

447 **A Algorithm**

448 Below we provide the pseudo-code for the LARGST-JUMP procedure. Give a list of noisily increasing
 449 values — in our case, a list of noisily increasing “naturalness” values as we increase λ , the goal of
 450 LARGST-JUMP is to identify the two values, adjacent or not, that have the largest gap between them.
 451 This enables iterative refinement in Alg. 1.

Algorithm 2 Procedure: LARGEST-JUMP

Require:

```

1: all  $\lambda$ 's  $\Lambda_{\text{all}}$ , all target values  $S$ .
2: length of sliding window  $L$ 
procedure LARGEST-JUMP ( $S, \Lambda_{\text{all}}$ )
3: Sort  $S$  based on  $\Lambda_{\text{all}}$ .
4: lower_bound_so_far, upper_bound_so_far = [], []
5: lower_idx_so_far, upper_idx_so_far = [], []
6: high_acc, low_acc = 10, -1
7: high_idx, low_idx = len(S), 1
8: for  $i = 0, \dots, \text{len}(S) - 1$  do
9:   if  $i = 1$  then
10:    lower_bound_so_far.append( $S[0]$ )
11:    lower_idx_so_far.append(low_idx)
12:   else
13:    irstart = max( $i + 1 - L, 0$ )
14:    lower_bound_so_far.append(min( $S[\text{istart} : i+1]$ ))
15:    lower_idx_so_far.append(argmin( $S[\text{istart} : i+1]$ ) + irstart)
16:   end if
17: end for
18: for  $i = \text{len}(S) - 1, \dots, 0$  do
19:   if  $i = 1$  then
20:    upper_bound_so_far.append( $S[-1]$ )
21:    upper_idx_so_far.append(high_idx)
22:   else
23:    upper_bound_so_far.append(max( $S[i : i+L]$ ))
24:    upper_idx_so_far.append(argmax( $S[i : i+L]$ ) + i)
25:   end if
26: end for
27: max_gap_so_far = -1, max_gap_idx = null
28: for  $i = \text{len}(S), \dots, 1$  do
29:   diff = upper_bound_so_far[ $i$ ] - lower_bound_so_far[ $i$ ]
30:   if diff > max_gap_so_far then
31:    max_gap_so_far = diff
32:    max_gap_idx = (lower_idx_so_far[ $i$ ], upper_idx_so_far[ $i$ ])
33:   end if
34: end for
   return  $\Lambda_{\text{all}}[\text{max\_gap\_idxs}[0]], \Lambda_{\text{all}}[\text{max\_gap\_idxs}[1]]$ 
end procedure

```

452 **B Environment Setup**

453 **Assistive Gym Itch Scratching** We use the itch scratching environment proposed in [55] with the
 454 original settings. The biggest difference is that *we limit the itch positions to randomizing from two*
 455 *fixed points, one in the middle of the forearm and one in the middle of the upper arm*, as opposed
 456 to freely sampling from any position on the arms. This is to simplify the robot assistance problem so
 457 that we can focus on studying robot robustness.

458 We also modify the environment time-span to 100 steps so as to speed up downstream RL training. The
 459 reward function in the original itch-scratching environment does not fully capture unwanted behaviors
 460 such as the robot swinging its arm and making unwanted contacts. We modify the environment
 461 reward function to increase the distance penalty (the robot’s end-effect being far from the human) and
 462 add in contact penalty when the robot impacts areas other than the human’s arms.

463 **Co-Optimization** We adopt the co-optimization framework proposed in [52], where we have both
 464 the human and robot jointly optimize for the task reward. We train both policies using PPO [49]. At
 465 every RL step, we update both the robot and the human policies.

466 More specifically, we use the ray library and train co-optimized policy pairs using batch sizes of
 467 19,200 timesteps per iteration. We use the default learning rate and PPO hyperparameters in the
 468 RLLib library and train for a total of 400 iterations.

469 **Training Personalized Robot Policies** We keep the human policy from the co-optimized pair as our
 470 synthetic human policies. We can then train a personalized robot policy to assist the synthetic human.
 471 This is akin to programming robotic agents to assist humans. The resulting robot policy – which
 472 we refer to as personalized policies – is the focus of the paper and the target on which we compute
 473 the Natural-Adversarial curves. Note that there are different methods to find personalized policies
 474 besides running Vanilla RL. We describe them in more detail in Sec. C.

