

Finally, we provide some more insight on our work. In particular, it consists of:

- Appendix **A Algorithm**, shows the pseudocode of our method.
- Appendix **B Benchmarks**, explains the different comparison environments in more detail.
- Appendix **C Ablations**, We study the effect of changing key parts of our algorithm or certain hyperparameters.
- Appendix **D Hyperparameters**, depicts the hyperparameters used in the different runs.

## A Algorithm

We reproduce some of the learning objectives here for posterity. The following is the objective for training the goal selector with human provided comparative feedback:

$$\mathcal{L}_{\text{rank}}(\theta) = -\mathbb{E}_{(s_i, s_j, g) \sim \mathcal{D}} \left[ \mathbb{1}_{i < j} \left[ \log \frac{\exp -d_\phi(s_i, g)}{\exp -d_\phi(s_i, g) + \exp -d_\phi(s_j, g)} \right] + \right. \quad (1)$$

$$\left. (1 - \mathbb{1}_{i < j}) \left[ \frac{\exp -d_\phi(s_j, g)}{\exp -d_\phi(s_i, g) + \exp -d_\phi(s_j, g)} \right] \right] \quad (2)$$

The density model  $p_\psi(s_t, g_{\text{sub}})$  can be trained on a dataset  $\mathcal{D} = \{(s_t^i, g_{\text{sub}}^i)\}_{i=1}^N$  of relabeled  $(s_t, g_{\text{sub}})$  tuples via the following objective:

$$\max_{\psi} \mathbb{E}_{(s_t, g_{\text{sub}}) \sim \mathcal{D}} [\log p_\psi(s_t, g_{\text{sub}})] \quad (3)$$

Different choices of family for  $p_\psi(s_t, g_{\text{sub}})$  yield different variants. We leverage tabular density models and autoregressive. Policies trained via hindsight self supervision optimize the following objective:

$$\arg \max_{\pi} \mathbb{E}_{\tau \sim \mathbb{E}_g [\bar{\pi}(\cdot|g), g \sim p(g)]} \left[ \sum_{t=0}^T \log \pi(a_t | s_t, \mathcal{G}(\tau)) \right] \quad (4)$$

To sample goals from the learned proximity metric, we can sample  $g_{\text{sub}} \sim p(g_{\text{sub}}|s, g)$ , where

$$p(g_{\text{sub}}|s, g) = \frac{\exp -d_\phi(s, g)}{\sum_{s' \in \mathcal{D}} \exp -d_\phi(s', g)} \quad (5)$$

Below, we show our algorithm in pseudo-code format.

---

### Algorithm 1 METHOD NAME

---

```

1: Input: Human  $\mathcal{H}$ , goal  $g$ , starting position  $s$ 
2: Initialize policy  $\pi$ , density model  $d_\theta$ , proximity model  $f_\theta$ , data buffer  $\mathcal{D}$ , proximity model buffer  $\mathcal{G}$ 
3: while True do
4:    $p \sim p(g)$ 
5:    $\mathcal{D}_\tau \leftarrow \text{PolicyExploration}(\pi, \mathcal{G}, g, \mathcal{D})$ 
6:    $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\tau$ 
7:    $\pi \leftarrow \text{TrainPolicy}(\pi, \mathcal{D})$  (hindsight relabeling [17], Eq 4)
8:    $\mathcal{G} \leftarrow \mathcal{G} \cup \text{CollectFeedback}(\mathcal{D}, \mathcal{H})$  ( Sec 4.2)
9:    $f_\theta \leftarrow \text{TrainGoalSelector}(f_\theta, \mathcal{G})$  (Eq 1 via the Bradley-Terry model [48])
10:   $d_\theta \leftarrow \text{TrainDensityModel}(d_\theta, \mathcal{G})$  (Eq 3, [50, 51, 52])
11: end while

