned}$$

Now let $s \in \mathcal{S}$ and let $o = f(s)$. It is well known, by Gibbs' inequality, that $-\overline{\pi}(o) \odot \log \pi_f(o)$ is minimized (in $\pi_f(o)$) by letting $\pi_f(o) = \overline{\pi}(o)$ and this minimizer is feasible as we have assumed that Π_f contains *all* f-partial policies. Hence it follows immediately that Eq. (7) is minimized by letting $\pi_f = \overline{\pi}$ which proves the claimed proposition.

A.3 Other Distance Measures

As discussed in Section 3.2, there are several different choices one may make when choosing a measure of distance between the expert policy π^{exp} and an f-partial policy π_f at a state $s \in \mathcal{S}$. The measure of distance we use in our experiments, $d^0_{\pi^{\text{exp}}}(\pi_f)(s) = d(\pi^{\text{exp}}(s), \pi_f(s))$, has the (potentially) undesirable property that $f(s) = f(s')$ does not imply that $d^0_{\pi^{\text{exp}}}(\pi_f)(s) = d^0_{\pi^{\text{exp}}}(\pi_f)(s')$. While an in-depth evaluation of the merits of different distance measures is beyond this current work, we suspect that a careful choice of such a distance measure may have a substantial impact on the speed of training. The following proposition lists a collection of possible distance measures with a conceptual illustration given in Fig. 6.

Proposition 2. *Let $s \in \mathcal{S}$ and $o = f(s)$ and for any $0 < \beta < \infty$ define, for any policy π and f-partial policy π_f,*

$$d^\beta_{\mu,\pi}(\pi_f)(s) = E_\mu[\big(d^0_\pi(\pi_f)(S)\big)^\beta \mid f(S) = f(s)]^{1/\beta}, \qquad (8)$$

with $d^\infty_{\mu,\pi}(\pi_f)(s)$ equalling the essential supremum of $d^0_\pi(\pi_f)$ under the conditional distribution $P_\mu(\cdot \mid f(S) = f(s))$. As a special case note that

$$d^1_{\mu,\pi}(\pi_f)(s) = E_\mu[d^0_\pi(\pi_f)(S) \mid f(S) = f(s)].$$

Then, for all $\beta \geq 0$ and $s \in \mathcal{S}$ (almost surely μ), we have that $\pi(s) \neq \pi_f(f(s))$ if and only if $d^\beta_\pi(\pi_f)(s) > 0$.

Proof. This statement follows trivially from the definition of π^{IL} and the fact that $d(\pi, \pi') \geq 0$ with $d(\pi, \pi') = 0$ if and only if $\pi = \pi'$. □

The above proposition shows that any d^β can be used to consistently detect differences between π^{exp} and π^{IL}_f, *i.e.*, it can be used to detect the imitation gap. Notice also that for any $\beta > 0$ we have that $d^\beta_{\mu,\pi^{\text{exp}}}(\pi^{\text{IL}}_f)(s) = d^\beta_{\mu,\pi^{\text{exp}}}(\pi^{\text{IL}}_f)(s')$ whenever $f(s) = f(s')$.

As an alternative to using d^0, we now describe how $d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{IL}}_f)(s)$ can be estimated in practice during training. Let π^{aux}_f be an estimator of π^{IL}_f as usual. To estimate $d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{IL}}_f)(s)$ we assume we have access to a function approximator $g_\psi : \mathcal{O}_f \to \mathbb{R}$ parameterized by $\psi \in \Psi$, *e.g.*, a neural network. Then we estimate $d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{IL}}_f)(s)$ with $g_{\widehat{\psi}}$ where $\widehat{\psi}$ is taken to be the minimizer of the loss

$$\mathcal{L}_{\mu,\pi^{\text{exp}},\pi^{\text{aux}}_f}(\psi) = E_\mu\Big[\Big(d(\pi^{\text{exp}}(S), \pi^{\text{aux}}_f(f(S))) - g_\psi(f(S))\Big)^2\Big]. \qquad (9)$$

The following proposition then shows that, assuming that $d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f) \in \{g_\psi \mid \psi \in \Psi\}$, $g_{\widehat{\psi}}$ will equal $d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)$ and thus $g_{\widehat{\psi}}$ may be interpreted as a plug-in estimator of $d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{IL}}_f)$.

