7 Appendix

In Appendix 7.1, we discuss algorithmic details for ThriftyDAgger and all comparisons. Then, Appendix 7.2 discusses implementation and hyperparameter details for all algorithms. In Appendix 7.3, we provide additional details about the simulation and physical experiment domains, and in Appendix 7.4, we describe the protocol and detailed results from the conducted user study.

7.1 Algorithm Details

Here we provide a detailed algorithmic description of ThriftyDAgger and all comparisons.

7.1.1 ThriftyDAgger

The full pseudocode for ThriftyDAgger is provided in Algorithm 1. ThriftyDAgger first initializes π_r via Behavior Cloning on offline transitions ($\mathcal{D}_h$ from the human supervisor, π_h) (line 1-2). Then, π_r collects an initial offline dataset $\mathcal{D}_r$ from the resulting π_r, initializes $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ by optimizing Equation (5) on $\mathcal{D}_r \cup \mathcal{D}_h$, and initializes parameters $\beta_h, \beta_r, \delta_h$, and δ_r as in Section 4.4 (lines 3-5). We then collect data for N episodes, each with up to T timesteps. In each timestep of each episode, we determine whether robot policy π_r or human supervisor π_h should be in control using the procedure in Section 4.3 (lines 10-20). Transitions in autonomous mode are aggregated into $\mathcal{D}_r$ while transitions in supervisor mode are aggregated into $\mathcal{D}_h$. Episodes are terminated either when the robot reaches a valid goal state or has exhausted the time horizon T. At this point, we re-initialize the policy to autonomous mode and update parameters $\beta_h, \beta_r, \delta_h$, and δ_r as in Section 4.4 (lines 21-23). After each episode, π_r is updated via supervised learning on $\mathcal{D}_h$, while $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ is updated on $\mathcal{D}_r \cup \mathcal{D}_h$ to reflect the task success probability of the resulting π_r (lines 24-26).

7.1.2 Behavior Cloning

We train policy π_r via direct supervised learning with a mean-squared loss to predict reference control actions given a dataset of (state, action) tuples. Behavior Cloning is trained only on full expert demonstrations collected offline from π_h and is not allowed access to online interventions. Thus, Behavior Cloning is trained only on dataset $\mathcal{D}_h$ (line 1, Algorithm 1) and the policy is frozen thereafter. In our simulation experiments, Behavior Cloning is given 50% more offline data than the other algorithms for a more fair comparison, such that the amount of additional offline data is approximately equal to the average amount of online data provided to the other algorithms.

7.1.3 SafeDAgger

SafeDAgger [19] is an interactive imitation learning algorithm which selects between autonomous and supervisor mode using a classifier f that discriminates between "safe" states, for which π_r's proposed action is within some threshold β_h of that proposed by supervisor policy π_h, and "unsafe" states, for which this action discrepancy exceeds β_h. SafeDAgger learns this classifier using dataset $\mathcal{D}_h$ from Algorithm 1, and updates f online as $\mathcal{D}_h$ is expanded through human interventions. During policy rollouts, if f marks a state as safe, the robot policy is executed (autonomous mode), while if f marks a state as unsafe, the supervisor is queried for an action. While this approach can be effective in some domains [19], prior work [18] suggests that this intervention criterion can lead to excessive context switches between the robot and supervisor, and thus impose significant burden on a human supervisor. As in ThriftyDAgger and other DAgger [12] variants, SafeDAgger updates π_r on an aggregated dataset of all transitions collected by the supervisor (analogous to $\mathcal{D}_h$ in Algorithm 1).

