From Imitation to Refinement – Residual RL for Precise Visual Assembly

Lars Ankile,[1,2,3] **Anthony Simeonov,**[1,2] **Idan Shenfeld,**[1,2]
Marcel Torne[1,2], **Pulkit Agrawal**[1,2]

[1]Massachusetts Institute of Technology [2]Improbable AI Lab
[3]Harvard University

Abstract: Behavior cloning (BC) currently stands as a dominant paradigm for learning real-world visual manipulation. However, in tasks that require locally corrective behaviors like multi-part assembly, learning robust policies purely from human demonstrations remains challenging. Reinforcement learning (RL) can mitigate these limitations by allowing policies to acquire locally corrective behaviors through task reward supervision and exploration. This paper explores the use of RL fine-tuning to improve upon BC-trained policies in precise manipulation tasks. We analyze and overcome technical challenges associated with using RL to directly train policy networks that incorporate modern architectural components like diffusion models and action chunking. We propose training residual policies on top of frozen BC-trained diffusion models using standard policy gradient methods and sparse rewards. Our experimental results demonstrate that this residual learning framework can significantly improve success rates beyond the base BC-trained models in high-precision assembly tasks by learning corrective actions. We also show that by combining our residual learning approach with teacher-student distillation and visual domain randomization, our method can enable learning real-world policies for robotic assembly directly from RGB images.

Keywords: Residual Learning, Robotic Assembly, Combining BC and RL

1 Introduction

Despite significant advancements in robot learning, autonomous systems still struggle with manipulation tasks requiring high precision, robustness to a wide array of initial conditions, and operation directly from RGB sensors. An example of such a challenging scenario is autonomous assembly, where a robot is presented with a set of multiple parts to pick up and precisely fit together. Such systems typically utilize engineered fixtures for consistent initial part positioning, making assembly robots expensive and impractical to adapt across diverse tasks. Learning to perform these tasks from vision can improve the adaptability and robustness of such systems, but the high-precision, contact-rich nature of such tasks poses substantial challenges for current robotic learning frameworks.

Behavior cloning (BC), has re-emerged as a popular technique for acquiring manipulation control policies [1, 2, 3, 4, 5, 6]. This is largely due to the ease of directly specifying complex tasks by providing demonstrations. However, BC requires large amounts of costly demonstration data to enable generalization to different scene configurations [5]. BC-trained policies are also often limited by the nature of human demonstrations that *directly* achieve the task. This can mean such policies never learn to perform *corrective* actions that recover from deviations that inevitably occur during deployment. Reinforcement learning (RL) offers a complementary paradigm, wherein locally corrective behavior can be learned by exploring and discovering action sequences that lead to downstream task success.

Although RL offers appealing advantages, many of its successes in robotics depend on carefully crafted reward functions that may sacrifice generality across tasks [7, 8, 9]. A common mitigation is to bootstrap the RL process by starting with a small set of demonstrations [10, 11, 12] – by initializing with a policy that occasionally solves the task, supervision from simple sparse rewards can be enough for RL to progress effectively. However, naively trained BC policies may fail to achieve any meaningful success with a small number of demonstrations in complex tasks like assembly [13]. Innovations in policy design, such as diffusion models [14] and action chunking [6], can alleviate some of this difficulty and enable non-zero success rates for challenging tasks using a modest number of demonstrations [13, 15]. However, introducing these modern architectural components makes it difficult to perform fine-tuning with RL. For instance, training diffusion with RL can require complex reformulation of the de-noising process to enable policy gradient estimation [16]. We also find that action chunking tends to make direct RL fine-tuning harder and unstable due to the expanded action space dimension induced by chunking.

In this paper, we propose adaptations to the BC + RL pipeline that facilitate its application to complex tasks like furniture assembly [15]. Our key contribution is to enable RL fine-tuning to work with modern advancements like diffusion policies and chunked action predictions. We first provide an analysis and empirical evidence supporting the need for action chunking and expressive policy architectures for achieving success rates that are high enough to bootstrap RL. We then propose training residual policies [17, 18, 19] with on-policy RL and sparse task rewards to improve upon pure BC. These residual models learn to predict corrective actions that more reliably achieve high precision. In this way, we side-step the aforementioned challenges of directly altering the base policy with RL, while improving overall task success. Finally, by distilling behaviors learned with RL into a large synthetic dataset of trajectories and combining them with a limited set of real-world demonstrations, we showcase the benefits of our residual learning approach in enabling real-world assembly policies that operate directly from RGB images.

2 Problem Setup and Approach Overview

Our goal is to develop a pipeline that enables robots to perform precise manipulation tasks without requiring massive amounts of manual human effort. While our pipeline can be applied to many tasks, we focus on applications to the domain of multi-step assembly directly from RGB images. Here, we introduce the specific assumptions of our problem setup and detail the main ideas of the pipeline that we propose.

Assumptions and System Components We assume an assembly task is specified to the robot via a mid-sized ($\sim$50-90) set of expert demonstrations. The tasks each have an underlying success criterion depending on a set of required alignments between the parts. Each task consists of a fixed set of rigid object parts for which we have access to CAD models, such that both the parts and the entire assembly process can be accurately simulated. Each task may have long horizons ($\sim$750-1000 steps) and require sequencing of behaviors such as corner alignment, 6-DoF grasping, reorientation, insertion, and screwing. Multi-part assembly interactions are simulated using the SDF-based collision geometry representations featured in the Factory [20] extension of NVIDIA's Isaac Gym simulator. We use the tasks `one_leg`, `round_table`, and `lamp` from the FurnitureBench [15] task suite.

Preliminaries. We formulate the robot's task as a discrete-time sequential decision-making problem. In each time step t, the robot receives either an observation $o_t \in \mathcal{O}$ if it operates in the real world or the state of the system $s_t \in \mathcal{S}$ if it operates in simulation. After receiving it, the agent produces an action a_t to execute in the environment. The action space in both simulation and the real world is the desired end-effector pose $\mathbf{T}^{\text{des}} \in \text{SE}(3)$. We use a differential inverse kinematics controller to convert the desired end-effector pose commands into joint position targets, which are tracked with a low-level PD controller. The real world observation space $\mathcal{O}$ contains the robot end-effector pose $\mathbf{T} \in \text{SE}(3)$, robot end-effector spatial velocity $\mathbf{V} \in \mathbb{R}^6$, the gripper width w_{g}, and RGB images from a fixed front-view camera ($I^{\text{front}} \in \mathbb{R}^{h \times w \times 3}$) and a wrist-mounted camera ($I^{\text{wrist}} \in \mathbb{R}^{h \times w \times 3}$), each with unknown camera poses. In the simulated task variants, the system

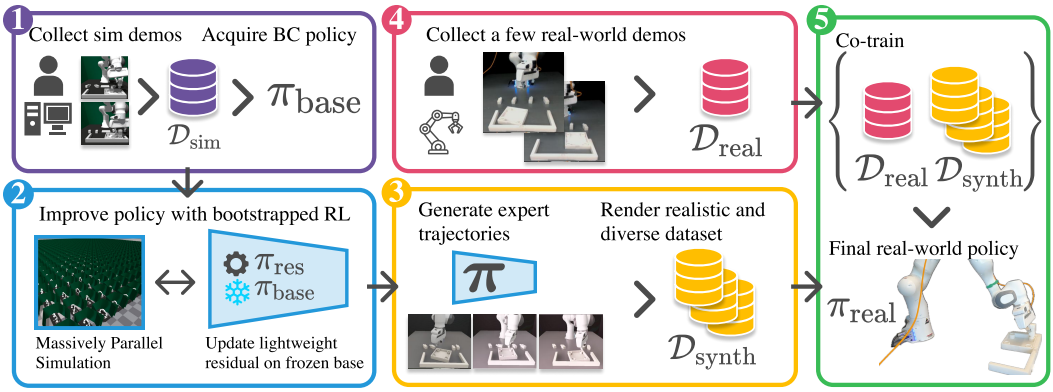

Figure 1: **Pipeline overview. (1)** Beginning with a policy trained with BC in simulation, **(2)** we train residual policies to improve task success rates with RL and sparse rewards. **(3)** We then distill the resulting behaviors to a policy that operates on RGB images. **(4)** By combining synthetic data with a small set of real demonstrations, **(5)** we deploy assembly policies that operate from RGB images in the real world.

states space $\mathcal{S}$ contains the same end-effector pose $\mathbf{T}$, spatial velocity $\mathbf{V}$, and gripper width w_{g}, along with the 6-DoF poses of all the parts in the environment $\{\mathbf{T}^{\text{part}_i}\}_{i=1}^{\text{num\_parts}}$. In addition to the state/observation, the simulated agent receives a binary reward signal from the environment, indicating whether two parts of the assembly have achieved their required geometric alignment.