```

---

---

**Algorithm 2** PolicyExploration

---

```
1: Input: policy  $\pi$ , goal selector  $f_\theta$ , goal  $g$ , data buffer  $\mathcal{D}$ 
2:  $\mathcal{D}_\tau \leftarrow \{\}$ 
3:  $s \leftarrow s_0$ 
4: for  $i = 1, 2, \dots, N$  do
5:   every  $k$  timesteps:
6:      $\mathcal{S} \sim \text{ObtainReachableStates}(d_\theta, s, \mathcal{D})$  (Sec 4.2, [50, 51])
7:      $g_b \sim \text{SampleClosestState}(f_\theta, g, \mathcal{S})$  (Sec 4.2, Eq 5)
8:     while NOT stopped do
9:       Take action  $a \sim \pi(a|s, g_b)$ 
10:    end while
11:    Execute  $\pi_{\text{random}}$  for  $H$  timesteps
12:    Add  $\tau$  to  $\mathcal{D}_\tau$  without redundant states
13: end for
14: return  $\mathcal{D}_\tau$ 
```

---

## B Evaluation Environments

We briefly discussed the evaluation environments we used to compare our method to previous work. In this section we will go through the details of each of them.

- **Pointmass navigation:**

As mentioned before, this is a holonomic navigation task in an environment with four rooms, where the objective is to move between the two farthest rooms. This is modification of a benchmark proposed in [17].

In this benchmark, the observation space consists of the position of the agent, that is,  $(x, y) \in \mathbb{R}^2$ , while the action space is discrete of cardinality 9. In particular there are 8 actions corresponding to moving a fixed amount relative to the current position, the directions are the ones parallel to the axis and their diagonals. Finally, there is an action that encodes no-movement.

The number of timesteps given to solve this task is 50. Finally, as for a human proxy, we use the distance to the commanded goal, taking into account the walls, i.e., we consider to shortest distance according to the restrictions of the environment.

- **LoCoBot navigation:**

This benchmark is similar to the previous one, since it also tackles 2D navigation. The main difference is that, in this one, we are working simulation a LoCoBot, in a real-life-looking environment, and dealing with differential driving instead of holonomic. The environment tries to resemble the one we do in the real world with a TurtleBot, so that results obtained in simulation are, to a certain extent, informative about how our robot would perform with the different algorithms in the real world.

This benchmark is similar to the Four Rooms one since we are also dealing with 2D navigation. The main difference is that we are working with a simulated robot in Mujoco, in particular a LoCoBot, in a real-life-like environment, in which there is a kitchen and a living room, thus presenting some obstacles for the robot such as tables or a couch. Additionally, the robot works with differential driving, as a LoCoBot or Turtlebot would do.

In this environment the goals the robot should be able to learn how to reach are the lower right and the upper left corners. In this environment, the state space is the absolute position of the robot, together with its angle  $(x, y, \theta) \in \mathbb{R}^3$ . As we are working with differential drive, the action space is discrete encoding 4 actions: rotate clockwise, rotate counterclockwise, move forward and no movement.

The LoCoBot should reach the given goal within 40 timesteps. As before, for the proxy human we just use the distance to the goal, accounting for obstacles.

579 • **Block Pusher:**

580 This is a robotic manipulation problem, where a Sawyer robotic arm pushes an obstacle to  
 581 a given location. This benchmark is also a modification of one of the benchmarks proposed  
 582 by [17]

583 In this environment the state space consists of the position of the puck and the position of  
 584 the arm  $(x_1, y_1, x_2, y_2) \in \mathbb{R}^4$ . The actions space is the same as in the Pointmass navigation  
 585 benchmark (i.e. discrete with 9 possible actions).

586 The arm should push the object to the desired location in at most 75 timesteps. As for the  
 587 human proxy, the reward function we use is the following:

$$r = \max(\text{distance\_puck\_finger}, 0.05) + \text{distance\_puck\_goal}$$

588 • **TurtleBot navigation in the Real World:**

589 This benchmark is similar to the LoCoBot navigation one, the major difference between  
 590 the two being that this one takes place in the real world instead of a simulation.

591 The goal is to learn how to navigate between two opposite corners in a home-looking  
 592 environment, with a lot of obstacles. The action and the observation space are the same  
 593 than in the LoCoBot navigation environment. That is, the action space is discrete with 4  
 594 possible actions (move clockwise, counterclockwise, forward and don't move), while the  
 595 state space consists of the absolute position of the TurtleBot and its angle  $(x, y, \theta) \in \mathbb{R}^3$ .  
 596 In order to get this state, we have a top-down camera and the TurtleBot has a blue and red  
 597 semispheres, whose position can be detected by the camera, thus obtaining the position of  
 598 the TurtleBot, and its angle (by computing the direction of the vector between the blue and  
 599 red semispheres of the locobot). Finally, we do collision avoidance by leveraging the depth  
 600 sensor of the top-down camera.

