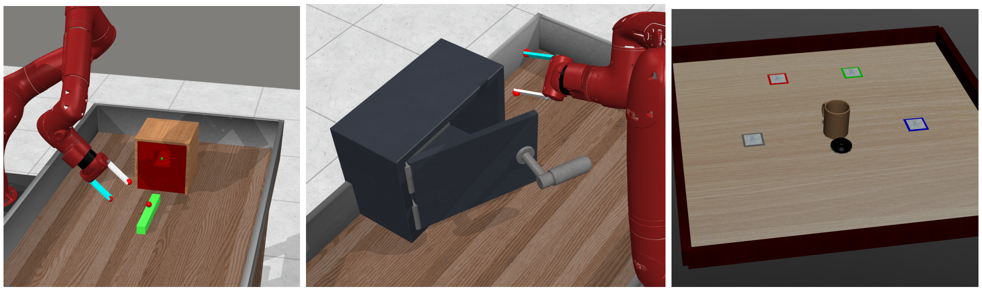

Figure 6: Environments from the EARL benchmark [5] used for simulated experiments. From left to right, the environments are: Peg insertion, Door closing and Tabletop organization.

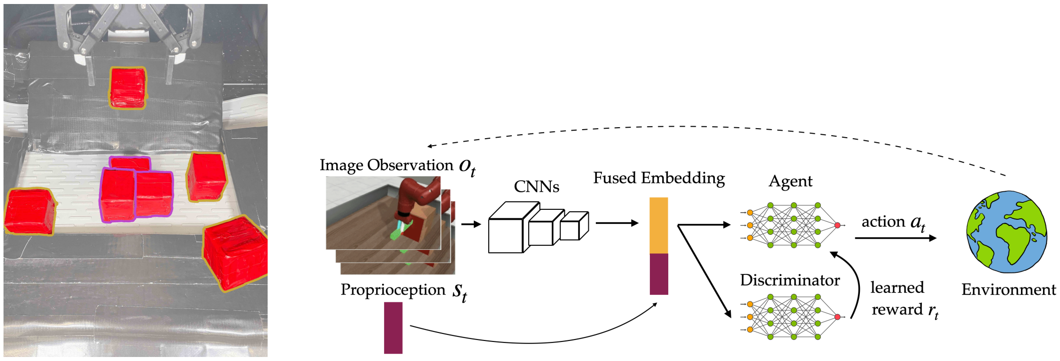

Figure 7: (*left*) Randomized position of the cube in the grasping task. The position marked by violet boundary are within the distribution of expert demonstrations, and the rest are outside the distribution. (*right*) Architecture overview for MEDAL++.

A Appendix

A.1 Algorithm Overview

The pseudocode for training is given in Algorithm 1. First, the parameters and data buffers in $\mathcal{F}$ and $\mathcal{B}$ are initialized and the forward and backward demonstrations are loaded into $\mathcal{D}_f^*$ and $\mathcal{D}_b^*$ respectively. Next, we update the forward and backward goal sets, as described above. After initializing the environment, the forward policy π_f interacts with the environment and collects data, updating the networks and buffers in $\mathcal{F}$. The control switches over to the backward policy π_b after a fixed number of steps, and the networks and buffers in $\mathcal{B}$ are updated. The backward policy interacts for a fixed number of steps, after which the control is switched over to the forward policy and this cycle is repeated thereon. When executing in the real world, humans are allowed to intervene and reset the environment intermittently, switching the control over to π_f after the intervention to restart the forward-backward cycle.

We now expand on how the networks are updated for π_f during training (also visualized in Figure 9); the updates for π_b are analogous. First, the new transition in the environment is added to $\mathcal{D}_f$. Next, we sample a batch of states from $\mathcal{D}_f$ and label them 0, and sample a batch of *equal size* from $\mathcal{D}_f^*$ and label them 1. The classifier C_f is updated using gradient descent on the combined batch to minimize the cross-entropy loss. Note, the classifier is not updated for every step collected in the environment. As stated earlier, the classification problem is easier than learning the policy, and therefore, it helps to train the classifier slower than the policy. Finally, the policy π_f, Q-value networks $\{Q_n^f, \bar{Q}_n^f\}_{n=1}^N$ and the encoder $\mathcal{E}$ are updated on a batch of transitions constructed by sampling $(1-\rho)B$ transitions from $\mathcal{D}_f$ and ρB transitions from $\mathcal{D}_f^*$. The Q-value networks and the encoder are updated by minimizing $\frac{1}{N}\sum_{n=1}^N \ell(Q_n, \mathcal{E})$ (Eq 1), and the target Q-networks are updated as an exponential moving average of Q-value networks. The policy π_f is updated by maximizing $\mathcal{L}(\pi)$. We update