BC + RL Teacher-Student Distillation Pipeline The pipeline's high-level goal is to leverage a small amount of human demonstrations and the availability of a simulator to develop a vision-based manipulation policy. To this end, the pipeline consists of two phases:

- **Phase one:** We train an agent to perform the task in a simulated environment using RL. To overcome the need for manual reward engineering, we start with an initial policy pretrained with behavior cloning on a limited set of demonstrations collected in the simulation. We then fine-tune the policy with RL, with only a sparse reward signal provided upon successful task completion.
- **Phase two:** Next, we distill the fine-tuned RL expert policy into a dataset of high-quality RGB image observations and corresponding actions for vision-based policy distillation (using BC). This dataset is combined with a small set of real-world task demonstrations, which helps bridge the simulation-to-reality gap for the final vision-based manipulation policy.

3 Methods

3.1 Imitation Learning

The first step in the pipeline is to use a small number of human demonstrations to train a "base" policy that serves as the starting point for RL training. For each task, we collect a dataset of 50 demonstrations in simulation, $\mathcal{D}_{\text{sim}}$. This dataset contains trajectories, e.g., $\mathcal{D} = \{\tau_1, ..., \tau_N\}$, where each trajectory contains the system states s_t, and robot actions a_t, i.e., $\tau_i = \{(s_t, a_1), ..., (s_T, a_T)\}$, with T being the trajectory length. We obtain these trajectories by teleoperating the simulated robot to complete the task.

Using the simulation dataset $\mathcal{D}_{\text{sim}}$, we train the base policy π_{base} with Behavior Cloning (BC), i.e., we maximize the likelihood of the data by optimizing $\max_\pi \mathbb{E}_{(a_t,s_t)\sim\mathcal{D}_{\text{sim}}}[\log \pi(a_t|s_t)]$. We use Diffusion Policy [14] as the base model, which has shown strong empirical performance in handling difficult manipulation tasks with relatively small datasets [13]. Consistent with recent advancements [14, 6, 13], our policy framework enhances its performance by predicting multiple future actions in chunks, as opposed to individual actions at each timestep. We denote the length of future

action sequences predicted by the policy as T_a, the output as $\mathbf{a_t} = [a_t, ..., a_{t+T_a}]$. When predicting an action chunk $\mathbf{a_t}$ of length T_a, we only execute a subset $[a_t, ..., a_{t+T_{\text{exec}}}]$, with execution horizon $T_{\text{exec}} \leq T_a$.

3.2 Online Reinforcement Learning with Residual Policies

Given the initial base policy π_{base}, we aim to improve it using RL. However, directly fine-tuning diffusion models with RL is an active area of research [21, 22, 16, 23], made difficult because of the multi-step inference process and unavailability of the policy action log-probabilities. Another category of methods upweight high-quality model outputs via importance sampling, return conditioning, or augmenting the original de-noising objective with a loss term for maximizing a Q-function [24, 25, 26, 27, 28]. However, these methods mainly enable better *extraction* of high-quality behavior in the data, while we are more concerned with learning new, *corrective* behaviors. Action chunks can also make optimization with policy gradients more difficult. This is partly due to chunking increasing the effective action space (e.g., chunks of 8 actions increase the dimension by $\times 8$), which we find to increase RL training instability (see Sec. 4.1). This means fine-tuning other popular BC architectures, like the Action-Chunked Transformer (ACT) [6], also brings technical challenges. Furthermore, recent work has shown that fine-tuning large pre-trained models can lead to forgetting of capabilities if the agent does not visit states seen during pre-training frequently enough [29].

We side-step these complications by training a residual Gaussian MLP policy π_{res}. This policy takes as input both the system state and the action predicted by the diffusion policy π_{base} and produces an "action correction" that modifies the action as $a_t = a_t^{\text{base}} + \alpha \cdot a_t^{\text{res}}$, with $\alpha \leq 1$ being a coefficient controlling what scale the residual policy operates on. We denote the resulting combined policy $\boldsymbol{\pi}$. This decoupling has several advantages. First, we can regard the diffusion model as a black box and not change its parameters [17][1]. This also allows the use of different prediction horizons T_a for the two policies. This flexibility in action horizon is helpful as most RL algorithms optimize single-action policies. It also removes the need to explicitly regularize the fine-tuned policy to stay near the pre-trained policy, which is often necessary to achieve stable optimization [30, 31].

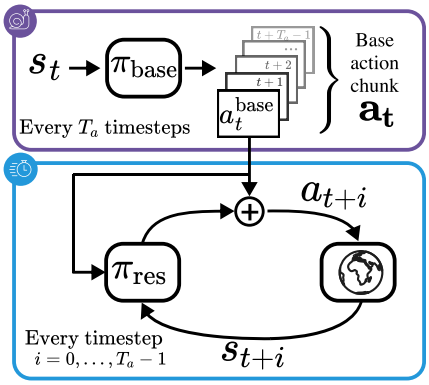

Figure 2: Per-timestep residual policies trained with PPO to locally correct action chunks predicted by a BC-trained diffusion policy.

Given the above, we train the π_{res} using standard PPO [32] with the alternation that we augment the state space with the base action. The base model observes the current state s_t and outputs a chunk of actions $\mathbf{a_t}$. For each action in the chunk, we concatenate it with the current observation $s_{t+i}^{\text{res}} = [s_{t+i}, a_{t+i}^{\text{base}}]$, and predict the correction $a_{t+i}^{\text{res}} \sim \pi^{\text{res}}(\cdot | s_{t+i}^{\text{res}})$ for $i = 0, ..., T_a - 1$. Besides the technical convenience, one can look at residual policy as a way to incorporate an inductive bias that the policy should only learn *locally* corrective actions [18]. This is motivated by our qualitative observations that most of the base BC policy failures are due to slight imprecision near "bottleneck" transitions when performing skills like grasping and insertion (see Fig. 3).

3.3 Distillation for RGB-based Visual Policy

Once the RL training has converged, the policy typically exhibits significantly higher success rates compared to the initial base policy. Our objective is to distill this enhanced performance

[1]This decoupling also implies that the ACT or any other strong BC or offline RL method can straightforwardly serve as the base model without changing anything else in the pipeline, an advantage of residual policy learning emphasized by other prior work in robotics [17, 18].

from the state-based policy into a vision-based policy that operates solely on RGB images. Following the established teacher-student distillation paradigm [9, 33, 7], we generate a dataset of successful trajectories, $\mathcal{D}_{\text{synth}}$, where the environment's observations, o_t, replace the states, i.e., $\mathcal{D}_{\text{synth}} = \{\tau_{\text{synth},1}, ..., \tau_{\text{synth},N}\}$ and $\tau_{\text{synth},i} = \{(o_1, a_1), ..., (o_T, a_T)\}$. We utilize this dataset to train a final image-conditioned diffusion policy with BC.

