

A Simulation Training Details

In this section, we provide details about simulation training, including the used simulator backend, task designs, reinforcement learning (RL) training of teacher policy, and student policy distillation.

A.1 The Simulator

We use Isaac Gym Preview 4 [9] as the simulator backend. NVIDIA PhysX¹ is used as the physics engine to provide realistic and precise simulation. Simulation settings are listed in Table A.I. The robot model is from Franka ROS package². We borrow furniture models from FurnitureBench [84] to create various tasks that require complex and contact-rich manipulation.

Table A.I: Simulation settings.

Hyperparameter	Value
Simulation Frequency	60 Hz
Control Frequency	60 Hz
Num Substeps	2
Num Position Iterations	8
Num Velocity Iterations	1

A.2 Task Implementations

We implement four tasks based on the furniture model `square_table`: *Stabilize*, *Reach and Grasp*, *Insert*, and *Screw*. An overview of simulated tasks is shown in Fig A.1. We elaborate on their initial conditions, success criteria, reward functions, and other necessary information.

A.2.1 Stabilize

In this task, the robot needs to push the square tabletop to the right corner of the wall such that it is supported and remains stable in following assembly steps. The robot is initialized such that its gripper locates at a neutral position. The tabletop is initialized at the coordinate (0.54, 0.00) relative to the robot base. We then randomly translate it with displacements drawn from $\mathcal{U}(-0.015, 0.015)$ along x and y directions (the distance unit is meter hereafter). We also apply random Z rotation with values drawn from $\mathcal{U}(-15^\circ, 15^\circ)$. Four table legs are initialized in the scene. The task is successful only when the following three conditions are met:

- 1) The square tabletop contacts the front and right walls;
- 2) The square tabletop is within a pre-defined region;
- 3) No table leg is in the pre-defined region.

We use the following reward function:

$$r_t = w_{success} \mathbb{1}_{success} - w_{\dot{\mathbf{q}}} \|\dot{\mathbf{q}}_t\| - w_{action} \|a_t\|, \quad (\text{A.1})$$

where $w_{success}$ is the success reward, $\mathbb{1}_{success}$ indicates the success according to aforementioned conditions, $w_{\dot{\mathbf{q}}}$ penalizes large joint velocities, $\dot{\mathbf{q}}_t$ is the joint velocity, w_{action} penalizes large action commands, and a_t represents the action command at time step t . We set $w_{success} = 10$, $w_{\dot{\mathbf{q}}} = 10^{-5}$, and $w_{action} = 10^{-5}$. The episode length is 100. One episode terminates upon success or timeout.

A.2.2 Reach and Grasp

In this task, the robot needs to reach and grasp a table leg that is randomly spawned in the valid workspace region. The task is successful once the robot grasps the table leg and lifts it for a certain

¹<https://developer.nvidia.com/physx-sdk>

²https://github.com/frankaemika/franka_ros

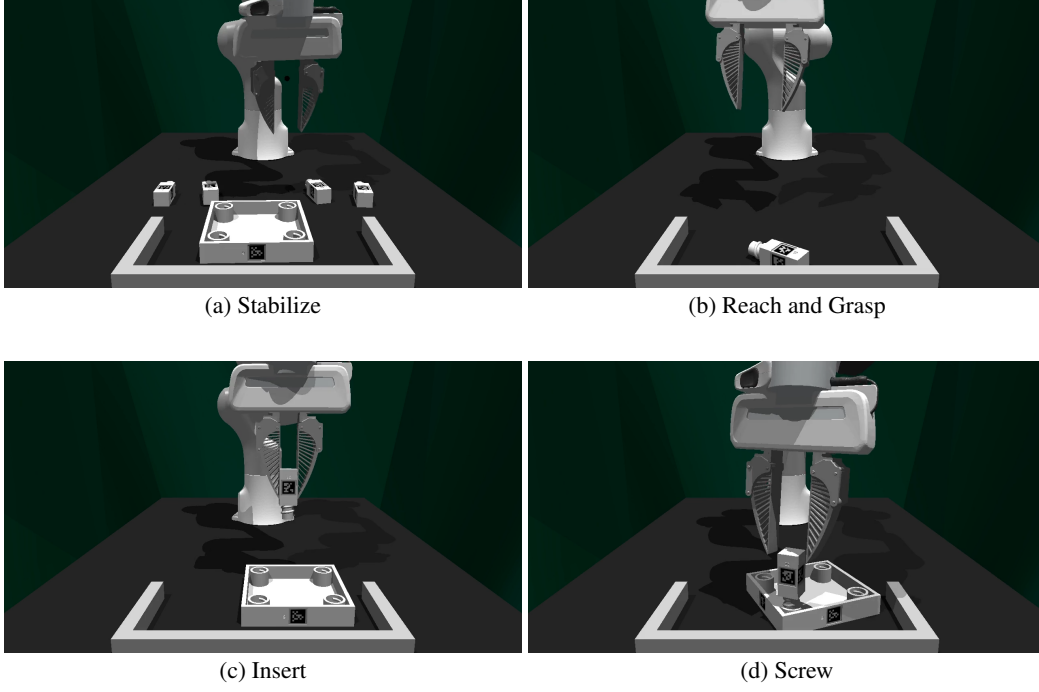


Figure A.1: Visualization of simulated tasks.

height. The object’s irregular shape limits certain grasping poses. For example, the end-effector needs to be near orthogonal to the table leg in the xy plane and far away from the screw thread. Therefore, we design a curriculum over the object geometry to warm up the RL learning. It gradually adjusts the object geometry from a cube, to a cuboid, and finally the table leg. In all curriculum stages, the reward function is

$$r_t = w_{distance}d + w_{lifted}\mathbb{1}_{lifted} + w_{success}\mathbb{1}_{success}. \quad (\text{A.2})$$

Here, $w_{distance}$ is the weight for distance reward, w_{lifted} is the reward for the leg being lifted, and $w_{success}$ is the success weight. d is the distance to the table leg and is calculated as

$$d = 1 - \tanh\left(\frac{10}{4}(d_{ee} + d_{left_finger} + d_{right_finger} + d_{orthogonal})\right), \quad (\text{A.3})$$

where d_{ee} is the distance between the end-effector and the table leg, d_{left_finger} is the distance between the left gripper tip to the table leg, d_{right_finger} is the distance between the right gripper tip to the table leg, and $d_{orthogonal}$ is the difference between the current and the orthogonal grasping orientations. We set $w_{distance} = 0.1$, $w_{lifted} = 1.0$, and $w_{success} = 200.0$. The episode length is 50. One episode terminates upon success or timeout.

A.2.3 Insert

In this task, the robot needs to insert a pre-grasped table leg into the far right assembly hole of the tabletop, while the tabletop is already stabilized. The tabletop is initialized at the coordinate $(0.53, 0.05)$ relative to the robot base. We then randomly translate it with displacements sampled from $\mathcal{U}(-0.02, 0.02)$ along x and y directions. We also apply random Z rotation with values drawn from $\mathcal{U}(-45^\circ, 45^\circ)$. We further randomize the robot’s pose by adding noises sampled from $\mathcal{U}(-0.25, 0.25)$ to joint positions. The task is successful when the table leg remains vertical and is close to the correct assembly position within a small threshold. We design curricula over the randomization strength to facilitate the learning. The following reward function is used:

$$r_t = w_{distance}d + w_{success}\mathbb{1}_{success}, \quad (\text{A.4})$$

where $w_{distance}$ is the weight for distance-based reward, d is the distance between the table leg and target assembly position, $w_{success}$ is the success weight, and $\mathbb{1}_{success}$ indicates task success. The distance d consists of an Euclidean distance $d_{position}$ and an orientation distance $d_{vertical}$ to encourage the robot to keep the table leg vertical.

$$d = 1 - \tanh\left(\frac{10}{2}(d_{position} + d_{vertical})\right) \quad (\text{A.5})$$

We set $w_{distance} = 1.0$ and $w_{success} = 100.0$. The episode length is 100. One episode terminates upon success or timeout.