Algorithm 1: MEDAL++

```
initialize $\mathcal{F}, \mathcal{B}$; // forward, backward parameters
$\mathcal{F}.\mathcal{D}_f^*, \mathcal{B}.\mathcal{D}_b^* \leftarrow$ load_demonstrations()
$\mathcal{F}.\mathcal{G}_f \leftarrow$ get_states($\mathcal{F}.\mathcal{D}_f^*, -K$:) // last $K$ states
// exclude last $K$ states from $\mathcal{D}_f^*$, use only the last $K$ states from $\mathcal{D}_b^*$
$\mathcal{B}.\mathcal{G}_b \leftarrow$ get_states($\mathcal{F}.\mathcal{D}_f^*$, :$-K$) $\cup$ get_states($\mathcal{B}.\mathcal{D}_b^*, -K$:)
$s \sim \rho_0$; $\mathcal{A} \leftarrow \mathcal{F}$; // initialize environment
while not done do
    $a \sim \mathcal{A}$.act($s$); $s' \sim \mathcal{T}(\cdot \mid s, a)$;
    $\mathcal{A}$.update_buffer($\{s, a, s'\}$);
    $\mathcal{A}$.update_classifier();
    $\mathcal{A}$.update_parameters();
    // switch policy after a fixed interval
    if switch then
        switch($\mathcal{A}, (\mathcal{F}, \mathcal{B})$);
    // allow intermittent human interventions
    if interrupt then
        $s \sim \rho_0$;
        $\mathcal{A} \leftarrow \mathcal{F}$;
    else
        $s \leftarrow s'$;
```

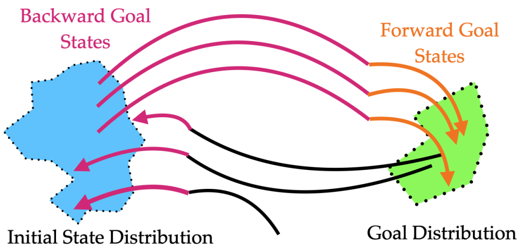

Figure 8: Visualizing the positive target states for forward classifier C_f and backward classifier C_b from the expert demonstrations. For forward demonstrations, last K states are used for C_f (*orange*) and the rest are used for C_b (*pink*). For backward demonstrations, last K states are used for C_b.

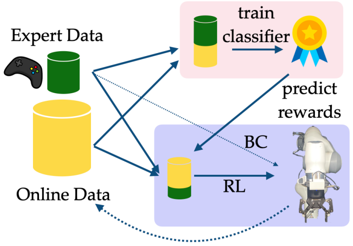

Figure 9: An overview of MEDAL++ training. The classifier is trained to discriminate states visited by an expert from the states visited online. The robot reinforcement learns on a combination of self-collected and expert transitions, and the policy learning is regularized using the behavior cloning loss.

the Q-value networks multiple times for every step collected in the environment, whereas the policy network is updated once for every step collected in the environment [15].

A.2 Implementation Details and Practical Tips

An overview of the architecture used by the forward and backward networks is shown in Figure 7.

Visual Encoder: For the encoder, we use the same architecture as DrQ-v2 [12]: 4 convolutional layers with 32 filters of size (3, 3), stride 1, followed by ReLU non-linearities. The high-dimensional output from the CNN is embedded into a 50 dimensional feature using a fully-connected layer, followed by LayerNorm and tanh non-linearity (to output the features normalized to $[-1, 1]$). For real-robot experiments, the first person and third person views are concatenated channel wise before being passed into the encoder. The output of the encoder is fused with proprioceptive information, in this case, the end-effector position, before being passed to actor and critic networks.