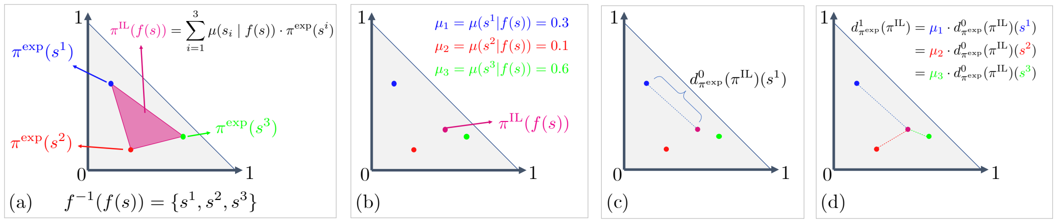
$\pi^{\text{IL}}(f(s)) = \sum_{i=1}^{3} \mu(s_i \mid f(s)) \cdot \pi^{\text{exp}}(s^i)$
$\pi^{\text{exp}}(s^1)$
$\pi^{\text{exp}}(s^2)$
$\pi^{\text{exp}}(s^3)$
(a) $f^{-1}(f(s)) = \{s^1, s^2, s^3\}$
$\mu_1 = \mu(s^1|f(s)) = 0.3$
$\mu_2 = \mu(s^2|f(s)) = 0.1$
$\mu_3 = \mu(s^3|f(s)) = 0.6$
$\pi^{\text{IL}}(f(s))$
(b)
$d^0_{\pi^{\text{exp}}}(\pi^{\text{IL}})(s^1)$
(c)
$d^1_{\pi^{\text{exp}}}(\pi^{\text{IL}}) = \mu_1 \cdot d^0_{\pi^{\text{exp}}}(\pi^{\text{IL}})(s^1)$
$= \mu_2 \cdot d^0_{\pi^{\text{exp}}}(\pi^{\text{IL}})(s^2)$
$= \mu_3 \cdot d^0_{\pi^{\text{exp}}}(\pi^{\text{IL}})(s^3)$
(d)
0
1

Figure 6: **Concept Illustration.** Here we illustrate several of the concepts from our paper. Suppose our action space $\mathcal{A}$ contains three elements. Then for any $s \in \mathcal{S}$ and policy π, the value $\pi(s)$ can be represented as a single point in the 2-dimensional probability simplex $\{(x, y) \in \mathbb{R}^2 \mid x \geq 0, y \geq 0, x + y \leq 1\}$ shown as the grey area in (a). Suppose that the fiber $f^{-1}(f)$ contains the three unique states s^1, s^2, and s^3. In (a) we show the hypothetical values of π^{exp} when evaluated at these points. Proposition 1 says that $\pi^{\text{IL}}(s)$ lies in the convex hull of $\{\pi^{\text{exp}}(s^i)\}_{i=1}^{3}$ visualized as a magenta triangle in (a). Exactly where $\pi^{\text{IL}}(s)$ lies depends on the probability measure μ, in (b) we show how a particular instantiation of μ may result in a realization of $\pi^{\text{IL}}(s)$ (not to scale). (c) shows how $d^1_{\pi^{\text{exp}}}$ measures the distance between $\pi^{\text{exp}}(s^1)$ and $\pi^{\text{IL}}(s^1)$. Notice that it ignores s^2 and s^3. In (d), we illustrate how $d^0_{\pi^{\text{exp}}}$ produces a "smoothed" measure of distance incorporating information about all s^i.

Proposition 3. *For any $\psi \in \Psi$,*

$$\mathcal{L}_{\mu,\pi^{exp},\pi^{aux}_f}(\psi) = E_\mu[(d^1_{\mu,\pi^{exp}}(\pi^{aux}_f)(S) - g_\psi(f(S)))^2] + c,$$

where $c = E_\mu[(d(\pi^{exp}(S), \pi^{aux}(f(S))) - d^1_{\mu,\pi^{exp},\widehat{\pi}}(S))^2]$ is constant in ψ and this implies that if $d^1_{\mu,\pi^{exp}}(\pi^{aux}_f) \in \{g_\psi \mid \psi \in \Psi\}$ then $g_{\widehat{\psi}} = d^1_{\mu,\pi^{exp}}(\pi^{aux}_f)$.

Proof. In the following we let $O_f = f(S)$. We now have that

$$\begin{aligned}
&E_\mu[\big(d(\pi^{\text{exp}}(S), \pi^{\text{aux}}_f(O_f)) - g_\psi(O_f)\big)^2] \\
&\quad= E_\mu[\big((d(\pi^{\text{exp}}(S), \pi^{\text{aux}}_f(O_f)) - d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(S)) + (d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(S) - g_\psi(O_f))\big)^2] \\
&\quad= E_\mu[(d(\pi^{\text{exp}}(S), \pi^{\text{aux}}_f(O_f)) - d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(S))^2] + E_\mu[(d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(S) - g_\psi(O_f)))^2] \\
&\qquad + 2 \cdot E_\mu[((d(\pi^{\text{exp}}(S), \pi^{\text{aux}}_f(O_f)) - d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(S)) \cdot (d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(S) - g_\psi(O_f)))] \\
&\quad= c + E_\mu[(d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(S) - g_\psi(O_f)))^2] \\
&\qquad + 2 \cdot E_\mu[((d(\pi^{\text{exp}}(S), \pi^{\text{aux}}_f(O_f)) - d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(S)) \cdot (d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(S) - g_\psi(O_f)))].
\end{aligned}$$