A.2.4 Screw

In this task, the robot is initialized such that its end-effector is close to an inserted table leg. It needs to screw the table leg clockwise into the tabletop. We design curricula over the action space: at the early stage, the robot only controls the end-effector’s orientation; at the latter stage, it gradually takes full control. We slightly randomize object and robot poses during initialization. The reward function is

$$r_t = (1 - \mathbb{1}_{failure})(w_{screw}d_{screw} + w_{success}\mathbb{1}_{success}) - w_{deviation}d_{deviation}. \quad (\text{A.6})$$

Here, $\mathbb{1}_{failure}$ indicates the task failure, w_{screw} is the screwing reward weight, d_{screw} measures the screwed angle, $w_{success}$ is the success weight, and $\mathbb{1}_{success}$ indicates the task success. The task is considered as successful when the leg has been screwed 180° into the tabletop. It is considered as failed when the table leg tilts more than 10° from the vertical pose. We set $w_{screw} = 0.1$, $w_{success} = 100.0$, and $w_{deviation} = 10^{-2}$. The episode length is 200. One episode terminates upon success, failure, or timeout.

A.3 Teacher Policy Training

A.3.1 Model Details

Observation Space Besides proprioceptive observations, teacher policies also receive privileged observations to facilitate the learning. They include objects’ states (poses and velocities), end-effector’s velocity, contact forces, gripper left and right fingers’ positions, gripper center position, and joint velocities. Full observations are summarized in Table A.II.

Table A.II: The observation space for teacher policies.

Name	Dimension	Name	Dimension
Proprioceptive		Privileged	
Joint Position	7	Objects States	$N_{objects} \times 13$
Cosine Joint Position	7	End-Effector Velocity	6
Sine Joint Position	7	Contact Forces	$N_{objects} \times 3$
End-Effector Position	3	Left and Right Fingers’ Positions	6
End-Effector Rotation	4	Gripper Center Position	3
Gripper Width	1	Joint Velocity	7

Controller and Action Space An operational space controller (OSC) [72] is used in teacher policy training to improve sample efficiency. We follow Mistry and Righetti [108] to add nullspace control torques to prevent large changes in joint configuration. The action space is thus the change of end-effector’s pose. We further add a binary action to control gripper’s opening and closing. Formally, it can be expressed as $\mathcal{A}_{teacher} = (\delta x, \delta y, \delta z, \delta r, \delta p, \delta y, \mathbb{1}_{grripper})$, where $(\delta x, \delta y, \delta z) \in \mathbb{R}^3$ is the translation change, $(\delta r, \delta p, \delta y) \in \mathbb{R}^3$ is the rotation change, and $\mathbb{1}_{grripper} \in \{0, 1\}$ is the gripper action.

Model Architecture We use feed-forward policies in RL training. It consists of MLP encoders to encode proprioceptive and privileged vector observations, and unimodal Gaussian distributions as the action head. Model hyperparameters are listed in Table A.III.

Table A.III: Model hyperparameters for RL teacher policies.

Hyperparameter	Value	Hyperparameter	Value
Obs. Encoder Hidden Depth	1	Obs. Encoder Activation	ReLU
Obs. Encoder Hidden Dim	256	Action Head Hidden Layers	[256, 128, 64]
Obs. Encoder Output Dim	256	Action Head Activation	ELU [109]

A.3.2 Domain Randomization

We apply domain randomization during RL training to learn more robust teacher policies. Parameters are summarized in Table A.IV.

Table A.IV: Domain randomization used in RL training.

Parameter	Type	Distribution
Robot		
Mass	Scaling	$\mathcal{U}(0.5, 1.5)$
Friction	Scaling	$\mathcal{U}(0.7, 1.3)$
Joint Lower Limit	Scaling	$\log \mathcal{U}(1.00, 1.01)$
Joint Upper Limit	Scaling	$\log \mathcal{U}(1.00, 1.01)$
Joint Stiffness	Scaling	$\log \mathcal{U}(1.00, 1.01)$
Joint Damping	Scaling	$\log \mathcal{U}(1.00, 1.01)$
Simulation		
Gravity	Additive	$\mathcal{U}(0.0, 0.4)$
Objects		
Mass	Scaling	$\mathcal{U}(0.5, 1.5)$
Friction	Scaling	$\mathcal{U}(0.5, 1.5)$
Rolling Friction	Scaling	$\mathcal{U}(0.5, 1.5)$
Torsion Friction	Scaling	$\mathcal{U}(0.5, 1.5)$
Restitution	Additive	$\mathcal{U}(0.0, 1.0)$
Compliance	Additive	$\mathcal{U}(0.0, 1.0)$

A.3.3 RL Training Details

We use the model-free RL algorithm Proximal Policy Optimization (PPO) [80] to learn teacher policies. Hyperparameters are listed in Table A.V. We customize the framework from Makoviichuk and Makoviychuk [110] to use as our training framework.

Table A.V: Hyperparameters used in PPO training.

Hyperparameter	Value	Hyperparameter	Value
Learning Rate	5×10^{-4}	Critic Weight	4
Discount Factor	0.99	GAE [111] λ	0.95
Entropy Weight	0	PPO ϵ	0.2
Optimizer	Adam [112]	Horizon	32

906 A.4 Student Policy Distillation

907 A.4.1 Data Generation

908 We use trained teacher policies as oracles to generate data for student policies training. Concretely,
 909 we roll out each teacher policy to generate 10,000 successful trajectories for each task. We exclude
 910 trajectories that are shorter than 20 steps.

911 A.4.2 Observation Space

912 Student policies receive observations that can be obtained in the real world. They are point-
 913 cloud and proprioceptive observations. We synthesize point clouds from objects' 6D poses to im-
 914 prove the training throughput. Concretely, given the groundtruth point cloud of the m -th object
 915 $\mathbf{P}^{(m)} \in \mathbb{R}^{K \times 3}$, we transform it into the global frame through $\mathbf{P}_g^{(m)} = \mathbf{P}^{(m)} (\mathbf{R}^{(m)})^\top + (\mathbf{p}^{(m)})^\top$.
 916 Here $\mathbf{R}^{(m)} \in \mathbb{R}^{3 \times 3}$ and $\mathbf{p}^{(m)} \in \mathbb{R}^{3 \times 1}$ denote the object's orientation and translation in the
 917 global frame. Further, the point-cloud representation of a scene \mathbf{S} with M objects is aggregated
 918 as $\mathbf{P}^{\mathbf{S}} = \bigcup_{m=1}^M \mathbf{P}_g^{(m)}$. For the robot, we only include point clouds for its two fingers and ignore
 919 other parts. To facilitate policies to differentiate gripper fingers from the scene, we extend the co-
 920 ordinate dimension to include a semantic label $\in \{0, 1\}$ that indicates gripper fingers or not. This
 921 information can be obtained on real robots through forward kinematics. A full point cloud is then
 922 downsampled to 768 points. Table A.VI lists the observation space.

Table A.VI: The observation space for student policies.

Name	Dimension
Point Cloud	768×4
Proprioceptive	
Joint Position	7
Cosine Joint Position	7
Sine Joint Position	7
End-Effector Position	3
End-Effector Rotation	4
Gripper Width	1

923 A.4.3 Action Space Distillation

924 To reduce the controller sim-to-real gap before transfer, we train student policies to output in the
 925 configuration space. To achieve that, we relabel actions \hat{a} in trajectories generated by teacher policies
 926 from end-effector's delta poses to absolute joint positions. This is equivalent to set $\hat{a}_t = \mathbf{q}_{t+1}$ for
 927 all time steps. Therefore, the action space for student policies is $\mathcal{A}_{student} = (\mathbf{q}, \mathbb{1}_{grripper})$, where
 928 $\mathbf{q} \in \mathbb{R}^7$ is the joint position within the valid range. In simulation, student policies' actions are
 929 deployed with a joint position controller.