601 The TurtleBot should reach any goal in 25 timesteps. And for the human proxy we just use  
 602 the euclidean distance to the goal.

- 603 • **Real World Pusher with Franka Panda:** This benchmark is relatively similar to the  
 604 pusher environment in simulation, except it is with a Franka Emika panda robot in the real  
 605 world. The goal is to learn how to push an object in the plane between two different corners  
 606 of an arena. The challenge here is that the pusher is a cylindrical object and planar pushing  
 607 in this case needs careful feedback control, otherwise it is quite challenging. The action  
 608 space is 9 dimensional denoting motion in each direction, diagonals and a no-op. The state  
 609 space consists of the position of the robot end effector and the object of interest. In order  
 610 to get this state we use a calibrated camera and an OpenCV color filter, although this could  
 611 be replaced with any more sophisticated state-estimation system. The system is provided  
 612 very occasional intervention when the object is stuck in corners, roughly one nudge every  
 613 30 minutes. Success is measured by resetting the object to one corner of the arena and  
 614 measuring success at reaching another corner.

615 In this environment, we leverage a synthetic replacement for a human supervisor. We sim-  
 616 ulate the feedback that would have been given by the human by designing a labelling func-  
 617 tion. This one will give preferences for having the cylindrical object close to the target goal  
 618 and for having the end effector close to the object. Furthermore, we will give preferences  
 619 based on the direction from which the end effector is reaching the object. The formula is  
 620 indicated below:

$$r = \max(\text{distance\_puck\_finger}, 0.05) + \text{distance\_puck\_goal}$$

621 **C Ablations**

622 To better understand the details of which design decisions affect the performance of GEAR, we con-  
 623 duct ablation studies to understand the impact of various design decisions on learning progress.  
 624 Specifically, we aim to understand the impact of (1) the threshold  $\epsilon$  for likelihood at which a state is  
 625 considered "reachable", (2) the frequency at which new intermediate subgoals are sampled during