7.1.4 LazyDAgger

LazyDAgger [18] builds on SafeDAgger [19] and trains the same action discrepancy classifier f to determine whether the robot and supervisor policies will significantly diverge at a given state. However, LazyDAgger introduces a few modifications to SafeDAgger which lead to lengthier and more informative interventions in practice. First, LazyDAgger observes that when the supervisor has control of the system (supervisor mode), querying f for estimated action discrepancy is no longer necessary since we can simply query the robot policy at any state during supervisor mode to obtain a true measure of the action discrepancy between the robot and supervisor policies. This prevents exploiting approximation errors in f when the supervisor is in control. Second, LazyDAgger introduces an asymmetric switching condition between autonomous and supervisor control, where switches are executed from autonomous to supervisor mode if f indicates that the predicted action discrepancy is above β_h, but switches are only executed from supervisor mode back to autonomous

Algorithm 1 ThriftyDAgger

Require: Number of episodes N, time horizon T, supervisor policy π_h, desired context switching rate α_h

Collect offline dataset $\mathcal{D}_h$ of (s, a^h) tuples with π_h
Initialize π_r via Behavior Cloning on $\mathcal{D}_h$
Collect offline dataset $\mathcal{D}_r$ of (s, a^r) tuples with π_r
Initialize $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ by optimizing Equation (4) on $\mathcal{D}_r \cup \mathcal{D}_h$
Optimize $\beta_h, \beta_r, \delta_h, \delta_r$ on $\mathcal{D}_h$ `#Online tuning based on` α_h `(Section 4.4)`
for $i \in \{1, \dots N\}$ **do**
    Initialize s_0, Mode $\leftarrow$ Autonomous
    for $t \in \{1, \dots T\}$ **do**
        $a^r_t = \pi_r(s_t)$
        if Mode = Supervisor or Intervene(s_t, δ_h, β_h) **then** `#Determine control mode (Section 4.3)`
            $a^h_t = \pi_h(s_t)$
            $\mathcal{D}_h \leftarrow \mathcal{D}_h \cup \{(s_t, a^h_t)\}$
            Execute a^h_t
            if Cede(s_t, δ_r, β_r) **then** `#Default control mode for next timestep (Section 4.3)`
                Mode $\leftarrow$ Autonomous
            else
                Mode $\leftarrow$ Supervisor
        else
            Execute a^r_t
            $\mathcal{D}_r \leftarrow \mathcal{D}_r \cup \{(s_t, a_t)\}$
        if Terminal state reached **then**
            Exit Loop, Mode $\leftarrow$ Autonomous
            Recompute $\beta_h, \beta_r, \delta_h$ `#Online tuning based on` α_h `(Section 4.4)`
    $\pi_r \leftarrow \arg\min_{\pi_r} \mathbb{E}_{(s_t, a^h_t) \sim \mathcal{D}_h} [\mathcal{L}(\pi_r(s_t), \pi_h(s_t))]$
    Collect $\mathcal{D}_r$ offline with robot policy π_r
    Update $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ on $\mathcal{D}_r \cup \mathcal{D}_h$ `#Update Q-function via Equation (6)`

mode if the true action discrepancy is below some value $\beta_r < \beta_h$. This encourages lengthier interventions, leading to fewer context switches between autonomous and supervisor modes. Finally, LazyDAgger injects noise into supervisor actions in order to spread the distribution of states in which reference controls from the supervisor are available. ThriftyDAgger builds on the asymmetric switching criterion introduced by LazyDAgger, but introduces a new switching criterion based on the estimated task success probability, which we found significantly improved performance in practice.

7.1.5 HG-DAgger

Unlike SafeDAgger, LazyDAgger, and ThriftyDAgger, which are robot-gated and autonomously determine when to solicit intervention requests, HG-DAgger is human-gated, and thus requires that the supervisor determine the timing and length of interventions. As in ThriftyDAgger, HG-DAgger updates π_r on an aggregated dataset of all transitions collected by the supervisor (analogous to $\mathcal{D}_h$ in Algorithm 1).

7.2 Hyperparameter and Implementation Details

Here we provide a detailed overview of all hyperparameter and implementation details for ThriftyDAgger and all comparisons to facilitate reproduction of all experiments. We also include code in the supplement, and will release a full open-source codebase after anonymous review.