930 A.4.4 Model Architecture

931 We use feed-forward policies for tasks *Reach* and *Grasp* and recurrent policies for tasks
 932 *Stabilize* and *Screw* as we find they achieve the best distillation results. PointNets [81] are used to en-
 933 code point clouds. Recall that each point in the point cloud also contains a semantic label indicating
 934 the gripper or not. We concatenate point coordinates with these semantic labels' vector embeddings
 935 before passing into the PointNet encoder. We use Gaussian Mixture Models (GMM) [67] as the
 936 action head. Detailed model hyperparameters are listed in Table A.VII.

Table A.VII: **Model hyperparameters for student policies.**

Hyperparameter	Value	Hyperparameter	Value
Point Cloud		RNN	
PointNet Hidden Dim	256	RNN Type	LSTM [113]
PointNet Hidden Depth	2	RNN Num Layers	2
PointNet Output Dim	256	RNN Hidden Dim	512
PointNet Activation	GELU [114]	RNN Horizon	5
Gripper Semantic Embd Dim	128	GMM Action Head	
Feature Fusion		Hidden Dim	128
MLP Hidden Dim	512	Hidden Depth	3
MLP Hidden Depth	1	Num Modes	5
MLP Activation	ReLU	Activation	ReLU

A.4.5 Data Augmentation

We apply strong data augmentation during distillation. For point-cloud observations, random translation and random jitter are independently applied with a probability $P_{pcd.aug} = 0.4$. We also add Gaussian noises to proprioceptive observations. Augmentation parameters are listed in Table A.VIII.

Table A.VIII: **Data augmentation used in distillation.**

Hyperparameter	Value
Point Cloud	
Augmentation Probability	0.4
Random Translation Distribution	$\mathcal{U}(-0.04, 0.04)$
Random Jittering Ratio	0.1
Random Jittering Distribution	$\mathcal{N}(0, 0.01)$
Random Jittering Low	-0.015
Random Jittering High	0.015
Proprioception	
Prop. Noise Distribution	$\mathcal{N}(0, 0.1)$
Prop. Noise Low	-0.3
Prop. Noise High	0.3

A.4.6 Training Details

To regularize point-cloud features, we separately collect a dataset containing 59 pairs of matched point clouds in simulation and reality. One pair from them is visualized in Fig A.2. Student policies are trained by minimizing the loss in Sec. 2.2, where we set $\beta = 10^{-3}$. We use the Adam optimizer [112] with a learning rate of 10^{-4} during training. We periodically roll out student policies in simulation for 1,000 episodes. We then select the checkpoint that corresponds to the highest success rate to use as the base policy in the real-world learning stage.

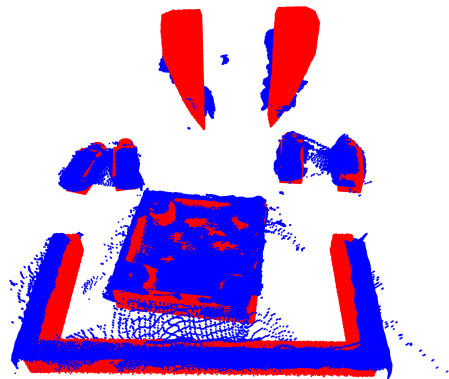


Figure A.2: **Visualization of paired point clouds in simulation (red) and reality (blue).**

B Real-World Learning Details

In this section, we provide details about real-world learning, including the hardware setup, human-in-the-loop data collection, and residual policy training.

B.1 Hardware Setup

As shown in Fig. A.3, our system consists of a Franka Emika 3 robot mounted on the tabletop. We use four fixed cameras and one wrist camera for point cloud reconstruction. They are three RealSense D435 and two RealSense D415. There is also a 3d-printed three-sided wall glued on top of the table to provide external support. We use a joint position controller from the Deoxys library [115] to control our robot at 1000 Hz.

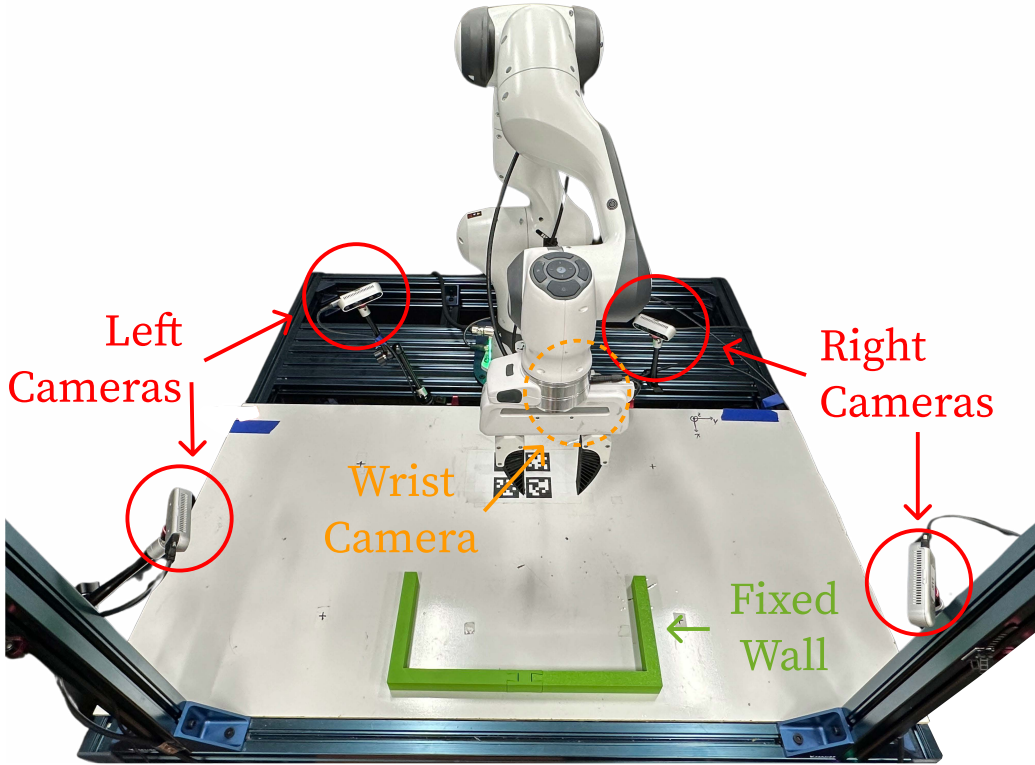


Figure A.3: **System setup.** Our system consists of a Franka Emika 3 robot mounted on the tabletop, four fixed cameras and one wrist camera (positioned at the rear side of the end-effector) for point cloud reconstruction, and a 3d-printed three-sided wall glued onto tabletop to provide external support.

B.2 Obtaining Point Clouds from Multi-View Cameras

We use multi-view cameras for point cloud reconstruction to avoid occlusions. Specifically, we first calibrate all cameras to obtain their poses in the robot base frame. We then transform captured point clouds in camera frames to the robot base frame and concatenate them together. We further perform cropping based on coordinates and remove statistical and radius outliers. To identify points belonging to the gripper so that we can add gripper semantic labels (Sec. A.4.2), we compute poses for two gripper fingers through forward kinematics. We then remove measured points corresponding to gripper fingers through K-nearest neighbor, given fingers' poses and synthetic point clouds. Subsequently, we add semantic labels to points belonging to the scene and synthetic gripper's point

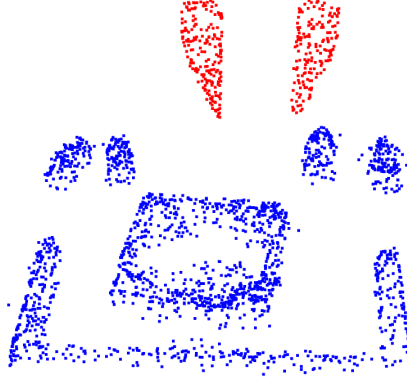


Figure A.4: **Visualization of real-world point-cloud observations.** We obtain them by 1) cropping point clouds fused from multi-view cameras based on coordinates, 2) removing statistical and radius outliers, 3) removing points corresponding to gripper fingers and replacing with synthetic point clouds through forward kinematics, 4) uniformly sampling without replacement, and 5) appending semantic labels to indicate gripper fingers (red) and the scene (blue).