Now as as $d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(s) = d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(s')$ for any s, s' with $f(s) = f(s')$ we have that $d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(S) - g_\psi(O_f)$ is constant conditional on O_f and thus

$$\begin{aligned}
&E_\mu[(d(\pi^{\text{exp}}(S), \pi^{\text{aux}}_f(O_f)) - d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(S)) \cdot (d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(S) - g_\psi(O_f)) \mid O_f] \\
&\quad= E_\mu[(d(\pi^{\text{exp}}(S), \pi^{\text{aux}}_f(O_f)) - d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(S) \mid O_f] \cdot E_\mu[d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(S) - g_\psi(O_f) \mid O_f] \\
&\quad= E_\mu[d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(S) - d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(S) \mid O_f] \cdot E_\mu[d^1_{\mu,\pi^{\text{exp}}}(\pi^{\text{aux}}_f)(S) - g_\psi(O_f) \mid O_f] \\
&\quad= 0.
\end{aligned}$$

Combining the above results and using the law of iterated expectations gives the desired result. □

A.4 Future Directions in Improving Distance Estimators

In this section we highlight possible directions towards improving the estimation of $d^0_{\pi^{\text{exp}}}(\pi^{\text{IL}}_f)(s)$ for $s \in \mathcal{S}$. As a comprehensive study of these directions is beyond the scope of this work, our aim in this section is intuition over formality. We will focus on d^0 here but similar ideas can be extended to other distance measures, *e.g.*, those in Sec. A.3.

As discussed in the main paper, we estimate $d^0_{\pi^{\exp}}(\pi^{\text{IL}}_f)(s)$ by first estimating π^{IL}_f with π^{aux}_f and then forming the "plug-in" estimator $d^0_{\pi^{\exp}}(\pi^{\text{aux}}_f)(s)$. For brevity, we will write $d^0_{\pi^{\exp}}(\pi^{\text{aux}}_f)(s)$ as $\widehat{d}$. While such plug-in estimators are easy to estimate and conceptually compelling, they need not be statistically efficient. Intuitively, the reason for this behavior is because we are spending too much effort in trying to create a high quality estimate π^{aux}_f of π^{IL}_f when we should be willing to sacrifice some of this quality in service of obtaining a better estimate of $d^0_{\pi^{\exp}}(\pi^{\text{IL}}_f)(s)$. Very general work in this area has brought about the targeted maximum-likelihood estimation (TMLE) (van der Laan & Gruber, 2016) framework. Similar ideas may be fruitful in improving our estimator $\widehat{d}$.

Another weakness of $\widehat{d}$ discussed in the main paper is that is not prospective. In the main paper we assume, for readability, that we have trained the estimator π^{aux}_f before we train our main policy. In practice, we train π^{aux}_f alongside our main policy. Thus the quality of π^{aux}_f will improve throughout training. To clarify, suppose that, for $t \in [0,1]$, $\pi^{\text{aux}}_{f,t}$ is our estimate of π^{IL}_f after $(100 \cdot t)\%$ of training has completed. Now suppose that $(100 \cdot t)\%$ of training has completed and we wish to update our main policy using the ADVISOR loss given in Eq. (2). In our current approach we estimate $d^0_{\pi^{\exp}}(\pi^{\text{IL}}_f)(s)$ using $d^0_{\pi^{\exp}}(\pi^{\text{aux}}_{f,t})(s)$ when, ideally, we would prefer to use $d^0_{\pi^{\exp}}(\pi^{\text{aux}}_{f,1})(s)$ from the end of training. Of course we will not know the value of $d^0_{\pi^{\exp}}(\pi^{\text{aux}}_{f,1})(s)$ until the end of training but we can, in principle, use time-series methods to estimate it. To this end, let q_ω be a time-series model with parameters $\omega \in \Omega$ (*e.g.*, q_ω might be a recurrent neural network) and suppose that we have stored the model checkpoints $(\pi^{\text{aux}}_{f,i/K} \mid i/K \le t)$. We can then train q_ω to perform forward prediction, for instance to minimize

$$\sum_{j=1}^{\lfloor t\cdot K \rfloor} \left(d^0_{\pi^{\exp}}(\pi^{\text{aux}}_{f,j/K})(s) - q_\omega(s, (\pi^{\text{aux}}_{f,i/K}(s))_{i=1}^{j-1}) \right)^2 ,$$

and then use this trained q_ω to predict the value of $d^0_{\pi^{\exp}}(\pi^{\text{aux}}_{f,1})(s)$. The advantage of this prospective estimator q_ω is that it can detect that the auxiliary policy will eventually succeed in exactly imitating the expert in a given state and thus allow for supervising the main policy with the expert cross entropy loss earlier in training. The downside of such a method: it is significantly more complicated to implement and requires running inference using saved model checkpoints.

A.5 Additional Task Details

In Sec. 4.1, we introduced the different tasks where we compare ADVISOR with various other IL and RL methods. Here, we provide additional details for each of them along with information about observation space associated with each task. For training details for the tasks, please see Sec. A.9.