7.2.1 ThriftyDAgger

Peg Insertion (Simulation): We initially populate $\mathcal{D}_h$ with 2,687 offline transitions, which correspond to 30 task demonstrations collected by an expert human supervisor, to initialize the robot policy π_r. We represent π_r with an ensemble of 5 neural networks, trained on bootstrapped samples of data from $\mathcal{D}_h$ in order to quantify uncertainty for novelty estimation. Each neural network is trained using the Adam Optimizer (learning rate $1\text{e}{-3}$) with 5 training epochs, 500 gradient steps in each training epoch, and a batch size of 100. All networks consist of 2 hidden layers, each with 256 hidden units with ReLU activations, and a Tanh output activation.

The Q-function used for risk-estimation, $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$, is trained with a batch size of 50, and batches are balanced such that 10% of all sampled transitions contain a state in the goal set. We train $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ with the Adam Optimizer, with a learning rate of $1\mathrm{e}{-3}$ and discount factor $\gamma = 0.9999$. In order to train $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$, we collect 10 test episodes from π_r every 2,000 environment steps. We represent $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ with a 2 hidden layer neural net in which each hidden layer has 256 hidden units with ReLU activations and with a sigmoid output activation. The state and action are concatenated before they are fed into $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$.

Block Stacking (Simulation): This is an additional simulation environment not included in the main text. Results and a description of the task are in Section 7.3.2. We populate $\mathcal{D}_h$ with 1,677 offline transitions, corresponding to 30 task demonstrations, to initialize π_r. All other parameters and implementation details are identical to the peg insertion environment.

Cable Routing (Physical): We initially populate $\mathcal{D}_h$ with 1,381 offline transitions, corresponding to 25 task demonstrations collected by an expert human supervisor, to initialize the robot policy π_r. We again represent π_r with an ensemble of 5 neural networks, trained on bootstrapped samples of data from $\mathcal{D}_h$ in order to quantify uncertainty for novelty estimation. Each neural network is trained using the Adam Optimizer (learning rate $2.5\mathrm{e}{-4}$) with 5 training epochs, 300 gradient steps per training epoch, and a batch size of 64. All networks consist of 5 convolutional layers (format: $(\text{in\_channels}, \text{out\_channels}, \text{kernel\_size}, \text{stride})$): $[(3, 24, 5, 2), (24, 36, 5, 2), (36, 48, 5, 2), (48, 64, 3, 1), (64, 64, 3, 1)]$ followed by 4 fully connected layers (format: $(\text{in\_units}, \text{out\_units})$): $[(64, 100), (100, 50), (50, 10), (10, 2)]$. Here we utilize ELU (exponential linear unit) activations with a Tanh output activation.

The Q-function used for risk-estimation, $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$, is trained with a batch size of 64 as well, and batches are balanced such that 10% of all sampled transitions contain a state in the goal set. We train $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ with the Adam Optimizer with a learning rate of $2.5\mathrm{e}{-4}$ and discount factor $\gamma = 0.9999$. In order to train $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$, we collect 5 test episodes from π_r every 500 environment steps. We represent $\hat{Q}^{\pi_r}_{\phi,\mathcal{G}}$ with a neural network with the same 5 convolutional layers as the policy networks above, but with the fully connected layers as follows (format: $(\text{in\_units}, \text{out\_units})$): $[(64{+}2, 100), (100, 50), (50, 10), (10, 1)]$. We concatenate the action with the state embedding resulting from the 5 convolutional layers (hence the 64 + 2) and feed the resulting concatenated embedding into the 4 fully connected layers above. We utilize ELU (exponential linear unit) activations with a sigmoid output activation.

7.2.2 Behavior Cloning

Peg Insertion (Simulation): For Behavior Cloning, we initially populate $\mathcal{D}_h$ with 4,004 offline transitions, corresponding to 45 task demonstrations collected by an expert human supervisor, to initialize the robot policy π_r (note that this is more transitions than are provided to ThriftyDAgger). All other details are the same as ThriftyDAgger for training π_r.