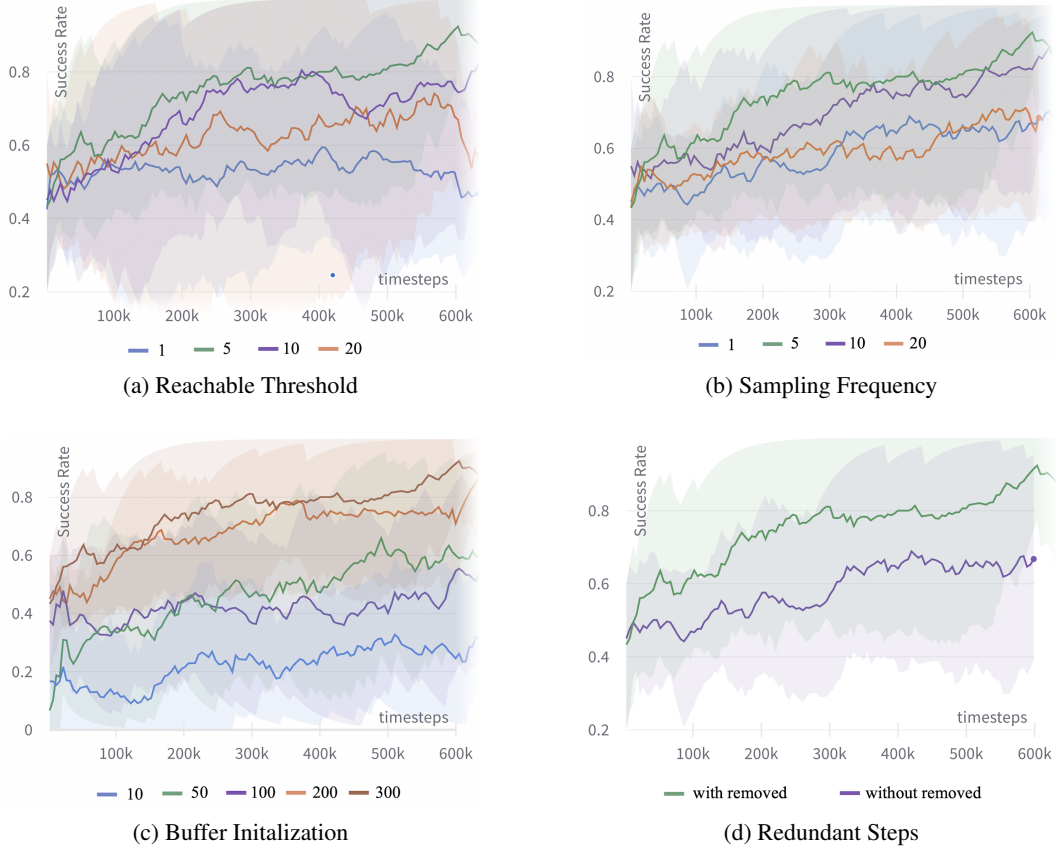


Figure 6: Ablations of GEAR: We studied the ablations of GEAR with four different setups. In a), we modify the reachable threshold with a parameter of 1, 5, 10, 20. In b), we verify the effects of sampling frequency. In c), we use a different number of offline data to initialize the buffer. The offline data are collected by driving the agent randomly explore the environment. In d), we evaluate the performance of our algorithm by removing and not removing the redundant exploration steps at each end of the trajectories.

626 exploration, (3) the algorithm removes redundant exploration steps during exploration, we ablate  
 627 how important this step is in performance, and (4) we ablate how much pre-training data is required  
 628 for learning and how this affects learning progress. We find that 1) choosing the right threshold  
 629 for the success of our algorithm is critical. The best reachable threshold used for the pointmass  
 630 navigation task is 5. Larger (threshold = 10, 20) or smaller (threshold = 1) would not make the  
 631 algorithm work better. 2) the ablation for sampling frequency shows that right sampling frequency  
 632 would help boost the performance. We tried sampling frequency with 1, 5, 10 and 20. The results  
 633 show that 5 and 10 have a similar performance. Too small (sampling frequency = 1) or too large  
 634 (sampling frequency = 20) do not work well. If the sampling frequency is too small, the agent  
 635 might not be able to reach the subgoal and too frequent subgoal selection would make the perfor-  
 636 mance drop. If the sampling frequency is too large, there would have more redundant wandering  
 637 steps which make the learning less efficient. 3) We found that removing the redundant steps would  
 638 help training significantly. Without removing the redundant steps in the trajectory sampling, there  
 639 would be stationary states when the agent is stuck in the environment which could lead to the drop  
 640 of performance. 4) More random pre-trained data would help build up the reachable set and further  
 641 improve the performance.

## D Hyperparameters

In this section we state the primary hyperparameters used across the different experiments. All the values are shown in Table 1

Parameter	Value
<i>default (to those that apply)</i>	
Optimizer	Adam [56]
Learning rate	$5 \cdot 10^{-4}$
Discount factor ( $\gamma$ )	0.99
Reward model architecture	MLP(400, 600, 600, 300)
Use Fourier Features in reward model	True
Use Fourier Features in policy	True
Use Fourier Features in density model	True
Batch size for policy	100
Batch size for reward model	100
Epochs policy	100
Epochs goal selector	400
Train policy freq	10
Train goal selector freq	10
goal selector num samples	1000
Stop threshold	0.05
<i>LoCoBot navigation</i>	
Stop threshold	0.25
<i>TurtleBot navigation in Real World</i>	
Stop threshold	0.1
policy updates per step	50
<i>Oracle Densities</i>	
reachable threshold	5
<i>VICE</i>	
reward model epochs	20
<i>Human Preferences</i>	
reward model epochs	20
<i>Autoregressive</i>	
reachable threshold	0.25
Epochs density model	30000
Train autoregressive model freq	300
Batch size for the density model	4096

Table 1: Hyperparameters used when GEAR

## 645 E Web Interface for Providing Feedback

646 Here we show an example interface for providing feedback for the TurtleBot navigation task:

**Autonomous Robotic Reinforcement Learning with Asynchronous Human Feedback**

Anonymous Author(s)

Affiliation

Navigation Task


Please first WATCH the video [here](#)

Please fill in the [form](#)


You will be prompted with 0/30 images

SHOW TASK DESCRIPTION


Goal State



LEFT



DON'T KNOW



RIGHT

Figure 7: Visualization of the human supervision web interface to provide feedback asynchronously during robot execution. Users are able to label which of two states is closer to a goal or say they are unable to judge.