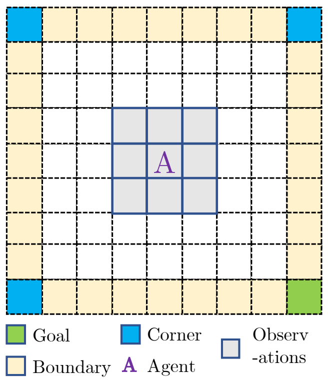

Figure 7: 2D-Lighthouse

A.5.1 PoisonedDoors (PD)

This environment is a reproduction of our example from Sec. 1. An agent is presented with $N = 4$ doors $d_1, \ldots, d_4$. Door d_1 is locked, requiring a fixed $\{0,1,2\}^{10}$ code to open, but always results in a reward of 1 when opened. For some randomly chosen $j \in \{2,3,4\}$, opening door d_j results in a reward of 2 and for $i \notin \{1,j\}$, opening door d_i results in a reward of -2. The agent must first choose a door after which, if it has chosen door 1, it must enter the combination (receiving a reward of 0 if it enters the incorrect combination) and, otherwise, the agent immediately receives its reward. See Fig. 1.

A.5.2 2D-Lighthouse (2D-LH)

2D variant of the exemplar grid-world task introduced in Ex. 2, aimed to empirically verify our analysis of the imitation gap. A reward awaits at a randomly chosen corner of a square grid of size $2N+1$ and the agent can only see the local region, a square of size $2i+1$ about itself (an f^i-partial observation). Additionally, all f^i allow the agent access to it's previous action. As explained in Ex. 2, we experiment with optimizing f^i-policies when given supervision from f^j-optimal experts (*i.e.*, experts that are optimal when restricted to f^j-partial observations). See Fig. 7 for an illustration.

A.5.3 WALLCROSSING (WC)

Initialized on the top-left corner the agent must navigate to the bottom-right goal location. There exists at least one path from start to end, navigating through obstacles. Refer to Fig. 3a where, for illustration, we show a simpler grid. Our environment is of size 25×25 with 10 `walls` ('S25, N10' as per the notation of (Chevalier-Boisvert et al., 2018b)), which are placed vertically or horizontally across the grid. The expert is a shortest path agent with access to the entire environment's connectivity graph and is implemented via the `networkx` python library.

A.5.4 LAVACROSSING (LC)

Similar to WALLCROSSING in structure and expert, except that obstacles are `lava` instead of `walls`. Unlike `walls` (into which the agent can bump without consequence) here the episode terminates if the agent steps on `lava`. See Fig. 3b. This LC environment has size 25×25 with 10 `lava` rivers ('S25, N10').

A.5.5 WC/LC SWITCH

In this task the agent faces a more challenging filtration function. In addition to navigational actions, agents for this task have a 'switch' action. Using this switch action, the agents can switch-on the lights of an otherwise darkened environment which is implemented as an observation tensor of all zeros. In WC, even in the dark, an agent can reach the target by taking random actions with non-negligible probability. Achieving this in LC is nearly impossible as random actions will, with high probability, result in stepping into `lava` and thereby immediately end the episode.

We experiment with two variants of this 'switch' – ONCE and FAULTY. In the ONCE SWITCH variant, once the the 'switch' action is taken, the lights remain on for the remainder of the episode. This is implemented as the unaffected observation tensor being available to the agent. In contrast, in the FAULTY SWITCH variant, taking the 'switch' action will only turn the lights on for a single timestep. This is implemented as observations being available for one timestep followed by zero tensors (unless the 'switch' action is executed again).

The expert for these tasks is the same as for WC and LC. Namely, the expert always takes actions along the shortest path from the agents current position to the goal and is unaffected by whether the light is on or off. For the expert-policy-based methods this translates to the learner agent getting perfect (navigational) supervision while struggling in the dark, with no cue for trying the switch action. For the expert-demonstrations-based methods this translates to the demonstrations being populated with blacked-out observations paired with perfect actions: such actions are, of course, difficult to imitate. As FAULTY is more difficult than ONCE (and LC more difficult than WC) we set grid sizes to reduce the difference in difficulty between tasks. In particular, we choose to set WC ONCE SWITCH on a (S25, N10) grid and the LC ONCE SWITCH on a (S15, N7) grid. Moreover, WC FAULTY SWITCH is set with a (S15, N7) grid and LC FAULTY SWITCH with a (S9, N4) grid.

A.5.6 WC/LC CORRUPT

In the SWITCH task, we study agents with observations affected by a challenging filtration function. In this task we experiment with corrupting the expert's actions. The expert policy flips over to a random policy when the expert is N_C steps away from the goal. For the expert-policy-based method this translates to the expert outputting uniformly random actions once it is within N_C steps from the target. For the expert-demonstrations-based methods this translates to the demonstrations consisting of some valid (observation, expert action) tuples, while the tuples close to the target have the expert action sampled from a uniform distribution over the action space. WC CORRUPT is a (S25, N10) grid with $N_C = 15$, while the LC CORRUPT is significantly harder, hence is a (S15, N7) grid with $N_C = 10$.

Table 1: **Baseline details.** IL/RL: Nature of learning, Expert supervision: the type of expert supervision leveraged by each method, Hps. searched: hps. that were randomly searched over, fairly done with the same budget (see Sec. A.8 for details).