Actor and Critic Networks: Both actor and critic networks are parameterized as 4 layer fully-connected networks with 1024 ReLU hidden units for every layer. The actor parameterizes a Gaussian distribution over the actions, where a tanh non-linearity on the output restricts the actions to $[-1, 1]$. We use an ensemble size of 10 critics.

Discriminators: The discriminator for the forward and backward policies use a similar visual encoder but with 2 layers instead of 4. The visual embedding is passed to a fully connected network with 2 hidden layers with 256 ReLU units. When training the network, we use mixup and spectral norm regularization [53, 52] for the entire network.

Training Hyperparameters: For all our experiments, $K = 20$, i.e. the number of frames used as goal frames. The forward policy interacts with the environment for 200 steps, then the backward policy interacts for 200 steps. In real world experiments, we also reset the arm every 1000 steps to avoid hitting singular positions. Note, this reset does not require any human intervention as the controller just resets the arm to a fixed joint position. We use a batch size of 256 to train the policy and critic networks, out of which 64 transitions are sampled from the demonstrations (*oversampling*). We use a batch size of 512 to train the discriminators, 256 of the states come from expert data and the other 256 comes from the online data. Further, the discriminators are updated every 1000 steps collected in the environment. The update-to-data ratio, that is the number of gradient updates per transition collected in the environment is 3 for simulated environments and 1 for the real-robot experiments. We use a linearly decaying schedule for behavior cloning regularization from 1 to 0.1 over the first 50000 steps which remains fixed at 0.1 onwards throughout training.

For real world experiments, we use a wrist camera to improve the overall performance [58], and provide **only** the wrist-camera view to both discriminators. We find that this further regularizes the discriminator. Finally, we provide no proprioceptive information for the VICE discriminator, but we give MEDAL discriminator the proprioceptive information, as it needs a stronger notion of the robot's localization to adequately reset to a varied number of initial positions for improved robustness.

Teleoperation: To collect our demonstrations on the real robot, we use an Xbox controller that manipulates the end-effector position, orientation and the gripper state. Two salient notes: (1) The forward and backward demonstrations are collected together, one after the other and (2) the initial position for demonstrations is randomized to cover as large a state-space as feasible. The increased coverage helps with exploration during autonomous training.

A.3 EARL Environments, Training and Evaluation

Environments. We consider three sparse-reward continuous-control environments from EARL benchmark [5], shown in Appendix, Fig 6). *Tabletop organization* is a simplified manipulation environment where a gripper is tasked to move the mug to one of the four coasters from a wide set of initial states, *sawyer door closing* task requires a sawyer robot arm to learn how to close a door starting from various positions, and finally the *sawyer peg insertion* task requires the sawyer robot arm to grasp the peg and insert it into a goal. Not only does the robot have to learn how to do the task (i.e. close the door or insert the peg), but it has to learn how to undo the task (i.e. open the door or remove the peg) to try task repeatedly in the non-episodic training environment. The sparse reward function is given by $r(s, a) = \mathbb{1}(\|s - g\| \leq \epsilon)$, where g denotes the goal, and ϵ is the tolerance for the task to be considered completed.

Training and Evaluation. The environments are setup to return 84×84 RGB images as observations with a 3-dimensional action space for the *tabletop organization* (2D end-effector deltas in the XY plane and 1D for gripper) and a 4-dimensional action space for *sawyer* environments (3D end-effector delta control + 1D gripper). The training environment is reset to $s_0 \sim \rho_0$ every 25,000 steps of interaction with the environment. This is extremely infrequent compared to episodic settings where the environment is reset to the initial state distribution every 200-1000 steps. EARL comes with 5-15 forward and backward demonstrations for every environment to help with exploration in these sparse reward environments. We evaluate the forward policy every $10,000$ training steps, where the evaluation approximates $\mathbb{E}_{s_0 \sim \rho_0}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$ by averaging the return of the