966 clouds. Finally, we uniformly down-sample without replacement. We opt to not use farthest point
 967 sampling [116] due to its slow speed. One example is shown in Fig. A.4.

968 B.3 Human-in-the-Loop Data Collection

969 This data collection procedure is illustrated in Algorithm 1. As shown in Fig. A.5, we use a 3Dcon-
 970 nexion SpaceMouse as the teleoperation device. We design a specific UI (Fig. A.6) to facilitate the
 971 synchronized data collection. Here, the human operator will be asked to intervene or not. The oper-
 972 ator answers through keyboard. If the operator does not intervene, the base policy’s next action will
 973 be deployed. If the operator decides to intervene, the SpaceMouse is then activated to teleoperate
 974 the robot. After the correction, the operator can exit the intervention mode by pressing one button
 975 on the SpaceMouse. We use this system and interface to collect 20, 100, 90, and 17 trajectories
 976 with correction for tasks *Stabilize*, *Reach and Grasp*, *Insert*, and *Screw*, respectively. We use 90%
 977 of them as training data and the remaining as held-out validation sets. We visualize the cumulative
 978 distribution function of human correction in Fig. A.7.



Figure A.5: **Real workspace setup for human-in-the-loop data collection.** The human operator provides online correction through a 3Dconnexion SpaceMouse while monitoring the robot’s execution.

979 B.4 Residual Policy Training

980 B.4.1 Model Architecture

981 The residual policy takes the same observations as the base policy (Table A.VI). Furthermore, to
 982 effectively predict residual actions, it is also conditioned on base policy’s outputs. Its action head
 983 outputs eight-dim vectors, while the first seven dimensions correspond to residual joint positions

Algorithm 1: Human Intervention and Online Correction Data Collection

input : Base policy π^B , human policy π^H , real-world environment \mathcal{E} **output** : Human correction dataset \mathcal{D}^H **initialize:** $\mathcal{D}^H \leftarrow \emptyset$ $o \leftarrow \mathcal{E}.reset()$ **while** *not* $\mathcal{E}.terminated$ **do**▷ **deploy the base policy for one step** $a^B \leftarrow a^B \sim \pi^B(o)$ $o^{next} \leftarrow \mathcal{E}.deploy(a^B)$ ▷ **human decides intervention or not** $\mathbb{1}^H \leftarrow \pi^H.intervene(o, o^{next})$ **if** $\mathbb{1}^H$ **then** $\mathbf{q}^{pre} \leftarrow \mathcal{E}.robot_state$ ▷ **deploy human correction** $a^H \leftarrow a^H \sim \pi^H(o, o^{next})$ $o^{next} \leftarrow \mathcal{E}.deploy(a^H)$ $\mathbf{q}^{post} \leftarrow \mathcal{E}.robot_state$ ▷ **update dataset** $\mathcal{D}^H \leftarrow \mathcal{D}^H \cup (\mathbf{q}^{pre}, \mathbf{q}^{post}, \mathbb{1}^H, o)$ **end**▷ **update observation for the next step** $o \leftarrow o^{next}$ **end**

984 and the last dimension determines whether to negate base policy’s gripper action or not. Besides,
985 a separate intervention head predicts whether the residual action should be applied or not (learned
986 gated residual policy, Sec. 2.4).

987 For tasks *Stabilize* and *Insert*, we use a PointNet [81] as the point-cloud encoder. For tasks *Reach*
988 *and Grasp* and *Screw*, we use a Perceiver [82, 83] as the point-cloud encoder. Residual policies
989 are instantiated as feed-forward policies in all tasks. We use GMM as the action head and a simple
990 two-way classifier as the intervention head. Model hyperparameters are summarized in Table A.IX.

Table A.IX: Model hyperparameters for residual policies.

Hyperparameter	Value	Hyperparameter	Value
PointNet		Feature Fusion	
PointNet Hidden Dim	256	MLP Hidden Dim	512
PointNet Hidden Depth	2	MLP Hidden Depth	1
PointNet Output Dim	256	MLP Activation	ReLU
PointNet Activation	GELU	GMM Action Head	
Gripper Semantic Embd Dim	128	Hidden Dim	128
Perceiver		Hidden Depth	3
Perceiver Hidden Dim	256	Num Modes	5
Perceiver Number of Heads	8	Activation	ReLU
Perceiver Number of Queries	8	Intervention Head	
Gripper Semantic Embd Dim	128	Hidden Dim	128
Base Policy Action Conditioning		Hidden Depth	3
Base Policy Gripper Action Embd Dim	64	Activation	ReLU

```

(...)
system: need human intervention? (y/n)
user: n
      (deploying the next action)
system: need human intervention? (y/n)
user: y
      (correction through teleoperation)
system: exiting human intervention...
(...)

```

Figure A.6: The UI for synchronized human-in-the-loop data collection.

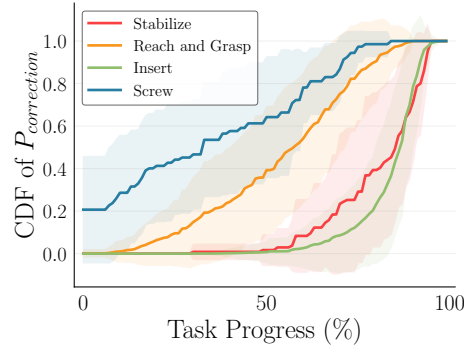


Figure A.7: **Cumulative distribution function (CDF) of human correction.** Shaded regions represent standard deviation. Human correction happens at different times across tasks. This fact necessitates TRANSIC’s learned gating mechanism.

991 B.4.2 Training Details

992 To train the learned gated residual policy, we first only learn the feature encoder and the action head.
 993 We then freeze the entire model and only learn the intervention head. We opt for this two-stage
 994 training since we find that training both action and intervention heads at the same time will result
 995 in sub-optimal residual action prediction. We follow the best practice for policy training, including
 996 using learning rate warm-up and cosine annealing [117]. Training hyperparameters are listed in
 997 Table A.X.

Table A.X: **Hyperparameters used in residual policy training.**

Hyperparameter	Value
Learning Rate	10^{-4}
Weight Decay	0
Learning Rate Warm Up Steps	1,000
Learning Rate Cosine Decay Steps	100,000
Minimal Learning Rate	10^{-6}
Optimizer	Adam

C Experiment Settings and Evaluation Details

In this section, we provide details about our experiment settings and evaluation protocols.

C.1 Task Definition

As shown in Fig. 3, we quantitatively benchmark four tasks. They are fundamental skills required to assemble a square table from FurnitureBench [84]. We randomize objects’ initial poses during evaluation.

- *Stabilize*: The robot pushes the square tabletop to the right corner of the wall such that it remains stable in following assembly steps.
- *Reach and Grasp*: The robot reaches and grasps the table leg. It needs to properly adjust the end effector’s orientation to avoid infeasible grasping poses.
- *Insert*: The robot inserts the pre-grasped table leg to the far right assembly hole of the tabletop.
- *Screw*: The robot’s end-effector is initialized close to an inserted table leg and it screws the table leg clockwise into the tabletop.