#	Method	IL/RL	Expert supervision	Hps. searched
1	BC	IL	Policy	lr
2	†	IL	Policy	lr, stage-split
3	$\text{BC}^{\text{tf=1}}$	IL	Policy[7]	lr
4	PPO	RL	Policy	lr
5	BC $\rightarrow$ PPO	IL$\rightarrow$RL	Policy	lr, stage-split
6	† $\rightarrow$ PPO	IL$\rightarrow$RL	Policy	lr, stage-split
7	$\text{BC}^{\text{tf=1}}$ $\rightarrow$ PPO	IL$\rightarrow$RL	Policy	lr, stage-split
8	BC + PPO	IL+RL	Policy	lr
9	BC^{demo}	IL	Demonstrations	lr
10	BC^{demo} + PPO	IL+RL	Demonstrations	lr
11	ADV	IL+RL	Policy	lr, α
12	† $\rightarrow$ ADV	IL+RL	Policy	lr, α, stage-split
13	$\text{BC}^{\text{tf=1}}$ $\rightarrow$ ADV	IL+RL	Policy	lr, α, stage-split
14	BC^{demo} + ADV	IL+RL	Demonstrations	lr, α

A.5.7 Observation space

Within our 2D-LH environment we wish to train our agent in the context of Proposition 1 so that the agent may learn any f-partial policy. As the 2D-LH environment is quite simple, we are able to uniquely encode the state observed by an agent using a $4^4 \cdot 5^2 = 6400$ dimensional $\{0, 1\}$-valued vector such that any f-partial policy can be represented as a linear function applied to this observation (followed by a soft-max).[5] Within the PD environment the agent's observed state is very simple: at every timestep the agent observes an element of $\{0, 1, 2, 3\}$ with 0 denoting that no door has yet been chosen, 1 denoting that the agent has chosen door d_1 but has not begun entering the code, 2 indicating that the agent has chosen door d_1 and has started entering the code, and 3 representing the final terminal state after a door has been opened or combination incorrectly entered. The MiniGrid environments (Chevalier-Boisvert et al., 2018b) enable agents with an egocentric "visual" observation which, in practice, is an integer tensor of shape $7 \times 7 \times 3$, where the channels contain integer labels corresponding to the cell's type, color, and state. Kindly see (Chevalier-Boisvert et al., 2018b;a) for details. For the above tasks, the cell types belong to the set of (`empty, lava, wall, goal`).

A.6 Additional Baseline Details

In Tab. 1, we include details about the baselines considered in this work, including – purely IL $(1-3, 9)$, purely RL (4), a sequential combination of them $(5-7)$, static combinations of them $(8, 10)$, and our dynamic combinations $(11-14)$. Moreover, we study methods which learn from both expert policy (expert action available for any state) and expert demonstrations (offline dataset of pre-collected trajectories). The hyperparameters (hps) we consider for optimization in our study have been chosen as those which, in preliminary experiments, had a substantial impact on model performance. This includes the learning rate (lr), portion of the training steps devoted to the first stage in methods with two stages (stage-split), and the temperature parameter in the weight function (α).[6] Implicitly, the random environment seed can also be seen as a hyperparameter. We sample hyperparameters uniformly at random. In particular, we sample lr from $[10^{-4}, 0.5)$ on a log-scale, stage-split from $[0.1, 0.9)$, and α from $\{5.0, 20.0\}$.

[5]As the softmax function prevents us from learning a truly deterministic policy we can only learn a policy arbitrarily close to such policies. In our setting, this distinction is irrelevant.

[6]See Sec. 3.2 for definition of the weight function for ADVISOR.

[7]While implemented with supervision from expert policy, due to the teacher forcing being set to 1.0, this method can never explore beyond states (and supervision) in expert demonstrations.

A.7 Architecture Details

2D-LH model. As discussed in Sec. A.5.7, we have designed the observation given to our agent so that a simple linear layer followed by a soft-max function is sufficient to capture any f-partial policy. As such, our main and auxiliary actor models for this task are simply linear functions mapping the input 6400-dimensional observation to a 4-dimensional output vector followed by a soft-max non-linearity. The critic is computed similarly but with a 1-dimensional output and no non-linearity.

PD model. Our PD model has three sequential components. The first embedding layer maps a given observation, a value in $\{0, 1, 2, 3\}$, to an 128-dimensional embedding. This 128-dimensional vector is then fed into a 1-layer LSTM (with a 128-dimensional hidden state) to produce an 128-output representation h. We then compute our main actor policy by applying a 128×7 linear layer followed by a soft-max non-linearity. The auxiliary actor is produced similarly but with separate parameters in its linear layer. Finally the critic's value is generated by applying a 128×1 linear layer to h.

MiniGrid model. Here we detail each component of the model architecture illustrated in Fig. 3c. The encoder ('Enc.') converts observation tensors (integer tensor of shape $7 \times 7 \times 3$) to a corresponding embedding tensor via three embedding sets (of length 8) corresponding to type, color, and state of the object. The observation tensor, which represents the 'lights-out' condition, has a unique (*i.e.*, different from the ones listed by (Chevalier-Boisvert et al., 2018b)) type, color and state. This prevents any type, color or state from having more than one connotation. The output of the encoder is of size $7 \times 7 \times 24$. This tensor is flattened and fed into a (single-layered) LSTM with a 128-dimensional hidden space. The output of the LSTM is fed to the main actor, auxiliary actor, and the critic. All of these are single linear layers with output size of $|\mathcal{A}|$, $|\mathcal{A}|$ and 1, respectively (main and auxiliary actors are followed by soft-max non-linearities).