For real-world transfer, we enhance the synthetic dataset $\mathcal{D}_{\text{synth}}$ by re-rendering its trajectories in Isaac Sim. This process improves image quality and introduces variability in environmental conditions such as object and table colors, textures, lighting, and camera perspectives, many of which cannot easily be done with standard image augmentation techniques. We denote this refined dataset as $\mathcal{D}_{\text{synth-render}}$. To ease the difficulty of zero-shot sim-to-real with RGB images, we opt for a co-training approach, integrating the synthetic dataset with a small set of real-world task demonstrations, $\mathcal{D}_{\text{real}}$, which similarly comprises only environmental observations without ground truth poses. This combined dataset, $\mathcal{D}_{\text{real}} \cup \mathcal{D}_{\text{synth-render}}$, is used to train the final student policy through BC. While loss reweighting or upsampling may be helpful given the imbalanced dataset sizes, we find that simply mixing the two datasets still provides performance improvement beyond training purely with real data (see Fig. 2).

4 Experiments and Results

In Sec. 4.1, we investigate the requirement of complex BC methods and the impact of our residual reinforcement learning approach on improving the success rates of policies trained with imitation learning. Next, in Sec. 4.2, we study the performance of downstream distillation from synthetic RL data, examining the relationship between the quantity and quality of the data and the resulting performance of the distilled vision-based policies. Finally, in Sec. 4.3 we demonstrate our approach enabling precise assembly tasks on a physical robot directly from vision.

4.1 Improving Imitation Learning with Online Residual Learning

Why Action Chunking and Diffusion Policies? Simple feed-forward MLPs of modest size have shown impressive performance in many domains when trained with RL [7, 33, 34], and offer a natural starting point for RL fine-tuning after BC pre-training. However, we find that MLP policies trained to predict single actions (MLP-S) completely fail across all tasks we consider. Therefore, we also trained MLP policies with action chunking (MLP-C). When we introduce chunking, MLP performance improves drastically. However, we also find that the more complex diffusion policy architecture generally outperforms MLPs, especially in tasks of intermediate difficulty. For example, an improvement from 10% success rate to 29% for the `one_leg` task on medium randomness, making subsequent fine-tuning far easier. In one case, `lamp` on low randomness, MLP-C outperformed DP. In qualitative evaluations, we find that DP has smoother and faster actions, which generally is beneficial, but in this case, seems to hurt performance, as it tends to retract before the gripper closes. We also find that all methods struggle for the hardest tasks, i.e., on which MLP-C and DP both achieve less than 5% success rate, indicating that there is still a need for stronger BC methods.

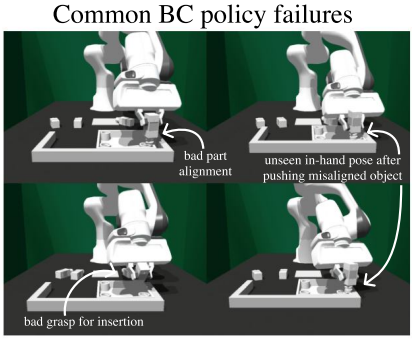

Figure 3: Representative failures from BC policies. RL-trained residual policies dramatically improve succcess rate by correcting for these errors.

Online Policy Fine-Tuning Comparison Next, we investigate the effectiveness of our residual RL approach in improving the performance of policies trained with BC. We train residual policies using PPO [32] on top of the diffusion policy and compare it to two baselines – (1) directly fine-tune the MLP-C with PPO while regarding every action chunk as one concatenated action. (2) Using the

recently proposed IDQL algorithm [24] where one samples multiple times from the diffusion policy and chooses which action chunk to execute based on its Q-value. We have found that learning a good Q-value estimator from offline data alone, as proposed in IDQL, is challenging with only 50 demonstrations. Therefore, we learn it in on-policy matter similar to [35].

The results in Tab. 1 show that our residual RL approach significantly improves success rates over BC and outperforms alternative RL fine-tuning methods. For tasks with lower initial randomization, such as the `one_leg` task, the residual policy increases the success rate from 55% to 95%. However, for tasks with higher part randomization, we observe performance saturates at lower success rates. We hypothesize this is because the residual policy is limited by the performance of the base model.

Methods	One Leg		Round Table		Lamp	
	Low	Med	Low	Med	Low	Med
IL						
MLP-S	0	0	0	0	0	0
MLP-C	45	10	5	2	**8**	1
DP	**54**	**29**	**12**	**4**	7	**2**
RL						
MLP-C + PPO	70	28	38	6	32	2
DP + IDQL	52	13	18	3	11	1
DP + Residual PPO	**95**	**68**	**55**	**10**	**60**	**8**

Table 1: **Top** BC-trained MLPs without chunking (MLP-S) cannot perform any of the tasks, and Diffusion Policies (DP) generally outperform MLPs with chunking (MLP-C). **Bottom** Training our proposed residual policies with RL on top of frozen diffusion policies performs the best among all evaluated fine-tuning techniques.

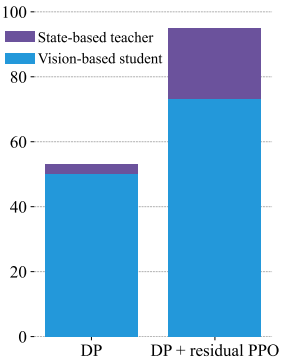

Figure 4: Comparison of distilled performance from BC and RL-based teacher.

Fig. 3 shows qualitative examples of common failure modes made by BC policies that are regularly corrected by the RL policy. For instance, a common BC policy error is to push down before the leg is aligned with the hole. This often results in a shift of the object in the grasp, which causes the policy to diverge due to the out-of-distribution grasp pose. We observe the residual policy regularly corrects these errors by performing small sideways translations while canceling out premature downward motions, typically only allowing the leg to be inserted once its properly above the hole. We also find the residual policy is better at performing initial grasps that allow accurate downstream alignment between the grasped object and the receptacle. The BC policies, on the other hand, more often grasp the object at angles that make insertion more difficult.

In addition to improved success rates, we also observed quite different training dynamics across each method. First, direct MLP-C fine-tuning proved unstable and required KL-regularization to avoid collapse. Second, the trained DP produced actions with low variance, even with $\eta = 1$ in the DDIM sampler [36], inhibiting Q-learning and constraining the potential for policy improvement. Finally, residual policy training was quite stable, likely because it is constrained to operate on a local scale, which prevents large deviations that can make RL unstable [37, 38].

4.2 Distillation Performance from Synthetic RL Data

Next, we study how synthetic RL data quantity and quality impact the performance of distilled vision-based policies. First, we find that distilling trajectories from the RL agent performs better than training directly on the 50 demonstrations. The RL-distilled vision-based policy reached 73%, outperforming the 50% achieved by training the vision policy directly on human demos. However, we also observe a performance gap between the RL-trained teacher (95%) and distilled student policy (73%). We consider whether this gap may be caused by training the student to operate on images. Upon examination, we find that distilling the same number of teacher rollouts into an image-based student and a state-based student results in comparable performance.

This leads us to conclude that the change in modality is not the primary source of the performance gap. Therefore, we also examine the impact of the distillation dataset size. Here, we scale up the number of state-based rollouts from the trained RL policy and distill these to a state-based student. In Fig. 5, we see that performance increases with more data but still does not reach the performance achieved by the teacher policy, with a gap of 18 percentage points between the best student policy (77% success rate) and the teacher policy (95% success rate) at 10k trajectories. Nevertheless, the improved performance obtained by training with more data highlights the advantage of using simulation for obtaining large-scale synthetic datasets.