C.2 Main Experiments

We evaluate all methods on four tasks for 20 trials. Each trail starts with different objects and robot poses. We make our best efforts to ensure the same initial settings when evaluating different methods. Specifically, we take pictures for these 20 different initial configurations and refer to them when resetting a new trial. See Figs. A.15, A.16, A.17, A.18 for initial configurations of tasks *Stabilize*, *Reach and Grasp*, *Insert*, and *Screw*, respectively. We follow Liu et al. [90] to label reward for IQL. Full numerical results are provided in Table A.XI.

Table A.XI: **Success rates per tasks.** TRANSIC outperforms all baseline methods in all four tasks.

Tasks	TRANSIC	Direct Transfer	DR. & Data Aug. [52]	BC Fine-Tune	IQL Fine-Tune	HG-Dagger [65]	IWR [66]	BC [85]	BC-RNN [67]	IQL [68]
Stabilize	100%	10%	35%	55%	0%	65%	65%	40%	40%	5%
Reach and Grasp	95%	35%	60%	35%	0%	30%	40%	25%	0%	5%
Insert	45%	0%	15%	15%	25%	35%	40%	10%	5%	0%
Screw	85%	0%	35%	50%	65%	40%	40%	15%	25%	0%

C.3 Experiments with Different Sim-to-Real Gaps

C.3.1 Experiment Setup

We explain how different sim-to-real gaps are created.

Perception Error This is done by applying random jitter to 25% points from point clouds, which corresponds to adding noise in observation space \mathcal{O} . We test this sim-to-real gap on the task *Reach and Grasp*. As visualized in Fig. A.8, with probability $P = 0.6$, we apply random jitter to 25% points from the point-cloud observation. The jittering noise is sampled independently from the distribution $\mathcal{N}(0, 0.03)$. We clip the noise to be within the ± 0.03 range.

Underactuated Controller This is done by making the joint position controller less accurate, which corresponds to mismatched action space \mathcal{A} . We test this gap on the task *Insert*. We emulate an underactuated controller through early stopping. Concretely, at every time a new joint position goal \mathbf{q}_{goal} is set, we record the distance to the goal in configuration space $d_{\mathbf{q}} = \|\mathbf{q} - \mathbf{q}_{goal}\|$ and sample a factor $\Gamma \sim \mathcal{U}(0.80, 0.95)$. The controller will stop reaching the desired goal once it achieves Γ progress, i.e., stop early when $\|\mathbf{q} - \mathbf{q}_{goal}\| \leq (1 - \Gamma)d_{\mathbf{q}}$. Fig. A.9 visualizes the effect.

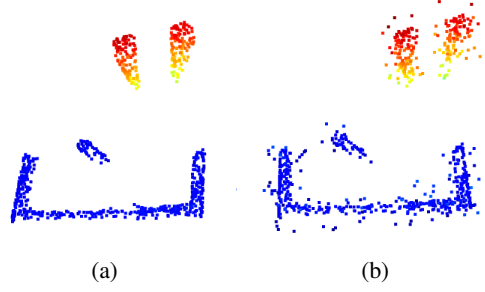


Figure A.8: **Visualization of introduced perception error.** **a)** The original point-cloud observation. **b)** The erroneous point-cloud observation with random jitter.

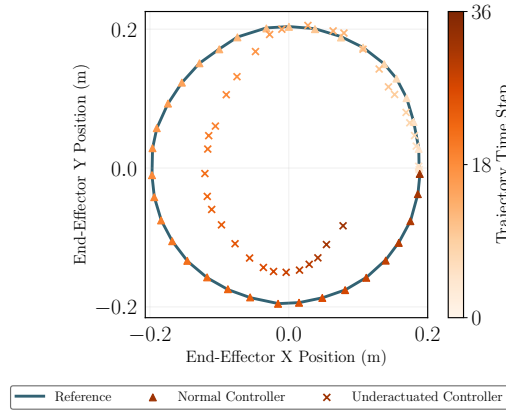


Figure A.9: **Visualization of the trajectory realized by an underactuated controller.** The plot displays the end-effector's position in the XY plane. It shows a reference circular movement, a trajectory tracked by the normal controller, and a trajectory tracked by the underactuated controller.

1033 **Embodiment Mismatch** This is done by changing the robot gripper to be shorter length as demon-
 1034 strated in Fig. A.10, which corresponds to discrepancy in state space \mathcal{S} and transition function \mathcal{T} .
 1035 We test this gap on the task *Screw*. We notice that the 9 cm length difference incurs a significant gap.

1036 **Dynamics Difference** This is done by changing object surfaces and increasing friction, which
 1037 corresponds to different transition function \mathcal{T} . We test this gap on the task *Stabilize*. Concretely, we

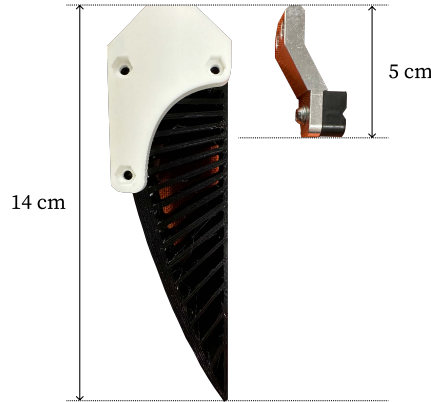


Figure A.10: **Two different gripper fingers used to create embodiment mismatch.** Policies are trained with the longer finger and tested on the shorter finger.

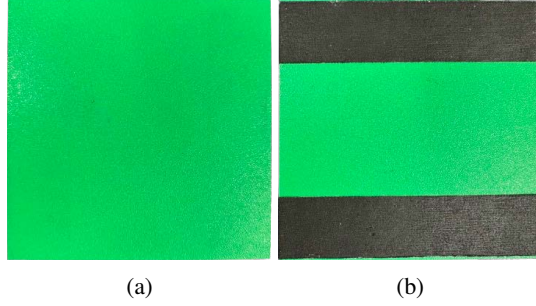


Figure A.11: **Two square tabletops used to create dynamics difference.** **a)** The original surface is smooth. **b)** We attach friction tapes to change the dynamics.

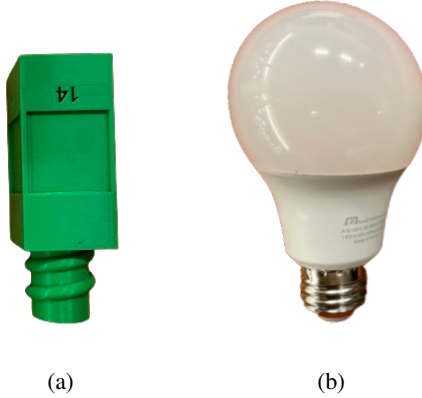


Figure A.12: **Two objects used to create asset mismatch.** **a)** Policies are trained with the table leg. **b)** We test policies with an unseen light bulb.

1038 attach friction tapes to the square tabletop’s surface to increase friction, hence change the dynamics
 1039 (Fig. A.11).

1040 **Object Assert Mismatch** As shown in Fig. A.12, this is done by replacing the table leg with a
 1041 light bulb, which corresponds to change in emitting function Ω . We test this gap on the task *Reach*
 1042 *and Grasp*.

1043 C.3.2 Evaluation

1044 We conduct 20 trails with different initial configurations. Initial conditions for first four experi-
 1045 ments are the same as main experiments (Figs. A.15, A.16, A.17, A.18). Fig. A.19 shows initial
 1046 configurations for the experiment *Object Asset Mismatch*.

1047 C.4 Data Scalability Experiments

1048 In Table A.XII, we show quantitative results for scalability with human correction dataset size on
 1049 four tasks.

1050 C.5 Ablation Studies

1051 C.5.1 Effects of Different Gating Mechanisms

1052 We introduce the learned gated residual policy in Sec. 2.4 where the gating mechanism controls
 1053 when to apply residual actions. To assess the quality of learned gating, we compare its performance
 1054 with an actual human operator performing gating. Results are shown in Table 1 (row “w/ Human

Table A.XII: Quantitative results for scalability with human correction dataset size on four tasks.