A.8 Fair Hyperparameter Tuning

As discussed in the main paper, we consider two approaches for ensuring that comparisons to baselines are fair. In particular, we hope to avoid introducing misleading bias in our results by extensively tuning the hyperparameters (hps) of our ADVISOR methodology while leaving other methods relatively un-tuned.

2D-LH: Tune by Tuning a Competing Method. The goal of our experiments with the 2D-LH environment are, principally, to highlight that increasing the imitation gap can have a substantial detrimental impact on the quality of policies learned by training IL. Because of this, we wish to give IL the greatest opportunity to succeed and thus we are not, as in our other experiments, attempting to understand its expected performance when we must search for good hyperparameters. To this end, we perform the following procedure for every $i, j \in \{1, 3, 5 \ldots, 15\}$ with $i < j$.

For every learning rate $\lambda \in \{100 \text{ values evenly spaced in } [10^{-4}, 1] \text{ on a log-scale}\}$ we train a f^i-partial policy to imitate a f^j-optimal expert using BC. For each such trained policy, we roll out trajectories from the policy across 200 randomly sampled episodes (in the 2D-LH there is no distinction between training, validation, and test episodes as there are only four unique initial world settings). For each rollout, we compute the average cross entropy between the learned policy and the expert's policy at every step. A "best" learning rate $\lambda^{i,j}$ is then chosen by selecting the learning rate resulting in the smallest cross entropy (after having smoothed the results with a locally-linear regression model (Wasserman, 2006)).

A final learning rate is then chosen as the average of the $\lambda^{i,j}$ and this learning rate is then used when training all methods to produce the plots in Fig. 5. As some baselines require additional hyperparameter choices, these other hyperparameters were chosen heuristically (post-hoc experiments suggest that results for the other methods are fairly robust to these other hyperparameters).

All Other Tasks: Random Hyperparameter Evaluations. As described in the main paper, we follow the best practices suggested by Dodge et al. (2019). In particular, for all tasks (except for 2D-LH) we train each of our baselines when sampling that method's hyperparameters, see Table 1 and recall Sec. A.6, at random 50 times. Our plots, *e.g.*, Fig. 4, then report an estimate of the expected (validation set) performance of each of our methods when choosing the best performing model from a fixed number of random hyperparameter evaluations. Unlike (Dodge et al., 2019), we compute this estimate using a U-statistic (van der Vaart, 2000, Chapter 12) which is unbiased. Shaded regions encapsulate the 25-to-75th quantiles of the bootstrap distribution of this statistic.

Table 2: Structural and training hyperparameters.

Hyperparamter	**Value**
Structural	
Cell type embedding length	8
Cell color embedding length	8
Cell state embedding length	8
LSTM layers	1
LSTM hidden size	128
# Layers in critic	1
# Layers in actor	1
PPO	
Clip parameter (ϵ) (Schulman et al., 2017)	0.1
Decay on ϵ	$\texttt{Linear}(1, 0)$
# Processes to sample steps	20
Rollout timesteps	100
Minibatch size	1000
Epochs	4
Value loss coefficient	0.5
Discount factor (γ)	0.99
GAE parameter (λ)	1.0
Training	
Optimizer	Adam (Kingma & Ba, 2017)
(β_1, β_2) for Adam	$(0.9, 0.999)$
Learning rate	`searched`
Gradient clip norm	0.5
Training steps (WC/LC & variants)	$1 \cdot 10^6$
Training steps (2D-LH & PD)	$3 \cdot 10^5$

A.9 Training Implementation

A summary of the training hyperparameters and their values is included in Tab. 2. Kindly see (Schulman et al., 2017) for details on PPO and (Schulman et al., 2015b) for details on generalized advantage estimation (GAE).

Max. steps per episode. The maximum number of steps allowed in the 2D-LH task is 1000. Within the PD task, an agent can never take more than 11 steps in a single episode (1 action to select the door and then, at most, 10 more actions to input the combination if d_1 was selected) and thus we do not need to set a maximum number of allowed steps. The maximum steps allowed for an episode of WC/LC is set by (Chevalier-Boisvert et al., 2018b;a) to $4S^2$, where S is the grid size. We share the same limits for the challenging variants – Switch and Corrupt. Details of task variants, their grid size, and number of obstacles are included in Sec. A.5.

Reward structure. Within the 2D-LH task, the agent receives one of three possible rewards after every step: when the agent finds the goal it receives a reward of 0.99, if it otherwise has reached the maximum number of steps (1000) it receives a -1 reward, and otherwise, if neither of the prior cases hold, it obtains a reward of -0.01. See Sec. A.5.1 for a description of rewards for the PD task. For WC/LC, (Chevalier-Boisvert et al., 2018b;a) configure the environment to give a 0 reward unless the goal is reached. If the goal is reached, the reward is $1 - \frac{\text{episode length}}{\text{maximum steps}}$. We adopt the same reward structure for our Switch and Corrupt variants as well.