475 **Collecting Canonical Datasets** We collect datasets of 40 trajectories of synthetic humans and
 476 personalized policies as canonical datasets, which we later use to train GAN to compute “naturalness”.
 477 Basically, the GAN enforces that perturbation trajectories stay in the proximity of the canonical
 478 trajectories. *Here by “natural”, we mean human motions that are indistinguishable from the*
 479 *canonical trajectories.* This notation can apply to general human-robot interaction settings because
 480 such applications typically assume canonical trajectories (i.e. default dressing motions), and allow
 481 humans to fluctuate within some range of motions.

482 **Visualizations** Here in Fig. 7 we provide visualizations of the trajectories of four different co-
 483 optimized human-robot policy pairs. We use the same hyperparameters except for random seeds. The
 484 human motions are different amongst different seeds, and remain within reasonable motion range.

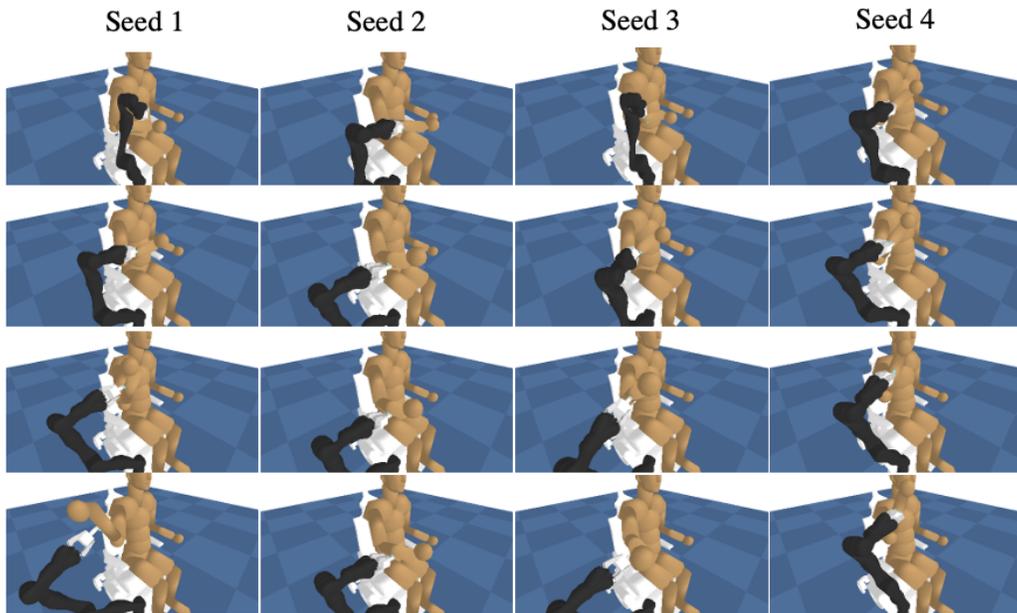


Figure 7

485 C Experiment Setup

486 In this section, we talk about the details of different methods for training personalized policies. We
 487 also detail the training of GAN for generating natural-adversarial human behaviors.

488 C.1 Learning Robot Policies

489 **Vanilla RL** We use off-the-shelf library on PPO algorithm. To facilitate policy training, we use the
490 original robot policy in the co-optimized human-robot pair as the expert to guide the training. More
491 specifically, we query the expert policy for actions and compute Behavior Cloning loss on the robot
492 policy. We set RL loss coefficient to 0.1 and BC loss coefficient to 1.

493 The robot policy is a 4-layer MLP with 100 hidden size. We use batch sizes of 9,600 timesteps per
494 iteration and train for a total of 240 iterations. We set environment gamma as 0.09. We use a learning
495 rate of 0.00005, and an eps of 0.0001. We also set the clip parameter as 0.3 in PPO. In each iteration,
496 we perform 30 epochs of policy optimization, with 20 mini-batch each. We use clipped policy loss,
497 clip gradient norm of robot policy by 20, and clip the value function by 10.