Method	Correction Dataset Size (%)				
	0	25	50	75	100
Stabilize					
TRANSIC	35%	80%	80%	100%	100%
IWR [66]		70%	75%	80%	65%
Reach and Grasp					
TRANSIC	60%	65%	80%	90%	95%
IWR [66]		60%	65%	40%	40%
Insert					
TRANSIC	5%	20%	35%	40%	45%
IWR [66]		5%	15%	30%	40%
Screw					
TRANSIC	35%	50%	65%	75%	85%
IWR [66]		20%	40%	40%	40%

Gating”). It is evident that the learned gating mechanism only incurs negligible performance drops compared to human gating. This suggests that TRANSIC can reliably operate in a fully autonomous setting once the gating mechanism is learned.

C.5.2 Policy Robustness

We investigate the policy robustness against 1) point cloud observations with inferior quality by removing two cameras, and 2) suboptimal correction data with noise injection. We remove two cameras and only keep three. Note that this is the same number of cameras as in FurnitureBench [84]. For tasks other than *Insert*, we keep the wrist camera, the right front camera, and the left rear camera. For the task *Insert*, we keep two front cameras and the left rear camera. We simulate suboptimal correction data by injecting noise into residual actions a^R . This noise is of large magnitude, which follows the normal distribution with zero mean and standard deviation corresponding to 5% of the largest residual action in the dataset. Results are shown in Table 1 (rows “Reduced Cameras” and “Noisy Correction”). We highlight that TRANSIC is robust to partial point cloud inputs caused by the reduced number of cameras. We attribute this to the heavy point cloud downsampling employed during training. Fishman et al. [118] echos our finding that policies trained with downsampled synthetic point cloud inputs can generalize to partial point cloud observations obtained in the real world without the need for shape completion. Meanwhile, when the correction data used to learn residual policies are suboptimal, TRANSIC only shows a relative decrease of 6% in the average success rate. We attribute this to the advantage of our integrated deployment—when the residual policy behaves suboptimally, the base policy could still compensate for the error in subsequent steps.

C.5.3 Consistency in Learned Visual Features

To learn consistent visual features between the simulation and reality, we propose to regularize the point cloud encoder during the distillation stage. As shown in Table 1 (row “w/o Regularization”), the performance significantly decreases without such regularization, especially for tasks that require fine-grained visual features. Without it, simulation policies would overfit to synthetic point cloud observations and hence are not ideal for sim-to-real transfer.

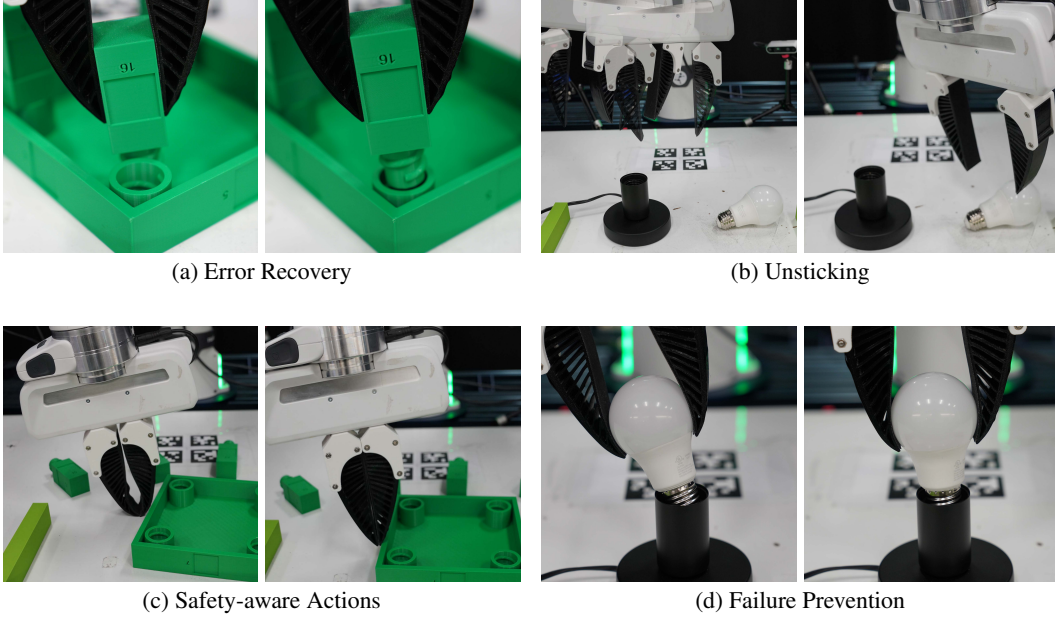


Figure A.13: **Emergent behaviors learned by TRANSIC.** **a) Error recovery.** Left: The robot tries to insert the table leg but the direction is wrong; Right: TRANSIC raises the end effector and moves to the correct insertion position. **b) Unsticking.** Left: The robot hovers for a while and never reaches the light bulb; Right: TRANSIC helps the robot get unstuck and move to the bulb. **c) Safety-aware actions.** Left: When pushing the tabletop, the gripper is too low and bends. This might damage the robot; Right: TRANSIC compensates for the command that causes the end effector to move too low. **d) Failure prevention.** Left: The light bulb will fall and break after gripper opening; Right: TRANSIC adjusts the bulb to a stable pose to prevent failure.

1081 C.6 Qualitative Analysis and Emergent Behaviors

1082 We examine the distribution of the collected human correction dataset. During the human-in-the-
 1083 loop data collection, the probability of intervening and correcting is reasonably low ($P_{\text{correction}} \approx$
 1084 0.20). This is consistent with our intuition that, with a good base policy, interventions are not neces-
 1085 sary for most of the time. However, they become critical when the robot tends to behave abnormally
 1086 due to unaddressed sim-to-real gaps. Moreover, as highlighted in Fig. A.7, interventions happen
 1087 at different times across tasks. This fact renders heuristics-based methods [119] for deciding when
 1088 to intervene difficult, and further necessitates our learned residual policy. Several representative
 1089 behaviors learned by TRANSIC are demonstrated in Fig. A.13.

D Additional Experiment Results and Discussions

D.1 Empirical Justifications for *Action Space Distillation*

Reasons for the proposed *action space distillation* are twofold.

The first is mainly because an OSC is hard to sim-to-real transfer, while a joint position controller can be seamlessly transferred. As suggested in Nakanishi et al. [73], an OSC requires accurate modeling of robot parameters, such as the task-space inertia matrix and gravity compensation. System identification helps but is insufficient. Furthermore, it is often the case that given the same joint torque, the end-effector moves differently in simulation and the real world. Because an OSC uses a task-space error to compute joint torques, this will lead to large joint position deviation.

The second is for better training efficiency. As shown in Fig. A.14, it is almost impossible to directly train RL with point cloud inputs and joint position action space. Even after 7-day training, RL still shows no sign of improvement. In contrast, TRANSIC takes around 3 days to train on NVIDIA GeForce RTX 3090 GPUs. Therefore, the distillation is important to make the training feasible.