Block Stacking (Simulation): We initially populate $\mathcal{D}_h$ with 3,532 offline transitions, corresponding to 60 task demonstrations, to initialize π_r. Note that Behavior Cloning has access to twice as many offline demonstrations as the other algorithms.

Cable Routing (Physical): We train π_r with the same architecture and procedure as for ThriftyDAgger, but only on the initial offline data.

7.2.3 SafeDAgger

We use the same hyperparameters and architecture for training π_r as for ThriftyDAgger. Unlike ThriftyDAgger, SafeDAgger does not have a mechanism to automatically set intervention thresholds when provided an intervention budget α_h. Thus, we must specify a value for the switching threshold β_h. We use $\beta_h = 0.008$, since this is recommended in [19] as the value which was found to work well in experiments (in practice, this value marks about 20% of states as "unsafe").

7.2.4 LazyDAgger

We use the same hyperparameters and architecture for training π_r as for ThriftyDAgger. Unlike ThriftyDAgger, LazyDAgger does not have a mechanism to automatically set intervention thresholds when provided an intervention budget α_h. Thus, we must specify a value for both switching thresholds

Table 4: **Peg Insertion in Simulation Additional Metrics:** We report additional statistics for the peg insertion task: total number of interventions (T Ints), total number of offline and online human actions (T Acts (H)), and total number of robot actions (T Acts (R))) at training time across all trajectories (successful and unsuccessful). We report these same metrics at execution time, but T Acts (H) does not include offline human actions, as at execution time it does not refer to the number of training samples for the robot policy. We also report the success rate of the mixed control policy at *training* time (Train Succ.). Results suggest that ThriftyDAgger solicits fewer interventions than prior algorithms at training time while achieving a comparable success rate and throughput to HG-DAgger. At execution time, ThriftyDAgger collects lengthier interventions than prior algorithms but succeeds more often at the task (Table 1).

Algorithm	Training Interventions			Train Succ.	Execution Interventions		
	T Ints	T Acts (H)	T Acts (R)		T Ints	T Acts (H)	T Acts (R)
Behavior Cloning	N/A	4004	N/A	N/A	N/A	N/A	N/A
SafeDAgger	334	4227	8460	48/73	81	396	1781
LazyDAgger	82	3683	9004	37/67	30	290	2422
HG-DAgger	124	4392	8295	83/83	23	342	2071
Ours (-Novelty)	60	5242	7445	62/80	12	157	2649
Ours (-Risk)	87	3623	9064	72/81	30	237	2255
Ours: ThriftyDAgger	84	6840	5847	76/86	27	426	1696

β_h and β_r. We use $\beta_h = 0.015$, $\beta_r = 0.25\beta_h$ and use a noise covariance matrix of $0.02\mathcal{N}(0, I)$ when injecting noise into the supervisor actions. These values were tuned to strike a balance between supervisor burden and policy performance.

7.2.5 HG-DAgger

All hyperparameters and architectures are identical to those used for Behavior Cloning, without the extra offline demonstrations. Note that for HG-DAgger, the supervisor determines the timing and length of interventions.