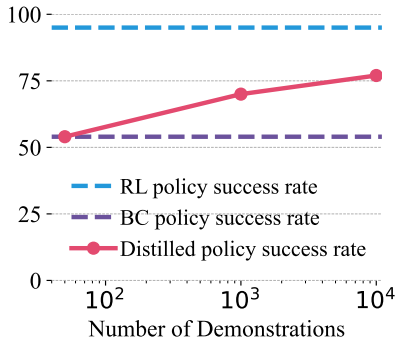

Figure 5: BC distillation scaling with dataset size.

4.3 Real-World Performance

Finally, we evaluate the real-world performance of a sim-to-real policy trained on a mixture of a few (10/40) real-world demonstrations and simulation data generated by the trained residual RL policy. We compare the co-trained policy to a baseline model trained solely on real-world demonstrations. We compare the success rates achieved by each policy on two sets of 10 trials for the `one_leg` task. In the first set, we randomize part poses, while in the second set, we randomize obstacle poses.

The results, presented in Tab. 2, show that incorporating simulation data improves real-world performance (e.g., increasing task completion rate from 20-30% to 50-60%). Qualitatively, the sim-to-real policy exhibits smoother behavior and makes fewer erratic movements that might exceed the robot's physical limits. Fig. 6 shows examples of successful and unsuccessful task attempts, and more examples of real-world task execution can be found in the supplementary video.

Training data	Corner		Grasp		Insert		Screw		Complete	
	Part	Obs	Part	Obs	Part	Obs	Part	Obs	Part	Obs
10 Real	5/10	5/10	5/10	7/10	2/10	3/10	0/10	2/10	0/10	2/10
10 Real + 350 Sim	9/10	9/10	7/10	8/10	0/10	3/10	0/10	3/10	0/10	3/10
40 Real	10/10	8/10	9/10	8/10	6/10	3/10	2/10	3/10	2/10	3/10
40 Real + 350 Sim	10/10	10/10	9/10	10/10	6/10	7/10	5/10	6/10	5/10	6/10

Table 2: We compare the impact of combining real-world demonstrations with simulation trajectories obtained by rolling our RL-trained residual policies. We find that co-training with both real and synthetic data leads to improved motion quality and success rate on the `one_leg` task.

5 Related Works

Training diffusion models with reinforcement learning The approach in [16, 22] studied how to cast diffusion de-noising as a Markov Decision Process, enabling preference-aligned image generation with policy gradient RL. However, this method is unstable and hard to tune. Other ways to combine diffusion architectures with RL include Q-function-based importance sampling [24], advantage weighted regression [25], or changing the objective into a supervised learning problem with return conditioning [26, 27, 39]. Some have also explored augmenting the de-noising training objective with a Q-function maximization objective [28] and iteratively changing the dataset with Q-functions [40]. Recent work developed techniques for training diffusion policies from scratch [23], leveraging combinations of unsupervised clustering and Q-learning to encourage multi-modal behavior discovery. Our method avoids such complexity involved with directly optimizing diffusion

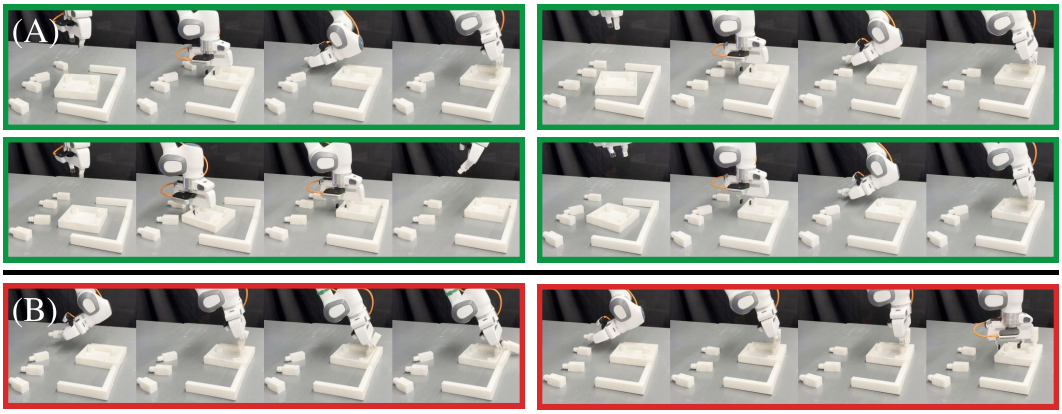

Figure 6: **(A)** Examples of successful real world assembly from RGB. Co-training with simulation data reduces jerkiness and improves insertion robustness, due to data from simulation containing higher diversity of initial part poses and insertion locations (see Table 2). **(B)** Representative failures, primarily due to overfitting to a particular corner location and difficulty adjusting the insertion angle/position when grasps lead to unfamiliar in-hand part poses.

models by instead using standard PPO to train simple residual policies that locally correct for errors made by the base policy.

Residual learning in robotics Learning corrective residual components in conjunction with learned or non-learned "base" models has seen widespread success in robotics. Common frameworks include learning residual policies that correct for errors made by a nominal behavior policy [17, 18, 41, 19, 42] and combining learned components to correct for inaccuracies in analytical models for physical dynamics [43, 44, 45] or sensor observations [46]. Residual policies have been used in insertion applications [47] and recent work has applied residual policy learning to the same FurnitureBench task suite we study in this paper [8]. Their approach uses the residual component to model online human-provided corrections via supervised learning, whereas we train our residual policy from scratch with RL using task rewards in simulation.

6 Limitations and Conclusion

Limitations The local nature of our residual policies is not well suited for learning the macro-level corrective behaviors that are required to recover from large-scale deviations like dropped parts and out-of-distribution in-hand object poses. Furthermore, despite showcasing the advantage of incorporating simulation data, we have yet to achieve high absolute success rates in the real world. Finally, our proposed pipeline struggles in regimes with very high initial scene randomness, as both the base policies and actions produced via RL exploration struggle to deal with out-of-support initial part poses. Exciting future investigations may include better sim-to-real transfer techniques, exploration mechanisms for discovering how to correct large-scale execution errors and incorporating inductive biases that help with generalization to much broader initial state distributions.

Conclusion This work presents an approach for fine-tuning BC-trained policies for precise manipulation tasks using sparse rewards. We use RL to train residual policies that produce locally corrective actions on top of base models with architecture components that complicate RL, such as diffusion models and chunked action predictions. Our results show the proposed method outperforms alternative techniques for fine-tuning imitation-learned assembly policies with RL. We furthermore show that through teacher-student distillation and sim-to-real co-training techniques, the precise behaviors acquired by our residual learner can be distilled into a real-world assembly policy that operates directly from RGB images.

References

[1] S. Schaal. Learning from Demonstration. In *Advances in Neural Information Processing Systems*, volume 9. MIT Press, 1996. URL https://proceedings.neurips.cc/paper_files/paper/1996/hash/68d13cf26c4b4f4f932e3eff990093ba-Abstract.html.

[2] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3 (6):233–242, 1999.

[3] N. Ratliff, J. A. Bagnell, and S. S. Srinivasa. Imitation learning for locomotion and manipulation. In *2007 7th IEEE-RAS international conference on humanoid robots*, pages 392–397. IEEE, 2007.

[4] P. Agrawal. *Computational sensorimotor learning*. University of California, Berkeley, 2018.

[5] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR, 2022.

[6] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi:10.15607/RSS.2023.XIX.016.

