Supplementary material

A Training procedure details

A.1 Algorithm details

We train a policy, critic and reward model simultaneously as shown in Algorithm 1. However, the reward model can be trained separately first, and then used to annotate datasets $\mathcal{D}_E$ and $\mathcal{D}_U$ with reward estimates. Both these approaches result in similar agent performance.

Algorithm 1 implements ORIL with PU learning (ORIL$_P$). The data augmentation is done as in [40]. The critic update is based on the discrepancy between the current estimate of the action-value and the TD-update (where the next step reward is estimated by reward model R_ψ). This discrepancy is measured by a divergence measure or a metric D, such as KL-divergence or squared error. In this paper, we use a distributional Q-function as in [37] and use the divergence measure proposed in [3].

Algorithm 1: Offline Reinforced Imitation Learning

Input: critic, policy and reward networks: Q_θ, π_ϕ and R_ψ, and its target versions: $Q_{\theta'}$, $\pi_{\phi'}$, $R_{\psi'}$, divergence measure D, expert dataset $\mathcal{D}_E$ and unlabeled dataset $\mathcal{D}_U$, hyperparameter η, number of updates $n_{updates}$;

Begin: Split $\mathcal{D}_U$ in half to get $\mathcal{D}_U^1$ and $\mathcal{D}_U^2$.
for $n_{updates}$ **do**
  `/* Reward learning */`
  Sample expert and unlabeled batches, i.e. $\mathcal{B}_E \subset \mathcal{D}_E$ and $\mathcal{B}_U \subset \mathcal{D}_U^1$;
  Augment images in $\mathcal{B}_E$ and $\mathcal{B}_U$ to obtain augmented batches $\overline{\mathcal{B}_E}$ and $\overline{\mathcal{B}_U}$;
  Train reward model using augmented batches $\overline{\mathcal{B}_E}$ and $\overline{\mathcal{B}_U}$:
    Update reward network parameters ψ with gradient:

$$-\nabla_\psi \eta \mathbb{E}_{s_t \sim \overline{\mathcal{B}_E}}[-\log(R_\psi(s_t))] + \mathbb{E}_{s_t \sim \overline{\mathcal{B}_U}}[-\log(1 - R_\psi(s_t))] - \eta \mathbb{E}_{s_t \sim \overline{\mathcal{B}_E}}[-\log(1 - R_\psi(s_t))]$$

  `/* Policy and critic learning */`
  Sample additional unlabeled batch from $\mathcal{D}_U^2$, i.e. $\mathcal{B}_U^2 \subset \mathcal{D}_U^2$;
  Concatenate (not augmented) expert and both unlabeled batches: $\mathcal{B} = \mathcal{B}_E \cup \mathcal{B}_U^1 \cup \mathcal{B}_U^2$;
  Apply CRR updates using rewards predicted by $R_{\psi'}$:
    Update critic network parameters θ with gradient:
      $-\nabla_\theta \mathbb{E}_{(s_t, a_t, s_{t+1}) \sim \mathcal{B}} D\big[Q_\theta(s_t, a_t), R_{\psi'}(s_{t+1}) + \gamma \mathbb{E}_{a \sim \pi_{\phi'}(s_{t+1})} Q_{\theta'}(s_{t+1}, a)\big]$;
    Update policy network parameters ϕ with gradient:
      $-\nabla_\phi \mathbb{E}_{(s_t, a_t) \sim \mathcal{B}} \log \pi_\phi(a_t|s_t) \mathbb{I}\left[Q_\theta(s_t, a_t) > Q_\theta(s_t, \pi_\phi(s_t))\right]$;
  `/* Target networks update */`
  Update the target networks every N steps by copying parameters: $\theta' \leftarrow \theta,\ \ \phi' \leftarrow \phi,\ \psi' \leftarrow \psi$;
end

A.2 Hyperparameters

We parametrize the critic, policy and reward models with neural networks. The policy and critic architectures are identical to the ones used in Critic Regularized Regression work [37], and the reward model architecture is inspired by the critic network. All networks are described in details below and presented in Figure 7.