Computing infrastructure. As mentioned in Sec. 4.3, for all tasks (except LH) we train 50 models (with randomly sampled hps) for each baseline. This amounts to 650 models per task or 5850 models in total. For each task, we utilize a `g4dn.12xlarge` instance on AWS consisting of 4 NVIDIA T4 GPUs and 48 CPUs. We run through a queue of 650 models using 48 processes. For tasks set in the MiniGrid environments, models each require ≈ 0.9 GB GPU memory and all training completes in 18 to 24 hours. For the PD task, model memory footprints are smaller and training all 650 models is significantly faster (< 8 hours).

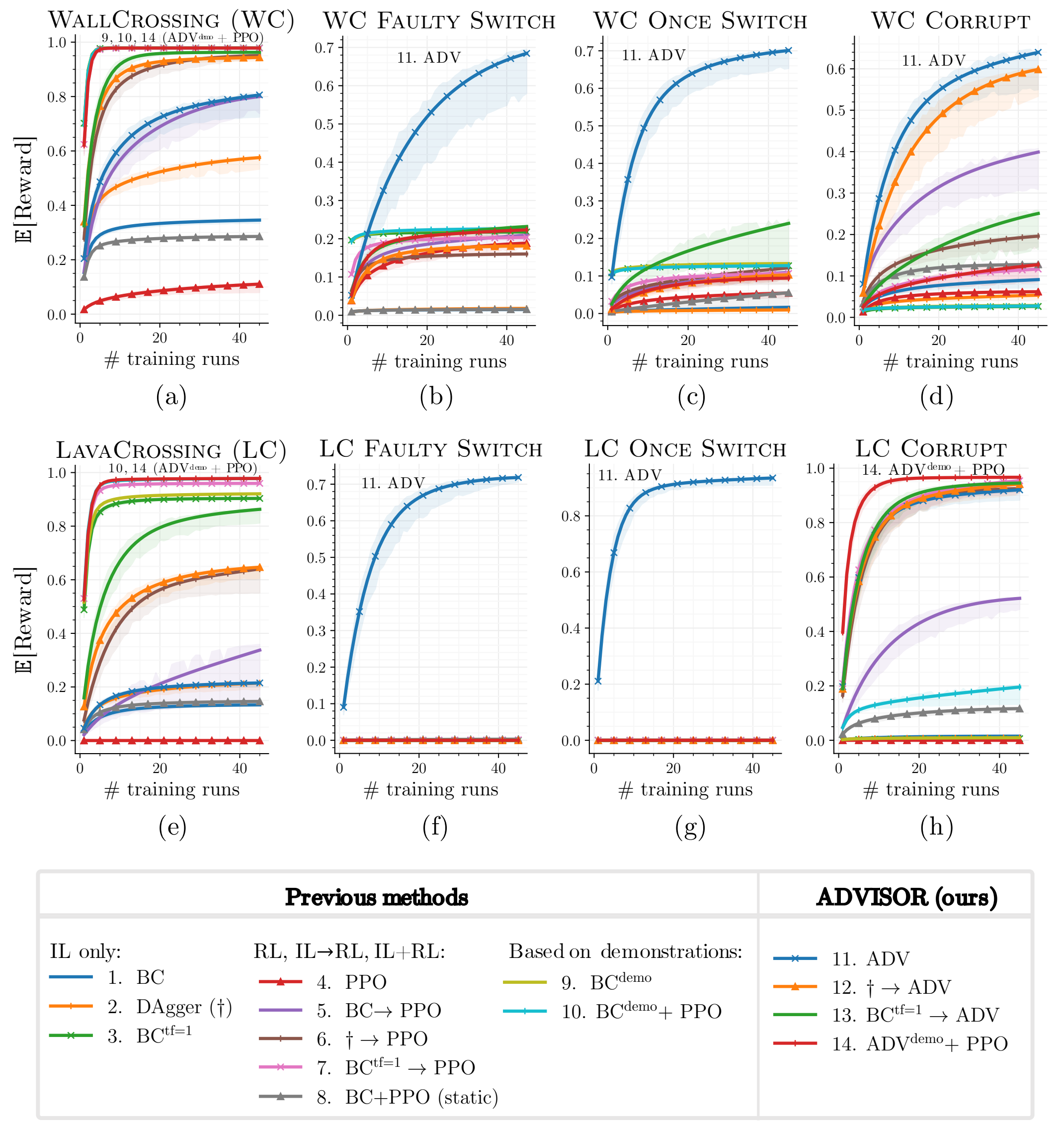

Figure 8: $\mathbb{E}$[**Reward**] **for baselines on MINIGRID tasks.** We include all variants of tasks considered in this work. Similar to Fig. 4, we plot estimates of the expected maximum validation set reward of all baselines (including our method), when allowing for increasingly many (random) hyperparameter evaluations (larger $\mathbb{E}$[Reward] with fewer evals. is better).