[7] A. Kumar, Z. Fu, D. Pathak, and J. Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

[8] Y. Jiang, C. Wang, R. Zhang, J. Wu, and L. Fei-Fei. Transic: Sim-to-real policy transfer by learning from online correction. *arXiv preprint arXiv:2405.10315*, 2024.

[9] T. Chen, M. Tippur, S. Wu, V. Kumar, E. Adelson, and P. Agrawal. Visual dexterity: In-hand reorientation of novel and complex object shapes. *Science Robotics*, 8(84):eadc9244, 2023.

[10] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

[11] S. James and A. J. Davison. Q-attention: Enabling efficient learning for vision-based robotic manipulation. *IEEE Robotics and Automation Letters*, 7(2):1612–1619, 2022.

[12] Y. Lu, K. Hausman, Y. Chebotar, M. Yan, E. Jang, A. Herzog, T. Xiao, A. Irpan, M. Khansari, D. Kalashnikov, et al. Aw-opt: Learning robotic skills with imitation andreinforcement at scale. In *Conference on Robot Learning*, pages 1078–1088. PMLR, 2022.

[13] L. Ankile, A. Simeonov, I. Shenfeld, and P. Agrawal. Juicer: Data-efficient imitation learning for robotic assembly. *arXiv preprint arXiv:2404.03729*, 2024.

[14] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion, June 2023. URL http://arxiv.org/abs/2303.04137. arXiv:2303.04137 [cs].

[15] M. Heo, Y. Lee, D. L. Kaist, and J. J. Lim. FurnitureBench: Reproducible Real-World Benchmark for Long-Horizon Complex Manipulation. *RSS 2023*, 2023. URL https://clvrai.com/furniture-bench. arXiv: 2305.12821v1.

[16] K. Black, M. Janner, Y. Du, I. Kostrikov, and S. Levine. Training diffusion models with reinforcement learning. *arXiv preprint arXiv:2305.13301*, 2023.

[17] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

[18] T. Davchev, K. S. Luck, M. Burke, F. Meier, S. Schaal, and S. Ramamoorthy. Residual learning from demonstration: Adapting dmps for contact-rich manipulation. *IEEE Robotics and Automation Letters*, 7(2):4488–4495, 2022.

[19] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine. Residual reinforcement learning for robot control. In *2019 international conference on robotics and automation (ICRA)*, pages 6023–6029. IEEE, 2019.

[20] Y. Narang, K. Storey, I. Akinola, M. Macklin, P. Reist, L. Wawrzyniak, Y. Guo, A. Moravanszky, G. State, M. Lu, et al. Factory: Fast contact for robotic assembly. *arXiv preprint arXiv:2205.03532*, 2022.

[21] Y. Fan and K. Lee. Optimizing ddpm sampling with shortcut fine-tuning. *arXiv preprint arXiv:2301.13362*, 2023.

[22] Y. Fan, O. Watkins, Y. Du, H. Liu, M. Ryu, C. Boutilier, P. Abbeel, M. Ghavamzadeh, K. Lee, and K. Lee. Dpok: Reinforcement learning for fine-tuning text-to-image diffusion models, 2023.

[23] Z. Li, R. Krohn, T. Chen, A. Ajay, P. Agrawal, and G. Chalvatzaki. Learning multimodal behaviors from scratch with diffusion policy gradient, 2024.

[24] P. Hansen-Estruch, I. Kostrikov, M. Janner, J. G. Kuba, and S. Levine. IDQL: Implicit Q-Learning as an Actor-Critic Method with Diffusion Policies, May 2023. URL http://arxiv.org/abs/2304.10573. arXiv:2304.10573 [cs].

[25] W. Goo and S. Niekum. Know your boundaries: The necessity of explicit behavioral cloning in offline rl. *arXiv preprint arXiv:2206.00695*, 2022.

[26] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[27] A. Ajay, Y. Du, A. Gupta, J. Tenenbaum, T. Jaakkola, and P. Agrawal. Is Conditional Generative Modeling all you need for Decision-Making?, July 2023. URL http://arxiv.org/abs/2211.15657. arXiv:2211.15657 [cs].

[28] Z. Wang, J. J. Hunt, and M. Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*, 2022.

[29] M. Wołczyk, B. Cupiał, M. Ostaszewski, M. Bortkiewicz, M. Zajkac, R. Pascanu, Ł. Kuciński, and P. Miłoś. Fine-tuning reinforcement learning models is secretly a forgetting mitigation problem. *arXiv preprint arXiv:2402.02868*, 2024.

[30] A. Nair, A. Gupta, M. Dalal, and S. Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[31] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms, Aug. 2017. URL http://arxiv.org/abs/1707.06347. arXiv:1707.06347 [cs].

[33] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.

[34] M. Torne, A. Simeonov, Z. Li, A. Chan, T. Chen, A. Gupta, and P. Agrawal. Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation. *arXiv preprint arXiv:2403.03949*, 2024.

[35] S. Han, I. Shenfeld, A. Srivastava, Y. Kim, and P. Agrawal. Value augmented sampling for language model alignment and personalization, 2024.

[36] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[37] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[38] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[39] M. Janner, Y. Du, J. B. Tenenbaum, and S. Levine. Planning with Diffusion for Flexible Behavior Synthesis, Dec. 2022. URL http://arxiv.org/abs/2205.09991. arXiv:2205.09991 [cs].

[40] L. Yang, Z. Huang, F. Lei, Y. Zhong, Y. Yang, C. Fang, S. Wen, B. Zhou, and Z. Lin. Policy representation via diffusion probability model for reinforcement learning, 2023.

[41] J. Carvalho, D. Koert, M. Daniv, and J. Peters. Residual robot learning for object-centric probabilistic movement primitives. *arXiv preprint arXiv:2203.03918*, 2022.

[42] M. Alakuijala, G. Dulac-Arnold, J. Mairal, J. Ponce, and C. Schmid. Residual reinforcement learning from demonstrations. *arXiv preprint arXiv:2106.08050*, 2021.

[43] A. Ajay, J. Wu, N. Fazeli, M. Bauza, L. P. Kaelbling, J. B. Tenenbaum, and A. Rodriguez. Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3066–3073. IEEE, 2018.

[44] A. Kloss, S. Schaal, and J. Bohg. Combining learned and analytical models for predicting action effects from sensory data. *The International Journal of Robotics Research*, 41(8):778–797, 2022.

[45] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*, 36(4):1307–1319, 2020.

[46] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

[47] G. Schoettler, A. Nair, J. Luo, S. Bahl, J. A. Ojea, E. Solowjow, and S. Levine. Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5548–5555. IEEE, 2020.

[48] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5745–5753, 2019.

[49] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[50] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[51] S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta. R3M: A Universal Visual Representation for Robot Manipulation, Nov. 2022. URL http://arxiv.org/abs/2203.12601. arXiv:2203.12601 [cs].

[52] J. Levinson, C. Esteves, K. Chen, N. Snavely, A. Kanazawa, A. Rostamizadeh, and A. Makadia. An analysis of svd for deep rotation estimation. *Advances in Neural Information Processing Systems*, 33:22554–22565, 2020.

[53] A. R. Geist, J. Frey, M. Zobro, A. Levina, and G. Martius. Learning with 3d rotations, a hitchhiker's guide to so(3), 2024.

[54] M. Reuss, M. Li, X. Jia, and R. Lioutikov. Goal-Conditioned Imitation Learning using Score-based Diffusion Policies, June 2023. URL http://arxiv.org/abs/2304.02532. arXiv:2304.02532 [cs].

[55] B. Tang, M. A. Lin, I. Akinola, A. Handa, G. S. Sukhatme, F. Ramos, D. Fox, and Y. Narang. Industreal: Transferring contact-rich assembly tasks from simulation to reality. In *Robotics: Science and Systems*, 2023.