We use the same hyperparameters as in [37]. We train all our models for $1e6$ learner steps and always with 3 seeds. We compute the mean return between $5e5$ and $1e6$ learner steps for each seed. This gives three values of mean returns (one per seed) for every agent and we report mean, and standard deviation across these results.

For pixel based tasks (i.e. Robotic Manipulation tasks), pixels are encoded with a residual CNN (see Figure 7, bottom). Two separate image encoders are trained. One of them is shared between critic and policy networks, and another is used (and trained) solely by the reward model.

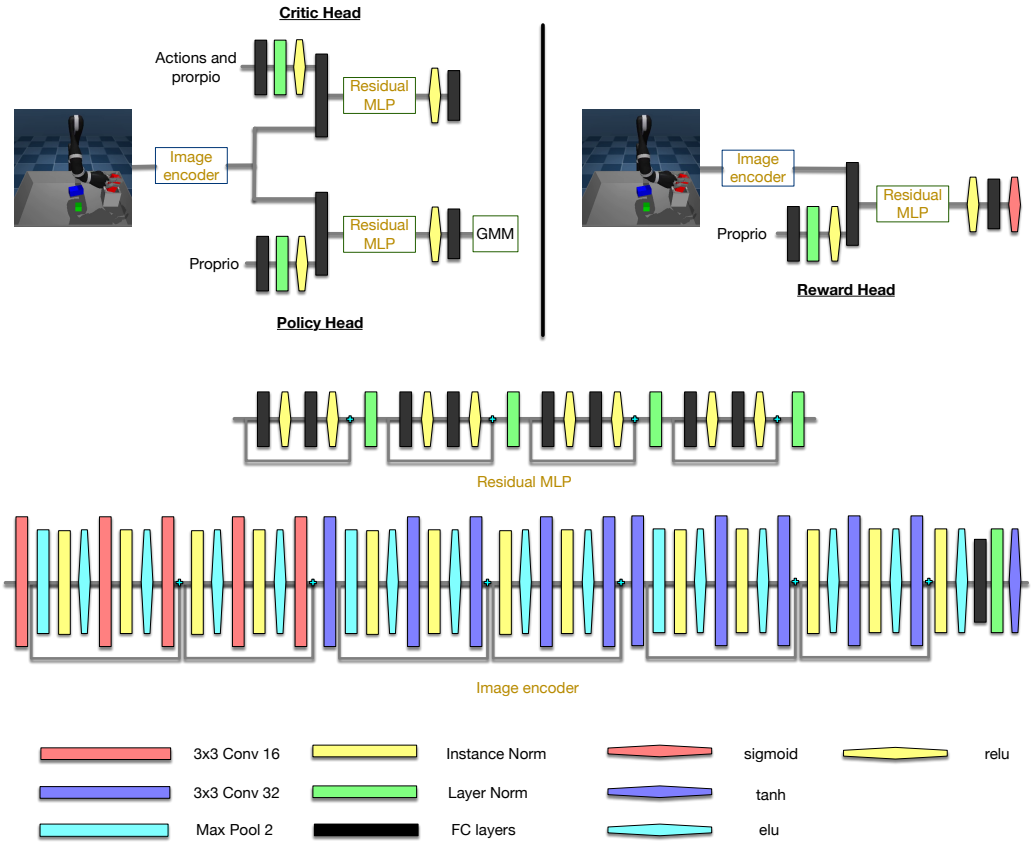

Figure 7: **The architectures used by the critic, policy and reward models.** There are two instances of the image encoder, one is shared between the critic and policy networks while the other is part of the reward network. All models implement the same residual MLP but it is never shared. GMM stands for Gaussian Mixture Model which outputs the final policy prediction (see details in the main text). This figure is based on the architecture figure from CRR paper [37].

All networks process proprio states (for critic, they are concatenated with actions) with one layer MLP to obtain preliminary representations which are then concatenated with image representations (for pixel based tasks) and further processed. The final layers depend on the network purpose (see details below).

Policy head Proprioceptive input is processed with a fully connected layer, layer normalization and tanh activation function, then concatenated with pixel encoding (when present) and passed into a residual MLP (see Figure 7, top left). The output of the MLP defines the policy which is a mixture of five multivariate Gaussians. Specifically, the output of the MLP is contains: five mean vectors; five vectors that (after passing through a softplus function) define diagonal of the covariance matrix of each Gaussian; and five scalars that define mixture log-probabilities.