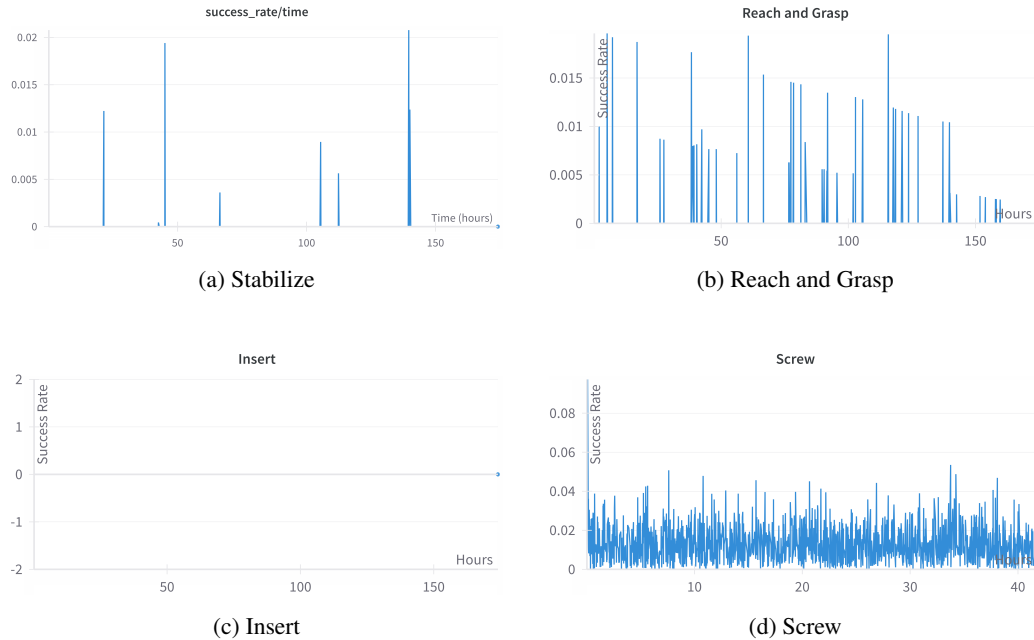


Figure A.14: Learning curves for RL with point-cloud observations and joint position actions.

D.2 Distilling Simulation Base Policy with Diffusion Policy

We experiment with learning simulation base policies (Sec. 2.2) with the Diffusion Policy [105]. Concretely, when performing *action space distillation* to learn student policies, we replace the Gaussian Mixture Model (GMM) action head with the Diffusion Policy. Proper data augmentation (Table A.VIII) is also applied to robustify learned policies. Hyperparameters are provided in Table A.XIII.

Table A.XIII: Diffusion Policy hyperparameters.

Hyperparameter	Value	Hyperparameter	Value
Architecture	UNet	T_o	2
UNet Hidden Dims	[64, 128]	T_a	8
UNet Kernel Size	5	T_p	16
UNet GroupNorm Num Groups	8	Num Denoising Steps (Train)	100
Diffusion Step Embd Dim	128	Num Denoising Steps (Eval)	16

The comparison between GMMs on the real robot is shown in Table A.XIV. We highlight two findings. First, the significant domain difference between simulation and reality generally exists regardless of different policy modeling methods. Second, since the Diffusion Policy plans and executes a future trajectory, it is more vulnerable to simulation-to-reality gaps due to planning inaccuracy and the consequent compounding error. Only executing the first action from the planned trajectory and re-planning at every step may help, but the inference latency renders the real-time execution infeasible.

Table A.XIV: **The real-robot performance difference between GMM and Diffusion Policy.** The policy error caused by simulation-to-reality gaps will be amplified by the Diffusion Policy because it plans and executes a future trajectory.

	Average	Stablize	Reach and Grasp	Insert	Screw
GMM	33.7%	35%	60%	5%	35%
Diffusion Policy	22.5%	35%	50%	5%	0%

D.3 Gating Mechanism Conceptual Comparison

Recall several design choices in the proposed gating mechanism: 1) takes inputs of unstructured sensory observations (point cloud); 2) conditioned on base policy’s outputs for effective prediction; 3) the intervention classifier shares the same feature encoder with the residual policy; and 4) the entire pipeline is learned end-to-end. We contrast against several mechanisms from the literature.

Table A.XV: **Gating mechanism conceptual comparison.**

	How to decide apply gating or not	Input	Condition on base policy’s outputs	Shared feature encoder
Ours	End-to-end learned	Point cloud and proprioception	Yes	Yes
Residual Policy Learning [78]	No gating	Low-dimensional state	No	No
Residual RL [77]	No gating	Low-dimensional state	No	No
ThriftyDAgger [119]	Thresholded based on neural network ensemble	Low-dimensional state	No	No
Runtime Monitoring [103]	End-to-end learned	RGB and proprioception	No	Yes

D.4 Long-Horizon Tasks Statistics

We show statistics about task length from FurnitureBench [84] in Table A.XVI.

Table A.XVI: **Statistics about long-horizon tasks from FurnitureBench [84].**

	Number of Steps	Average Human Demo Length
Lamp	594	2 Minutes
Square Table	1689	6 Minutes

1123 E Extended Preliminaries

1124 E.1 Problem Formulation

1125 We formulate a robot manipulation task as an infinite-horizon discrete-time Partially Observable
 1126 Markov Decision Process (POMDP) $\mathcal{M} := (\mathcal{S}, \mathcal{O}, \Omega, \mathcal{A}, \mathcal{T}, R, \gamma, \rho_0)$, where \mathcal{S} is the state space,
 1127 \mathcal{O} is the observation space, and \mathcal{A} is the action space. At time step t , a robot observes $o_t \in \mathcal{O}$
 1128 emitted from observation function $\Omega(o_t | s_t, a_{t-1}) : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{O}$, executes an action a_t , and receives
 1129 a scalar reward r_t from the reward function $R(s_t, a_t) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. The environment proceeds
 1130 to the next state governed by the transition function $\mathcal{T}(s_{t+1} | s_t, a_t) : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. The robot
 1131 learns a parameterized policy $\pi_\theta(\cdot | o) : \mathcal{O} \rightarrow \Delta \mathcal{A}$ to maximize the expected discounted return
 1132 $\mathcal{J} := \mathbb{E}_{\tau \sim p_{\pi_\theta}} [\sum_{t=0}^{\infty} \gamma^t r_t]$ over induced trajectory distribution $\tau := (s_0, o_0, a_0, r_0, \dots) \sim p_{\pi_\theta}$, where
 1133 $s_0 \sim \rho_0$ is sampled from the initial state distribution. Additionally, $\gamma \in [0, 1]$ is a discount factor.
 1134 In this work, we model simulation and real environments as two different POMDPs.

1135 E.2 Intervention-Based Policy Learning

1136 We adopt an intervention-based learning framework [65, 66, 90] where a human operator can inter-
 1137 vene and take control during the execution of the robot base policy π_B . Denote the human policy as
 1138 π_H , the following combined policy is deployed during data collection:

$$\pi^{\text{deployed}} = \mathbb{1}^H \pi^H + (1 - \mathbb{1}^H) \pi^B, \quad (\text{A.7})$$

1139 where $\mathbb{1}^H$ is a binary function indicating human interventions. Introducing a trajectory distribution
 1140 $q(\tau)$ that consists of two observation-action distributions generated by the robot ρ^B and human
 1141 operator ρ^H , the original RL objective leads to the maximization of a variational lower bound on
 1142 logarithmic return [66, 120]:

$$\mathcal{J}(\theta, q) = \mathbb{E}_{q(\tau)} [\log R(\tau) + \log p_{\pi_\theta} - \log q(\tau)], \quad (\text{A.8})$$

1143 where p_{π_θ} is the induced trajectory distribution. While the human operator optimizes Eq. A.8
 1144 through intervention and correction, the robot learner maximizes it through

$$\theta = \arg \max_{\theta \in \Theta} \mathbb{E}_{(o,a) \sim q(\tau)} [\log \pi_\theta(a | o)]. \quad (\text{A.9})$$

1145 Various intervention-based policy learning methods have been derived by weighting observation-
 1146 action pairs in Eq. A.9 differently. For example, HG-Dagger [65] completely ignores robot data \mathcal{D}^B
 1147 and only trains on human data \mathcal{D}^H that contain intervention samples. This is equivalent to $q(\tau) \propto$
 1148 ρ^H . Intervention Weighted Regression (IWR) [66] balances the data distribution by emphasizing
 1149 human intervention: $q(\tau) \propto \alpha \rho^H + \rho^B$ with $\alpha = |\mathcal{D}^B|/|\mathcal{D}^H|$. Non-intervention-based methods
 1150 such as traditional behavior cloning (BC) [85] only learn on \mathcal{D}^H with full human demonstrations
 1151 instead of intervention. This effectively sets $q(\tau) \propto \rho^H$.