A.10 ADDITIONAL PLOTS

As mentioned in Sec. 4.3, we record three metrics for our tasks. Reward is the metric that best jointly captures success and effective path planning (see Sec. A.9 for reward structure). In the main paper, we included some reward plots in Fig. 4. Specifically, Fig. 8d, 8e, and 8g have already been included in the main paper (as Fig. 4b, 4c, and 4d). The remaining variants for WC/LC, FAULTY/ONCE SWITCH, and CORRUPT are presented in Fig. 8.

Success rate shows a similar trend, following from the definition of rewards, *i.e.*, agents which reach the target more often, mostly end up with higher rewards. In Fig. 9, we plot success rate for WC/LC, FAULTY/ONCE SWITCH, and CORRUPT tasks.

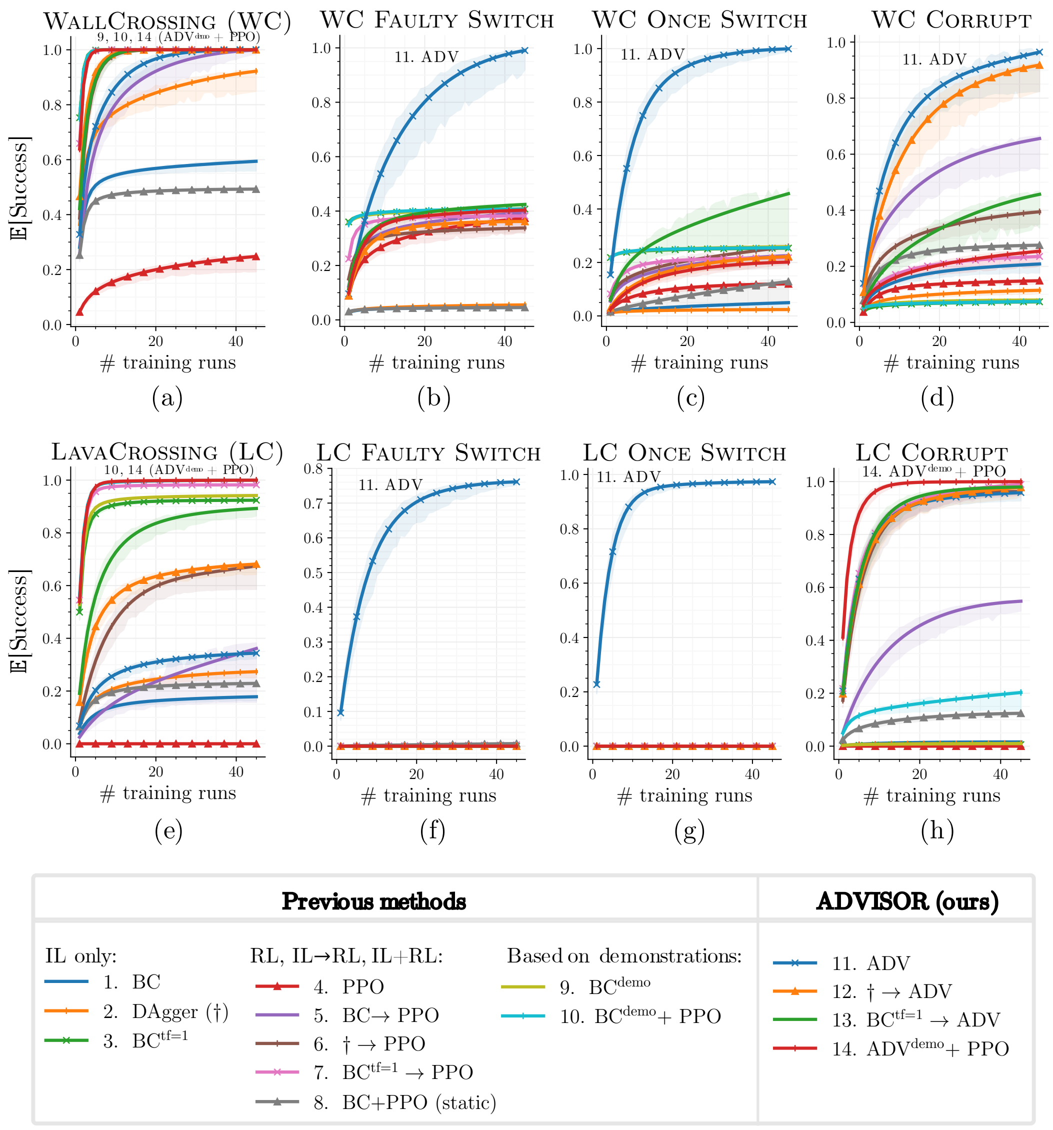

Figure 9: $\mathbb{E}$[**Success Rate**] **for baselines on MiniGrid tasks.** We include all variants of tasks considered in this work. This is the expected maximum validation set success rate of all baselines (including our method), when allowing for increasingly many (random) hyperparameter evaluations (larger $\mathbb{E}$[Success Rate] with fewer evals. is better).