Critic head Proprioceptive input concatenated with actions is processed through a fully connected layer, layer normalization and tanh, then concatenated with pixel encoding (when present) and passed into a residual MLP (see Figure 7, top left). The output of the MLP defines a discrete distribution of the distributional critic.

Reward head Proprioceptive input is processed through a fully connected layer, layer normalization and tanh, then concatenated with pixel encoding (when present) and passed into a residual MLP (see Figure 7, top right). The output is proccesed by a fully connected layer to obtain a scalar which is then scaled to $(0, 1)$ by applying sigmoid.

B Positive-unlabeled learning

If we treat reward learning as a binary decision problem to distinguish success ($\mathcal{D}_E$) and failure ($\mathcal{D}_F$) trajectories, we can write the loss for the reward model R as

$$\eta \mathbb{E}_{s_t \sim \mathcal{D}_E}[-\log(R(s_t))] + (1-\eta)\mathbb{E}_{s_t \sim \mathcal{D}_F}[-\log(1-R(s_t))], \tag{3}$$

where η is the proportion of the trajectory space corresponding to success. This corresponds to a more general form of Equation 1 where before $\eta = 0.5$ was assumed.

The key insight of PU-learning [8, 7] is to observe that the second term in the loss (computed for $\mathcal{D}_F$) can be written in terms of a dataset of unlabeled trajectories $\mathcal{D}_U$ (that contains an unknown mixture of successes and failures) and the dataset of positive trajectories $\mathcal{D}_E$:

$$(1-\eta)\mathbb{E}_{s_t \sim \mathcal{D}_F}[-\log(1-R(s_t))] = \mathbb{E}_{s_t \sim \mathcal{D}_U}[-\log(1-R(s_t))] - \eta \mathbb{E}_{s_t \sim \mathcal{D}_E}[-\log(1-R(s_t))]. \tag{4}$$

This allows the loss for the reward model to be rewritten in a way that avoids any dependence on explicitly labeled failures

$$\eta \mathbb{E}_{s_t \sim \mathcal{D}_E}[-\log(R(s_t))] + \mathbb{E}_{s_t \sim \mathcal{D}_U}[-\log(1-R(s_t))] - \eta \mathbb{E}_{s_t \sim \mathcal{D}_E}[-\log(1-R(s_t))] \tag{5}$$

In this paper we treat η as a hyperparameter and set it to $\eta = 0.5$ throughout.

C Task-relevant adversarial imitation learning

TRAIL proposes to constrain the GAIL discriminator such that it is *not* able to distinguish between certain, preselected expert and agent observations which do not contain task behavior. For example, the discriminator should not be able to distinguish whether early observations at the very start of an episode come from the demonstration or unlabeled set, since no meaningful behavior has yet been performed.

We adapt TRAIL to the offline setting and use early observations to form our constraint sets. Specifically, we construct subsets of early states, i.e. $\mathcal{D}'_U = \{s_t \in \mathcal{D}_U \mid t < 10\}$, and analogously $\mathcal{D}'_E$ for $\mathcal{D}_E$. We compute reward loss as in Equation 1 for the original datasets and also, separately, for the states from the constraint sets. The loss optimized by the reward model is the following:

$$L_\psi(\mathcal{D}_E, \mathcal{D}_U) - \mathbf{1}_{R_\psi(\mathcal{D}'_U) > R_\psi(\mathcal{D}'_E)} L_\psi(\mathcal{D}'_E, \mathcal{D}'_U), \tag{6}$$

where $\mathbf{1}_{R_\psi(\mathcal{D}'_E) > R_\psi(\mathcal{D}'_U)}$ is one if average prediction R_ψ for expert early observations ($\mathcal{D}'_E$) is higher than for unlabeled early observations ($\mathcal{D}'_U$). Intuitively, we use early observations to first control if the reward model is overfitting, and if so, we use them again to compute reversed loss to regularize discriminator. We refer to the original paper [40] for motivation and detailed description.

The original paper [40] proposes another indicator for applying reversed loss, but we find the average prediction for early observations used by us to work similarly well.