[56] X. Zhang, S. Jin, C. Wang, X. Zhu, and M. Tomizuka. Learning insertion primitives with discrete-continuous hybrid action space for robotic assembly tasks. In *2022 International conference on robotics and automation (ICRA)*, pages 9881–9887. IEEE, 2022.

[57] O. Spector and D. Di Castro. Insertionnet-a scalable solution for insertion. *IEEE Robotics and Automation Letters*, 6(3):5509–5516, 2021.

[58] Y. Tian, K. D. Willis, B. A. Omari, J. Luo, P. Ma, Y. Li, F. Javid, E. Gu, J. Jacob, S. Sueda, et al. Asap: Automated sequence planning for complex robotic assembly with physical feasibility. *arXiv preprint arXiv:2309.16909*, 2023.

[59] H. Hu, S. Mirchandani, and D. Sadigh. Imitation Bootstrapped Reinforcement Learning, Nov. 2023. URL http://arxiv.org/abs/2311.02198. arXiv:2311.02198 [cs].

[60] J. Luo, Z. Hu, C. Xu, Y. L. Tan, J. Berg, A. Sharma, S. Schaal, C. Finn, A. Gupta, and S. Levine. SERL: A Software Suite for Sample-Efficient Robotic Reinforcement Learning, Jan. 2024. URL http://arxiv.org/abs/2401.16013. arXiv:2401.16013 [cs].

[61] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

7 Appendix

7.1 Code and Data

Our code is attached to the supplementary materials. All code and data will be open-sourced upon publication.

7.2 Implementation Details

7.2.1 Training Hyperparameters

State-based behavior cloning We provide a detailed set of hyperparameters used for training. General hyperparameters for all models can be found in Tab. 3, while specific hyperparameters for the diffusion models are in Tab. 4, and those for the MLP baseline are in Tab. 5.

Table 3: Training hyperparameters shared for all state-based BC models

Parameter	Value
Control mode	Absolute end-effector pose
Action space dimension	10
Proprioceptive state dimension	16
Orientation Representation	6D [48]
Max LR	10^{-4}
LR Scheduler	Cosine
Warmup steps	500
Weight Decay	10^{-6}
Batch Size	256
Max gradient steps	400k

Table 4: State-based diffusion pre-training hyperparameters

Parameter	Value
U-Net Down dims	$[256, 512, 1024]$
Diffusion step embed dim	256
Kernel size	5
N groups	8
Parameter count	66M
Observation Horizon T_o	1
Prediction Horizon T_p	32
Action Horizon T_a	8
DDPM Training Steps	100
DDIM Inference Steps	4

Table 5: State-based MLP pre-training hyperparameters

Parameter	Value
Residual Blocks	5
Residual Block Width	1024
Layers per block	2
Parameter count	11M
Observation Horizon T_o	1
Prediction Horizon T_p (S / C)	1 / 8
Action Horizon T_a (S / C)	1 / 8

State-based reinforcement learning Below, we list the hyperparameters used for online reinforcement learning fine-tuning. The parameters that all state-based RL methods methods shared are in Tab. 6. Method-specific hyperparameters for training the different methods are in the tables below, direct fine-tuning of the MLP in Tab. 7, online IDQL in Tab. 8, and the residual policy in Tab. 9. The different methods were tuned independently, but the same hyperparameters were used for all tasks within each method.

Table 6: Hyperparameters shared for all online fine-tuning approaches

Parameter	Value
Control mode	Absolute end-effector pose
Action space dimension	10
Proprioceptive state dimension	16
Orientation Representation	6D [48]
Num parallel environments	1024
Max environment steps	500M
Critic hidden size	256
Critic hidden layers	2
Critic activation	ReLU
Critic last layer activation	Linear
Critic last layer bias initialization	0.25
Discount factor	0.999
GAE [49] lambda	0.95
Clip ϵ	0.2
Max gradient norm	1.0
Target KL	0.1
Num mini-batches	1
Episode length, `one_leg`	700
Episode length, `lamp/round_table`	1000
Normalize advantage	true

Table 7: Hyperparameters for direct fine-tuning of MLP

Parameter	Value
Update epochs	1
Learning rate actor	10^{-4}
Learning rate critic	10^{-4}
Value function loss coefficient	1.0
KL regularization coefficient	0.5
Actor Gaussian initial log st.dev.	-4.0

Table 8: Hyperparameters for training value-augmented diffusion sampling (IDQL)

Parameter	Value
Update epochs	10
Learning rate Q-function	10^{-4}
Learning rate scheduler	Cosine
Num action samples	20
Actor added Gaussian noise, log st.dev.	-4

Table 9: Hyperparameters for residual PPO training

Parameter	Value
Residual action scaling factor	0.1
Update epochs	50
Learning rate actor	$3 \cdot 10^{-4}$
Learning rate critic	$5 \cdot 10^{-3}$
Learning rate scheduler	Cosine
Value function loss coefficient	1.0
Actor Gaussian initial log st.dev.	-1.5

Image-based real-world distillation For the real-world experiments, we use a separate set of hyperparameters, presented in Tab. 10. The main difference is that we found in experimentation that the transformer backbone in [14] worked better than the UNet for real-world experiments. These models are also operating from RGB observations instead of privileged states, and we provide parameters for the image augmentations applied to the front camera in Tab. 11 and the wrist camera in Tab. 12.

Table 10: Training hyperparameters for real-world distilled policies

Parameter	Value
Control mode	Absolute end-effector pose
Action space dimension	10
Proprioceptive state dimension	16
Orientation Representation	6D [48]
Max policy LR	10^{-4}
Max encoder LR	10^{-5}
LR Scheduler (both)	Cosine
Policy scheduler warmup steps	1000
Policy scheduler warmup steps	5000
Weight decay	10^{-3}
Batch size	256
Max gradient steps	500k
Image size input	$2 \times 320 \times 240 \times 3$
Image size encoder	$2 \times 224 \times 224 \times 3$
Vision Encoder Model	ResNet18 [50]
Encoder Weights	R3M [51]
Encoder Parameters	2×11 million
Encoder Projection Dim	128
Diffusion backbone architecture	Transformer (similar to [14])
Transformer num layers	8
Transformer num heads	4
Transformer embedding dim	256
Transformer embedding dropout	0.0
Transformer attention dropout	0.3
Transformer causal attention	true

Table 11: Parameters for front camera image augmentation

Parameter	Value
Color jitter (all parameters)	0.3
Gaussian blur, kernel size	5
Gaussian blur, sigma	$(0.01, 1.2)$
Random crop area	280×240
Random crop size	224×224
Random erasing, fill value	random
Random erasing, probability	0.2
Random erasing, scale	$(0.02, 0.33)$
Random erasing, ratio	$(0.3, 3.3)$

Table 12: Parameters for front camera image augmentation

Parameter	Value
Color jitter (all parameters)	0.3
Gaussian blur, kernel size	5
Gaussian blur, sigma	$(0.01, 1.2)$
Random crop	Not used
Image resize	$320 \times 240 \rightarrow 224 \times 224$

7.2.2 Action and State-Space Representations

Action space The policies predict 10-dimensional actions consisting of absolute poses in the robot base frame as the actions and a gripper action. In particular, the first 3 dimensions predict the desired end-effector position in the workspace, the next 6 predict the desired orientation using a 6-dimensional representation described below. The final dimension is a gripper action, 1 to command closing gripper and -1 for opening.

Proprioceptive state space The policy receives a 16-dimensional vector containing the current end-effector state and gripper width. In particular, the first 3 dimensions is the current position in the workspace, the next 6 the current orientation in the base frame (the same 6D representation), the next 3 the current positional velocity, the next 3 the current roll, pitch, and yaw angular velocity, and finally the current gripper width.