7.3 Environment Details and Additional Metrics

7.3.1 Peg Insertion in Simulation

We evaluate our algorithm and baselines in the Robosuite environment (`https://robosuite.ai`) [48], which builds on MuJoCo [51] to provide a standardized suite of benchmark tasks for robot learning. Specifically, we consider the "Nut Assembly" task, in which a robot must grab a ring from a random initial pose and place it over a cylinder at a fixed location. We consider a variant of the task that considers only 1 ring and 1 target, though the simulator allows 2 rings and 2 targets. The states are $s \in \mathbb{R}^{51}$ and actions $a \in \mathbb{R}^5$ (translation in the XY-plane, translation in the Z-axis, rotation around the Z-axis, and opening or closing the gripper). The simulated robot arm is a UR5e, and the controller reaches a commanded pose via operational space control with fixed impedance. To avoid bias due to variable teleoperation speeds and ensure that the Markov property applies, we abstract 10 timesteps in the simulator into 1 environment step, and in supervisor mode we pause the simulation until keyboard input is received. This prevents accidentally collecting "no-op" expert labels and allows the end effector to "settle" instead of letting its momentum carry on to the next state. In practice it does not make the task more difficult, as control is still fine-grained enough for precise manipulation. Each episode is terminated upon successful task completion or when 175 actions are executed. Interventions are collected through a keyboard interface. In Table 4, we report additional metrics for the peg insertion simulation experiment and find that ThriftyDAgger solicits fewer interventions than prior algorithms at training time while achieving a higher success rate during training than all algorithms other than HG-DAgger, though it does request more human actions. The train success rate column also indicates that ThriftyDAgger achieves *throughput* comparable to HG-DAgger and higher than other baselines, as ThriftyDAgger has more task successes in the same amount of time (10,000 timesteps for all algorithms). At execution time, ThriftyDAgger collects lengthier interventions than prior algorithms, but as a result is able to succeed more often at execution time as discussed in the main text.

7.3.2 Block Stacking in Simulation

To further evaluate the algorithm and baselines in simulation, we also consider the block stacking task from Robosuite (see previous section). Here the robot must grasp a cube in a randomized initial pose and place it on top of a second cube in another randomized pose. See Table 5 for training results

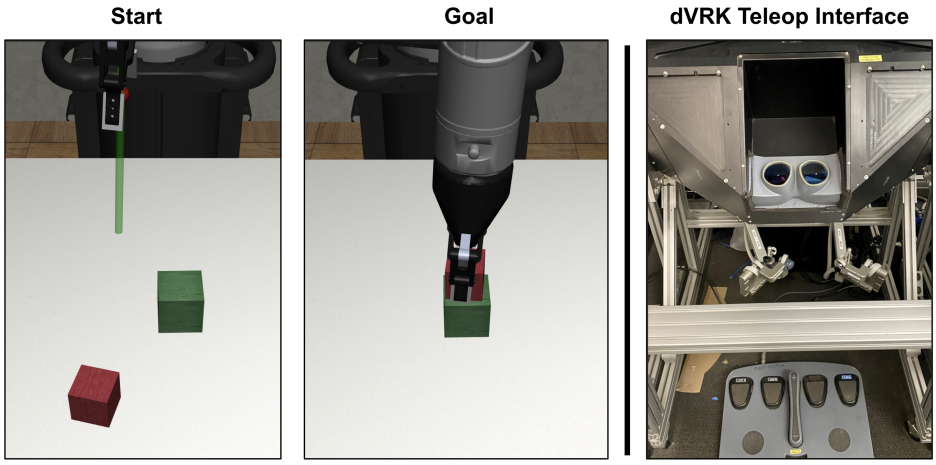

Figure 3: **Left:** An example start and goal state for the block stacking environment in Robosuite. The goal is to place the red block on top of the green one. Initial poses of both blocks are randomized. **Right:** The da Vinci Research Kit Master Tool Manipulator (MTM) 7DOF interface used to provide human interventions in the physical experiments. The human expert views the workspace through the viewer (top) and teleoperates the robot by moving the right joystick (middle) in free space while pressing the rightmost pedal (bottom).

Table 5: **Block Stacking in Simulation Results:** We report the number of interventions (Ints), number of human actions (Acts (H)), and number of robot actions (Acts (R)) during training (over successful trajectories as in Table 1) and report the success rate of the robot policy after training when no interventions are allowed (Auto Succ.). We also report the total number of interventions (T Int), total number of actions from the human (offline and online, in T Acts (H)), total number of actions executed by the robot (T Acts (R)), and the success rate of the mixed control policy during training (Train Succ.). Results suggest that ThriftyDAgger achieves comparable performance to HG-DAgger in terms of both autonomous and training success rates while outperforming the other baselines and ablations. ThriftyDAgger also solicits fewer interventions than prior algorithms, but generally requires more human actions.