F Extended Related Work

Robot Learning via Sim-to-Real Transfer Physics-based simulations [6–10, 49, 121–123] have become a driving force [1, 2] for developing robotic skills in tabletop manipulation [124–127], mobile manipulation [128–131], fluid and deformable object manipulation [132–135], dexterous in-hand manipulation [13–17], locomotion with various robot morphology [18–26, 136], object tossing [79], acrobatic flight [28, 29], etc. However, the domain gap between the simulators and the reality is not negligible [10]. Successful sim-to-real transfer includes locomotion [18–27], in-hand re-orientation for dexterous hands where objects are initially placed near the robot [13–17], and non-prehensile manipulation limited to simple tasks [30–39]. In this work, we tackle more challenging sim-to-real transfer for complex manipulation tasks and successfully demonstrate that our approach can solve sophisticated contact-rich manipulation tasks. More importantly, it requires significantly fewer real-robot data compared to the prevalent imitation learning and offline RL approaches [67, 68, 85]. This makes solutions that are based on simulators and sim-to-real transfer more appealing to roboticists.

Sim-to-Real Gaps in Manipulation Tasks Despite the complex manipulation skills recently learned with RL in simulation [137], directly deploying learned control policies to physical robots often fails. The sim-to-real gaps [10, 40, 44, 138] that contribute to this performance discrepancy can be coarsely categorized as follows: **a)** perception gap [18, 41–43], where synthetic sensory observations differ from those measured in the real world; **b)** embodiment mismatch [18, 44, 45], where the robot models used in simulation do not match the real-world hardware precisely; **c)** controller inaccuracy [46–48], meaning that the results of deploying the same high-level commands (such as in configuration space [139] and task space [140]) differ in simulation and real hardware; and **d)** poor physical realism [49], where physical interactions such as contact and collision are poorly simulated [86].

Although these gaps may not be fully bridged, traditional methods to address them include system identification [18, 30, 50, 51], domain randomization [13, 52–54], real-world adaptation [55], and simulator augmentation [57–59]. However, system identification is mostly engineered on a case-by-case basis. Domain randomization suffers from the inability to identify and randomize all physical parameters. Methods with real-world adaptation, usually through meta-learning [87], incur potential safety concerns during the adaptation phase. Most of these approaches also rely on explicit and domain-specific knowledge about tasks and the simulator *a priori*. For instance, to perform system identification for closing the embodiment gap for a quadruped, Tan et al. [18] disassembles the physical robot and carefully calibrates parameters including size, mass, and inertia. Kim et al. [32] reports that collaborative robots, such as the commonly used Franka Emika robot, have intricate joint friction that is hard to identify and randomized in typical physics simulators. To make a simulator more akin to the real world, Chebotar et al. [39] deploys trained virtual robots multiple times to refine the distributions of simulation parameters. This procedure not only introduces a significant real-world sampling effort, but also incurs potential safety concerns due to deploying suboptimal policies. In contrast, our method leverages human intervention data to implicitly overcome the transferring problem in a domain-agnostic way and also leads to safer deployment.

Human-in-The-Loop Robot Learning Human-in-the-loop machine learning is a prevalent framework to inject human knowledge into autonomous systems [61, 88, 89]. Various forms of human feedback exist [62], ranging from passive judgement, such as preference [141–150] and evaluation [151–156], to active involvement, including intervention [157–159] and correction [160, 161]. They are widely adopted in solutions for sequential decision-making tasks. For instance, interactive imitation learning [65, 66, 90, 162] leverages human intervention and correction to help naïve imitators address data mismatch and compounding error. In the context of RL, reward functions can be derived to better align agent behaviors with human preferences [144, 147, 148, 151]. Noticeably, recent trend focuses on continually improving robots’ capability by iteratively updating and deploying policies with human feedback [90], combining active human involvement with RL [161], and autonomously generating corrective intervention data [91]. Our work further extends this trend

1202 by showing that sim-to-real gaps can be effectively eliminated by using human intervention and
1203 correction signals.

1204 In shared autonomy, robots and humans share the control authority to achieve a common goal [63,
1205 64, 92–94]. This control paradigm has been largely studied in assistive robotics and human-robot
1206 collaboration [95–97]. In this work, we provide a novel perspective by employing it in sim-to-real
1207 transfer of robot control policies and demonstrating its importance in attaining effective transfer.

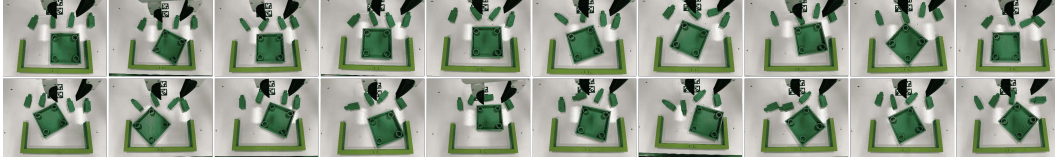


Figure A.15: Initial settings for evaluating the task *Stabilize*.

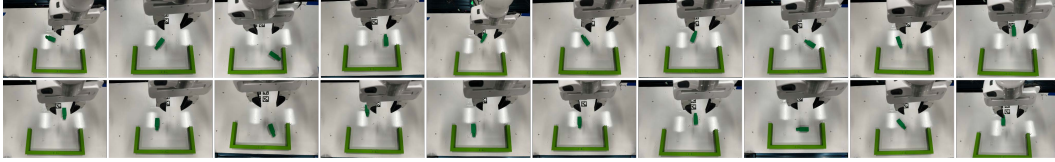


Figure A.16: Initial settings for evaluating the task *Reach and Grasp*.

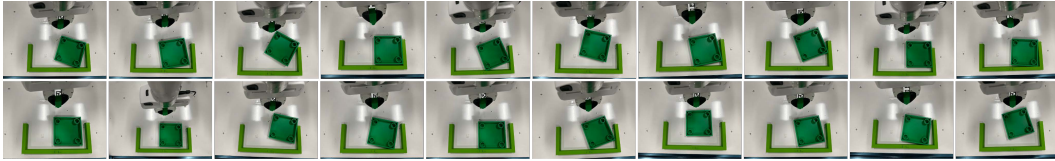


Figure A.17: Initial settings for evaluating the task *Insert*.

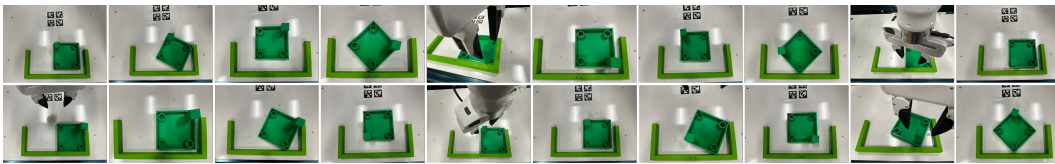


Figure A.18: Initial settings for evaluating the task *Screw*.

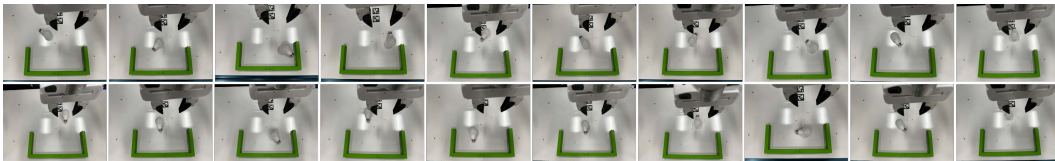


Figure A.19: Initial settings for the experiment *Object Asset Mismatch*.