Rotation representation We use a 6D representation to represent all orientations and rotations for the predicted action, and proprioceptive end-effector pose orientation [48, 52]. The poses of the parts in state-based environments are represented with unit quaternions. While this representation contains redundant dimensions, it is continuous, meaning that small changes in orientation lead to small changes in the representation values, which can make learning easier[48, 52, 53]. This is not generally the case for Euler angles and quaternions. The 6D representation is constructed by taking two arbitrary 3D vectors and performing Gram-Schmidt orthogonalization to obtain a third orthogonal vector to the first two. The resulting three orthogonal vectors form a rotation matrix that represents the orientation. The end-effector rotation angular velocity is still encoded as roll, pitch, and yaw values.

Action and State-Space Normalization All dimensions of the action, proprioceptive state, and parts pose (for state-based environments), were independently scaled to the range [-1, 1]. That is, we did not handle orientation representations (quaternions/6D [48]) in any particular way. The normalization limits were calculated over the dataset at the start of behavior cloning training. They were stored in the actor with the weights and reused as the normalization limits when training with reinforcement learning. The normalization used here follows the same approach as in previous

works such as [54, 14]. This normalization method is widely accepted for diffusion models. In [54], the input was standardized to have a mean of 0 and a standard deviation of 1, instead of using min-max scaling to the range of [0, 1]. This approach was not tested in our experiments.

7.2.3 Image Augmentation

During training, we apply image augmentation and random cropping to both camera views. Specifically, only the front camera view undergoes random cropping. We also apply color jitter with a hue, contrast, brightness, and saturation set to 0.3. Additionally, we apply Gaussian blur with a kernel size of 5 and sigma between 0.1 and 5 to both camera views.

At inference time, we statically center-crop the front camera image from 320x240 to 224x224 and resize the wrist camera view to the same dimensions. For both the random and center crops, we resized the image to 280x240 to ensure that essential parts of the scene are not cropped out due to excessive movement.

The specific values mentioned above were chosen based on visual assessment to strike a balance between creating adversarial scenarios and keeping essential features discernible. We have included examples of these augmentations below.

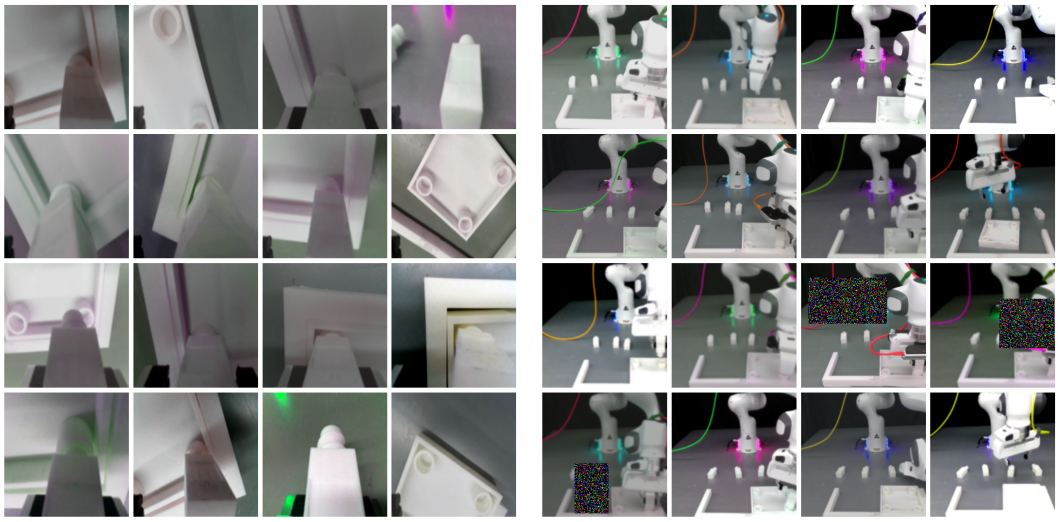

Figure 7: **Left:** Examples of augmentations of the wrist camera view, consisting of color jitter and Gaussian blur. **Right:** Examples of augmentations for the front view also consist of color jitter and Gaussian blur augmentations, as well as random cropping.

7.3 Sim2Real Transfer

Visualization of overlap in action space in real and sim For data from the simulation to be useful for *increasing* the support of the policy for real-world deployment, we posit that it needs to *cover* the real-world data. We visualize the distributions of actions in the training data in Fig. 8. Since actions are absolute poses in the robot base frame, we can take the x, y, z coordinates for all actions from simulation and real-world demonstration data and plot them. Each of the 3 plots is a different cross-section of the space, i.e., a view from top-down, side, and front. In general, we see that the simulation action distribution is more spread out and mostly covers real-world actions.

Visual Domain randomization In addition to randomizing part poses and the position of the obstacle, we randomize parts of the rendering which is not easily randomized by simple image augmentations, like light placement (changing shadows), camera pose, and individual part colors. See Fig. 9 for examples of front-view images obtained from our domain randomization and re-rendering procedure.

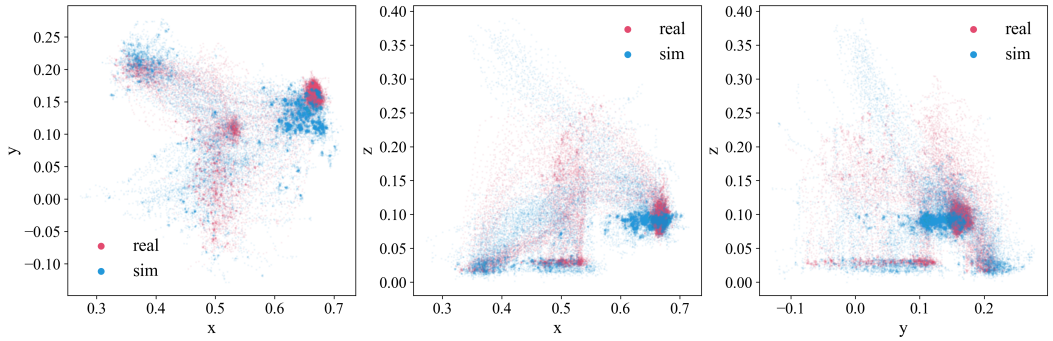

Figure 8: Plots of the x, y, z action coordinates in the demo datasets for the `one_leg` task in the real world and the simulator. That is, each dot represents one action from one of the 40/50 trajectories. Red is from real-world demos, and blue is from the simulator. **Left:** Top-down view, showing the x, y positions in the workspace visited. In the top right, the insertion point is shown, where we see that the simulator has a wider distribution but could have covered better in the positive y-direction. **Middle:** Side-view of the actions taken in the x, z plane. The insertion point is to the right in the plot; again, we see more spread in the simulation data. **Right:** Front view of the y, z actions.

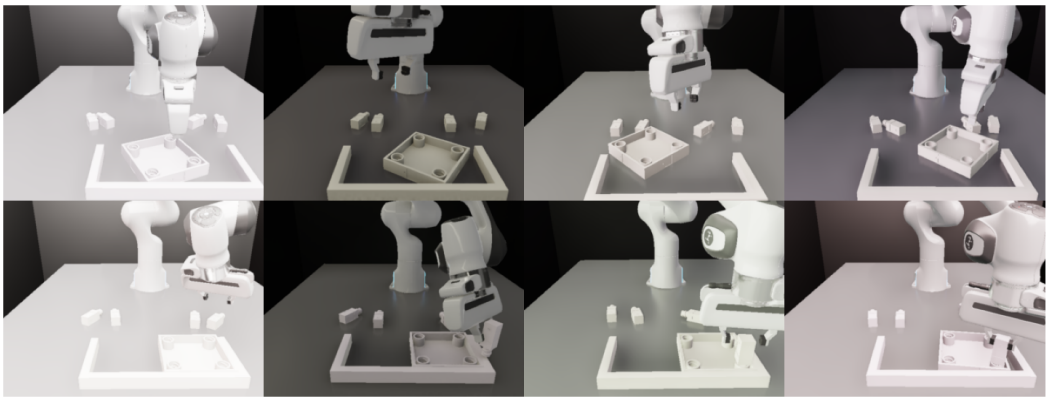

Figure 9: Examples of the randomization applied when rendering out the simulation trajectories used for co-training for the real-world policies.