Algorithm	Ints	Acts (H)	Acts (R)	Auto Succ.	T Ints	T Acts (H)	T Acts (R)	Train Succ.
Behavior Cloning	N/A	N/A	68.0 ± 11.4	$5/100$	N/A	3532	N/A	N/A
SafeDAgger	5.00 ± 3.41	40.5 ± 14.1	44.3 ± 25.6	$3/100$	574	4387	7290	27/68
LazyDAgger	1.81 ± 1.02	25.8 ± 17.8	56.6 ± 28.3	$40/100$	85	2940	8737	36/75
HG-DAgger	1.62 ± 0.91	22.5 ± 16.5	54.6 ± 14.2	$\mathbf{56/100}$	201	4535	7142	124/125
Ours (-Novelty)	0.65 ± 0.70	43.7 ± 13.3	28.6 ± 28.5	$8/100$	37	3599	8078	23/69
Ours (-Risk)	1.89 ± 0.72	12.9 ± 7.7	72.4 ± 25.5	$31/100$	109	2518	9159	47/79
Ours: ThriftyDAgger	1.33 ± 0.76	35.4 ± 15.8	37.2 ± 27.5	$\mathbf{52/100}$	153	5873	5804	111/120

and Figure 3 for an illustration of the experimental setup. Due to the randomized place position, small placement region, and geometric symmetries, the task is more difficult than peg insertion, as evidenced by the lower autonomous success rate for all algorithms. However, we still see that ThriftyDAgger achieves comparable performance to HG-DAgger in terms of autonomous success rate, success rate during training, and throughput, while outperforming the other baselines and ablations. ThriftyDAgger also solicits fewer interventions than prior algorithms, but generally requires more human actions as these interventions tend to be lengthier. This makes ThriftyDAgger well-suited to situations in which the cost of context switches (latency) may be high.

7.3.3 Physical Cable Routing

Finally, we evaluate our algorithm on a visuomotor cable routing task with a da Vinci Research Kit surgical robot. We take RGB images of the scene with a Zivid One Plus camera inclined at about 45 degrees to the vertical. These images are cropped into a square and downsampled to 64×64 before they are passed to the neural network policy. The cable state is initialized to approximately the same shape (see Figure 2) with the cable initialized in the robot's gripper. The workspace is approximately 10 cm $\times$ 10 cm, and each component of the robot action $(\Delta x, \Delta y)$ is at most 1 cm in magnitude. To avoid collision with the 4 obstacles, we implement a "logical obstacle" as 1-cm radius balls around the center of each obstacle. Actions that enter one of these regions are projected to the boundary of the circle. Each episode is terminated upon successful task completion or 100 actions executed. Interventions are collected through a 7DOF teleoperation interface (Figure 3) that matches

Table 6: **Physical Cable Routing Additional Metrics:** We report additional statistics for the peg insertion task: total number of interventions (T Ints), total number of offline and online human actions (T Acts (H)), and total number of robot actions (T Acts (R))) at training time across all trajectories. We report these same metrics at execution time, but T Acts (H) does not include offline human actions, as at execution time it does not refer to the number of training samples for the robot policy. We also report the success rate of the mixed control policy at *training* time (Train Succ.). Results suggest that ThriftyDAgger needs fewer interventions than HG-DAgger while achieving a similar training success rate. At execution time, we find that ThriftyDAgger solicits the same number of interventions as HG-DAgger, but requires fewer human and robot actions.