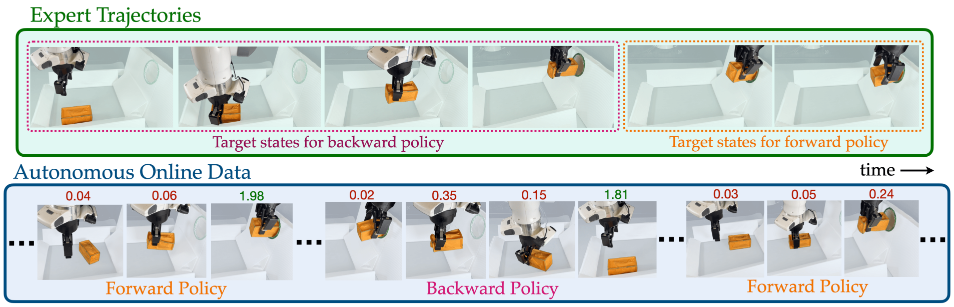

Figure 10: An overview of MEDAL++ on the task of inserting the peg into the goal location. (*top*) Starting with a set of expert trajectories, MEDAL++ learns a forward policy to *insert the peg* by matching the goal states and a backward policy to *remove and randomize the peg position* by matching the rest of the states visited by an expert. (*bottom*) Chaining the rollouts of forward and backward policies allows the robot to practice the task autonomously. The rewards indicate the similarity to their respective target states, output by a discriminator trained to classify online states from expert states.

policy over 10 episodes starting from $s_0 \sim \rho_0$. These roll-outs are used only for evaluation, and not for training.

A.4 Real-world Experiment Analysis

We discuss the four manipulation tasks in detail. We recommend viewing the supplemental website for training and evaluation videos:

(1) *Cube Grasping*: The goal in this task is to grasp the cube from varying initial positions and configurations and raise it. For this task, we consider a controlled setting to isolate one potential source of improvement from autonomous reinforcement learning: robustness to the initial state distribution. Specifically, all the forward demonstrations are collected starting from a narrow set of initial states (*ID*), but, the robot is evaluated starting from both ID states and out-of-distribution (*OOD*) states, visualized in Appendix, Figure 7. BC policy is competent on ID states, but it performs poorly on states that are OOD. However, after autonomous self-improvement using MEDAL++, we see an improvement of 15% on ID performance, and a large improvement of 74% on OOD performance. Autonomous training allows the robot to practice the task from a diverse set of states, including states that were OOD relative to the demonstration data. This suggests that improvement in success rate results partly from being robust to the initial state distribution, as a small set of demonstrations is unlikely to cover all possible initial states a robot can be evaluated from.

(2) *Cloth on the Hook*: In this task, the robot is tasked with grasping the cloth and putting it through a fixed hook. To practice the task repeatedly, the backward policy has to remove the cloth from the hook and drop it on platform. Here, MEDAL++ improves the success rate over BC by 36%. The BC policy has several failure modes: (1) it fails to grasp the cloth, (2) it follows through with hooking because of memorization, or (3) it hits into into the hook because it drifts from the right trajectory and could not recover. Autonomous self-improvement improves upon all these issues, but particularly, it learns to re-try grasping the cloth if it fails the first time, rather than following a memorized trajectory observed in the forward demonstrations.

(3) *Bowl Covering with Cloth*: The goal of this task is to cover a bowl entirely using the cloth. The cloth can be a wide variety of initial states, ranging from 'laid out flat' to 'scrunched up' in varying locations. The task is challenging as the robot has to grasp the cloth at the correct location to successfully cover the entire bowl (partial coverage is counted as a failure). Here, MEDAL++ improves the performance over BC by 34%. The failure modes of BC are similar to previous task, including failure to grasp, memorization and failure to re-try, and incomplete coverage due to wrong initial grasp. Autonomous self-improvement substantially helps with the grasping (including re-trying) and issues related to memorization. While it plans the grasps better than BC, there is room for improvement to reduce failures resulting from partially covering the bowl.

(4) *Peg Insertion*: Finally, we consider the task of inserting a peg into a goal location. The location and orientation of the peg is randomized, in service of which we use 5DoF control for this task. A successful insertion requires the toy to be perpendicular to the goal before insertion, and the error

margin for a successful insertion is small given the size of the peg and the goal. Additionally, the peg here is a soft toy, it can be grasped while being in the wrong orientation. Here, MEDAL++ improves the performance by 48% over BC. In addition to failures described in the previous tasks, a common cause of failure is the insertion itself where the agent takes an imprecise trajectory and is unable to insert the peg. After autonomous self-improvement, the robot employs an interesting strategy where it keeps retries the insertion till it succeeds. The policy is also better at grasping, though the failures of insertion often result from orienting the gripper incorrectly before the grasp which makes insertion infeasible.