7.4 Tasks and Environment

Tasks details and reward signal We detail a handful of differentiating properties for each of the three tasks we use in Tab. 13. `one_leg` involves assembling 2 parts, the tabletop and one of the 4 table legs. The assembly is characterized as successful if the relative poses between the parts are close to a predefined assembled relative pose. When this pose is achieved, the environment returns a reward of 1. That is, for the `one_leg` task, the policy received a reward of 1 only at the very end of the episode. For `round_table` and `lamp`, which consists of assembling 3 parts together, the policy receives a reward signal of 1 for each pair of assembled parts. E.g. for the `lamp` task, when the bulb is fully screwed into the base, the first reward of 1 is received, and the second is received when the shade is correctly placed.

Table 13: Task Attribute Overview

	One Leg	Round Table	Lamp
Mean episode length	∼500	∼700	∼600
# Parts to assemble	2	3	3
Num rewards	1	2	2
Dynamic bbject	✗	✗	✓
# Precise insertions	1	2	1
# Screwing sequences	1	2	1
Precise grasping	✗	✓	✗
Insertion occlusion	✗	✓	✗

Details on randomization scheme The "low" and "medium" randomness settings we used for data collection and evaluation reflect how much the initial part poses may vary when the environment is reset. We tuned these conditions to mimic the levels of randomness introduced in the original FurnitureBench suite [15]. However, we found that their method of directly sampling random poses often leads to initial part configurations that collide with each other, requiring expensive continued sampling to eventually find an initial layout where all parts do not collide with each other.

Our modified randomization scheme instead initializes parts to a single pre-specified set of feasible configurations and then applies a randomly sampled force and torque to each part (where the force/torque magnitudes are tuned for each part and scaled based on the desired level of randomness). This scheme allows the physics simulation to ensure parts stay out of collision while still providing a controlled amount of variation in the initial scene randomness.

The second way we modified the randomization scheme was to randomize the position of the U-shaped obstacle fixture and the parts (the obstacle fixture was always kept in a fixed position in [15]). Our reasoning was that, for visual sim-to-real without known object poses, we could only imperfectly and approximately align the obstacle location in the simulated and real environment. Rather than attempting to make this alignment perfect, we instead trained policies to cover some range of possible obstacle locations, hoping that the real-world obstacle position would fall within the distribution the policies have seen in simulation. Fig. 10 shows examples of our different randomness levels for each task in simulation.

Additional adjustments to FurnitureBench simulation environments In addition to our modified force-based method of controlling the initial randomness, we introduced multiple other modifications to the original FurnitureBench environments proposed in [15] to enable the environment to run fast enough to be feasible for online RL training. With these changes, we were able to run at a total ∼4000 environment steps per second across 1024 parallel environments. The main changes are listed below:

1. Vectorized reward computation, done check, robot, part, and obstacle resets, and differential inverse kinematics controller.
2. Removed April tags from 3D models to ensure vision policies would not rely on tags to complete the tasks. We tried to align with the original levels of randomness, but only to an approximation.
3. Deactivate camera rendering when running the environment in state-only mode.
4. Correct an issue where the physics was not stepped a sufficient amount of time for sim time to run at 10Hz, and subsequently optimize calls to fetch simulation results, stepping of graphics, and refreshing buffers.
5. Artificially constrained bulb from rolling on the table until robot gripper is nearby as the rolling in the simulator was exaggerated compared to the real-world parts.

Figure 10: Examples of initial scene layouts for `one_leg`, `lamp`, and `round_table` with different levels of initial part pose and obstacle fixture randomness

7.5 Real-World Results

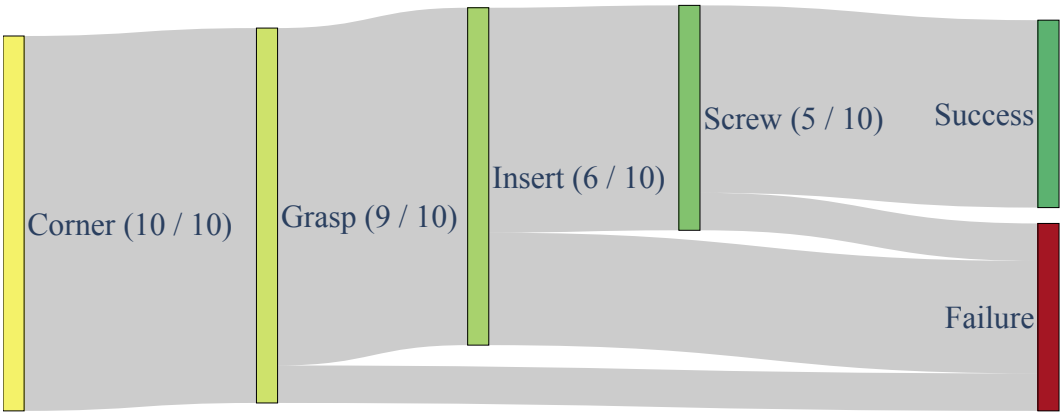

Figure 11: Sankey diagram for the success rate and failure points for the real-world rollouts with 40 real and 350 simulation demos.

The diagram in Fig. 11 shows how successful and failed completion of individual sub-skills along the `one_leg` task amount to our overall final success rates reported in Tab. 2 (bottom row, corresponding to "40 real + 350 sim" with random initial part poses and a fixed obstacle pose).

8 Expanded Related Work

Learning robotic assembly skills Robotic assembly has been used by many as a problem setting for various behavior learning techniques [55, 56, 18, 57, 58]. Enabling assembly that involves multi-skill sequencing (e.g., fixturing → grasping → insertion → screwing) directly from RGB images has remained challenging, especially *without* explicitly defining sub-skill-specific boundaries and supervision. Concurrent work [8] explores a similar framework to ours on FurnitureBench tasks [15], but instead supervises learned policies on a per-skill basis and incorporates 3D point clouds. IndustReal [55] also leverages RL in simulation to train high-precision skills for tight-tolerance part insertion in the real world. However, they train their RL policies from scratch using carefully-designed shaped rewards and curricula, whereas we bootstrap RL from BC pre-training, which enables RL to operate with simple sparse rewards for achieving the desired assembly.

Complementary combinations of behavior cloning and reinforcement learning Various combinations of learning from demonstrations/behavior cloning and reinforcement learning have begun maturing into standard tools in the learning-based control development paradigm [12, 10]. For instance, demonstrations are often used to support RL in overcoming exploration difficulty and improving sample efficiency [59, 34, 60]. RL can also act as a robustification operator to improve upon base BC behaviors [12, 34], paralleling the RL fine-tuning paradigm that has powered much of the recent advancement in other areas like NLP [61] and vision [16]. Additionally, many successful robotics deployments [9, 33, 7] have been powered by the "teacher-student distillation" paradigm, wherein perception-based "student" policies are trained to clone behaviors produced by a state-based "teacher" policy, which is typically trained via RL in simulation. We demonstrate that our residual RL approach for fine-tuning modern diffusion policy architectures can allow each of these complementary ways to combine BC and RL to come together and enable precise manipulation directly from RGB images.