Algorithm	Training Interventions			Train Succ.	Execution Interventions		
	T Ints	T Acts (H)	T Acts (R)		T Ints	T Acts (H)	T Acts (R)
Behavior Cloning	N/A	1381	N/A	N/A	N/A	N/A	N/A
HG-DAgger	31	1682	1199	20/20	6	41	1109
Ours: ThriftyDAgger	27	1728	1153	19/21	6	23	919

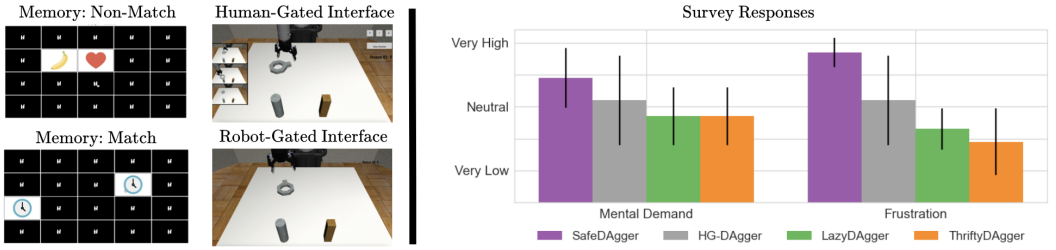

Figure 4: **User Study Survey Results:** We illustrate the user study interface for the human-gated and robot-gated algorithms (left) and users' survey responses regarding their mental load and frustration (right) for each algorithm. Results suggest that users report similar levels of mental load and frustration for ThriftyDAgger and LazyDAgger, but significantly higher levels of both metrics for HG-DAgger and SafeDAgger. We hypothesize that the sparing and sustained interventions solicited by ThriftyDAgger and LazyDAgger lead to greater user satisfaction and comfort compared to algorithms which force the user to constantly monitor the system (HG-DAgger) or frequently context switch between teleoperation and the distractor task.

the pose of the robot arm, with rotation of the end effector disabled. Teleoperated actions are mapped to the robot's action space by projecting pose deltas to the 2D plane at 1 second intervals. The human teleoperates the robot at a frequency that roughly corresponds to taking actions with the maximum magnitude (1 cm / sec). In Table 6, we report additional metrics for the physical cable routing experiment and find that ThriftyDAgger solicits a number of interventions similar to HG-DAgger while achieving a similar success rate during training. This again indicates that ThriftyDAgger is able to learn intervention criteria competitive with human judgment. At execution time, we find that ThriftyDAgger solicits the same number of interventions as HG-DAgger, but requires fewer human and robot actions than HG-DAgger.

7.4 User Study Details

Here we detail the protocol for conducting user studies with ThriftyDAgger and comparisons and discuss qualitative results based on participants' answers to survey questions measuring their mental load and levels of frustration when using each of the algorithms.

7.4.1 User Study Interface

Figure 4 (left) illustrates the interface used for the user study. The user study is performed with the same peg insertion environment used for simulation experiments, but with 3 robots performing the task in parallel. The base policy is initialized from 30 demos, as in the other simulation experiments. To speed up the task execution for the user study, each action has twice the magnitude as in the peg insertion experiments. Since this results in shorter trajectories that are easier for Behavior Cloning to accomplish, we also inject a small amount of Gaussian noise (covariance matrix = $0.02\mathcal{N}(0, I)$).

In the human-gated study with HG-DAgger, participants are shown videos of all 3 robots attempting to perform the task in a side pane (Figure 4, top right of left pane) and are instructed to monitor all of the robots and intervene when they deem it appropriate. In all robot-gated studies, participants are instructed to play the Concentration game until they hear a chime, at which point they are instructed to switch screens to the teleoperation interface. The Concentration game (also called Memory) is

Table 7: **Wall Clock Time:** We compare the total amount of wall clock time and total amount of human wall clock time averaged over the 10 subjects in the user study. Human Wall Clock Time refers to the amount of time the human spent actively teleoperating a robot, while Total Wall Clock Time measures the amount of time taken by the total experiment. ThriftyDAgger requires the lowest amount of human time, and the total amount of time is relatively consistent. Note that HG-DAgger takes more Total Wall Clock Time as it takes longer to simulate the "bird's eye view" of all 3 robots, and that autonomous robots can still make task progress independently while a human is operating a robot.