498 **PALM** We adopt the same setup as in [37]. To create a diverse human distribution, we vary the itch
499 position to randomly sample from anywhere on the two arms. This leads to more diverse human
500 movements. We use a recurrent history of 4 timesteps for PALM, and use a 4-layer recurrent VAE
501 with 24 encoder hidden size, and 4 latent size to predict human motions.

502 **Gleeve et al** Based on [54], we diversify the human population from the co-optimization phase.
503 During co-optimization, we jointly train 1 robot policy with 3 human policies initialized from
504 different seeds. The resulting human policies are different from each other and naturally induces
505 diversity in robot training. During the personalization phase, we train 1 robot personalized policy to
506 simultaneously work with the 3 human policies from co-optimization.

507 **Robust GT** We perform Robust GT by first computing the Natural-Adversarial curve, and then
508 manually select adversarial human policies that leads to the lowest robot reward given that the policy
509 naturalness $\in [0.2, 0.8]$. We visualize these failure cases in Sec. F. To train with these adversarial
510 humans, we sample them at 15% rate beside the original synthetic human. While one can train robot
511 policy this way from scratch, we load the previous vanilla RL policy and continue training with this
512 enhanced population.

513 C.2 Improving GAN training

514 While GANs are knownly difficult to train, there exists a large number of practical tricks in improving
515 GAN training. We find that the most helpful tricks are: adding noise with annealing. LS-GAN,
516 gradient penalty, and applying different weights for expert and human data. We experimented with
517 training discriminators on concatenated observation and action, and find that it does not lead to much
518 change. Thus we end up using observation-only discriminators.

519 Because human joints move at different rates and scales, it is important to add different amounts of
520 noise to different joint observations. We compute the joint movements from the existing dataset [53],
521 and multiply each joint movement’s standard deviation by a factor of 10. We then anneal this by a
522 decay rate of 0.98. We apply a gradient penalty of 0.3. We also set the expert loss rate in GAN as 4,
523 and the agent loss rate as 1. This helps prevent the discriminator from overfitting to the agent data
524 distribution and collapsing early in the training.

525 D Running RIGID Finding Natural-Adversarial Frontier Curve

526 To perform the RIGID algorithm to find the Natural-Adversarial Frontier Curve, we sweep over
527 $\lambda \in [0.00001, 10]$. We perform RIGID algorithm in an iterative refinement manner as in Alg. 1. We
528 keep 3 separate RIGID histories over 3 different random seeds. During each iteration, we select 6 new
529 λ ’s for each seed. We terminate after three iterations. This results to a total of 54 RL runs per curve.

530 **Plotting in the Natural-Adversarial Coordinate** We set the naturalness (x value) of the resulting
531 policies as the mean prediction result from the discriminator. To compute the adversarialness, we
532 normalize the negative robot reward in [200, 1400] range, and clip values that exceed this range to the
533 boundary. We find this range of manually performing different motions VR to find the mean negative
534 reward values of natural motions as well as adversarial human behaviors that lead to failures.

535 **E More Natural-Adversarial Curves**

536 We visualize additional Natural-Adversarial curves of different Vanilla RL robot policies, trained on
537 differently-seeded synthetic humans.