Algorithm	Human Wall Clock Time (s)	Total Wall Clock Time (s)
SafeDAgger	448.0 ± 48.1	613.0 ± 33.1
LazyDAgger	415.3 ± 90.3	609.6 ± 49.5
HG-DAgger	532.6 ± 105.2	792.8 ± 68.7
Ours: ThriftyDAgger	$\mathbf{365.4 \pm 88.1}$	625.5 ± 52.3

illustrated on the left of the left pane in Figure 4: the objective is to find pairs of cards (all of which are initially face-down) which have matching pictures on their front side. Examples of a non-matching pair and a matching pair are illustrated in Figure 4.

All robots which require interventions are placed in a FIFO queue, with participants serving intervention requests sequentially until no robot requires intervention. Thus, the participant may be required to provide interventions for multiple robots in succession if multiple robots are currently in the queue. When no robot requires assistance, the teleoperation interface turns black and reports that no robot currently needs help, at which point participants are instructed to return to the Concentration game.

7.4.2 NASA TLX Survey Results

After each participant is subjected to all 4 conditions (SafeDAgger, LazyDAgger, ThriftyDAgger, and HG-DAgger) in a randomized order, we give each participant a NASA TLX survey asking them to rate their mental demand and frustration for each of the conditions on a scale of 1 (very low) - 5 (very high). Results (Figure 4 right pane) suggest that ThriftyDAgger and LazyDAgger impose less mental demand and make participants feel less frustrated than HG-DAgger and SafeDAgger. During experiments, we found that participants found it cumbersome to keep track of all of the robots simultaneously in HG-DAgger, while the frequent context switches in SafeDAgger made participants frustrated since they were often unable to make much progress in the Concentration Game and felt that the robot repeatedly asked for interventions in very similar states.

7.4.3 Wall Clock Time

We report additional metrics on the wall clock time of each condition in Table 7. Since all experiments are run for the same 350 time steps, total wall clock time is relatively consistent. However, HG-DAgger takes longer, as it takes more compute to render all three robot views at once. ThriftyDAgger takes less total human time than the baselines, allowing the human to make more progress on independent tasks. Note that other robots in autonomous mode can still make task progress during human intervention. Note also that HG-DAgger requires human attention for the Total Wall Clock Time, as the human must supervise all the robots even if he or she is not actively teleoperating one (as recorded by Human Wall Clock Time).

7.4.4 Detailed Protocol

For the user study, we recruited 10 participants aged 18-37, including members without any knowledge or experience in robotics or AI. All participants are first assigned a randomly selected user ID. Then, participants are instructed to play a 12-card game of Concentration (also known as Memory) (`https://www.helpfulgames.com/subjects/brain-training/memory.html`) in order to learn how to play. Then, users are given practice with both the robot-gated and human-gated teleoperation interfaces. To do this, the operator of the study (one of the authors) performs one episode of the task in the robot-gated interface and briefly explains how to control the human-gated interface. Then, participants are instructed to perform one practice episode in the robot-gated teleoperation interface and spend a few minutes exploring the human-gated interface until they are confident in the usage of both interfaces and in how to teleoperate the robots. In the robot-gated experiments, participants are instructed to play Concentration when no robot asks for help, but to immediately switch to helping the robot whenever a robot asks for help. In the human-gated experiment with HG-DAgger, participants are instructed to continuously monitor all of the robots and perform interventions which they believe will maximize the number of successful episodes. During the robot-gated study, participants play

the 24-card version of Concentration between robot interventions. If a participant completes the game, new games of Concentration are created until a time budget of robot interactions is hit. Then for each condition, the participant is scored based on (1) the number of times the robot successfully completed the task and (2) the number of total matching pairs the participant found across all games of Concentration.