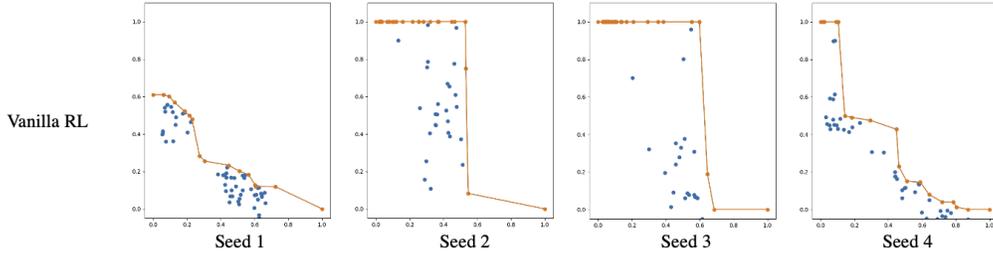


Figure 8

538 **F More Visualizations of Robot Failure Cases**

539 We visualize more failure cases of Vanilla RL as well as robust baselines.

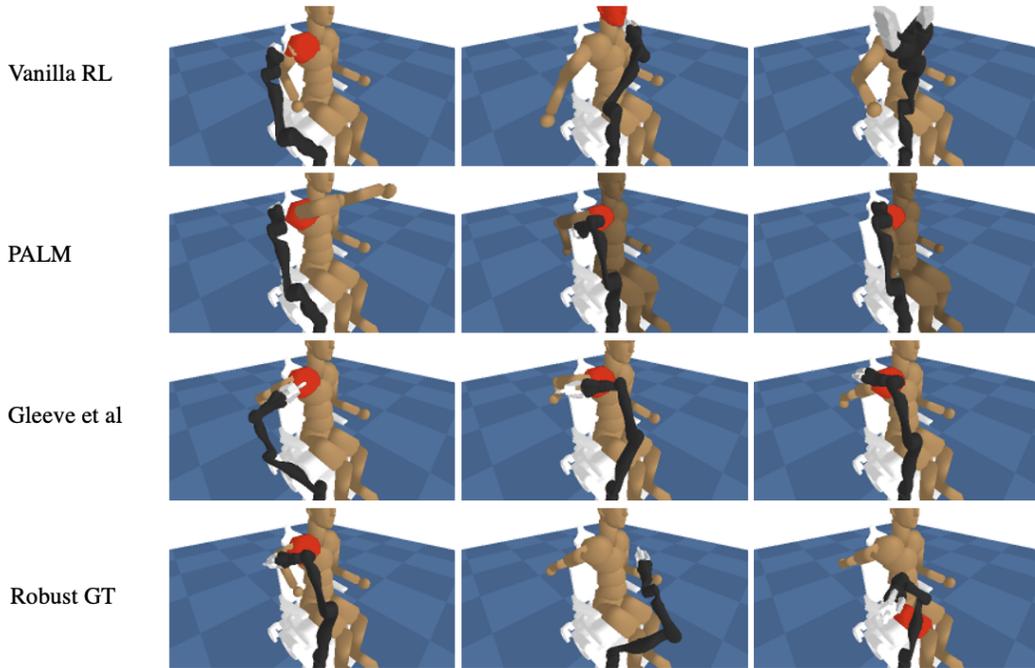


Figure 9

540 **G More details on User Study**

541 We provide more details on the user study in Sec. 5.2.

542 **G.1 VR Visualizations**

543 In the following figure Fig. 10, left and center are the user interacting with virtual robots through the
544 HTC VIVE headset and the hand controller. The right is the first-person view in VR[53].

545 We first have the users watch 3 iterations of canonical trajectories executed by the personalized robot
546 policies and the original synthetic humans. We then instruct them to perform similar trajectories in
547 their own ways.



Figure 10

548 **G.2 Interface**

549 For the study on evaluating faithfulness, we use the following interface in Fig. 11, where we display a
 550 canonical trajectory on the left, and display single-timestep snapshots of two trajectories, one from
 551 RIGID policies and one from user VR executions. We ask the user to select which snapshot has
 552 stronger correspondence to the left trajectory. We randomize the sequence for each question.

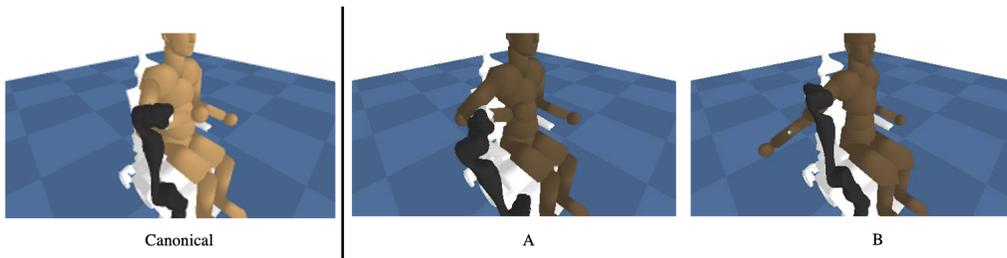


Figure 11