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

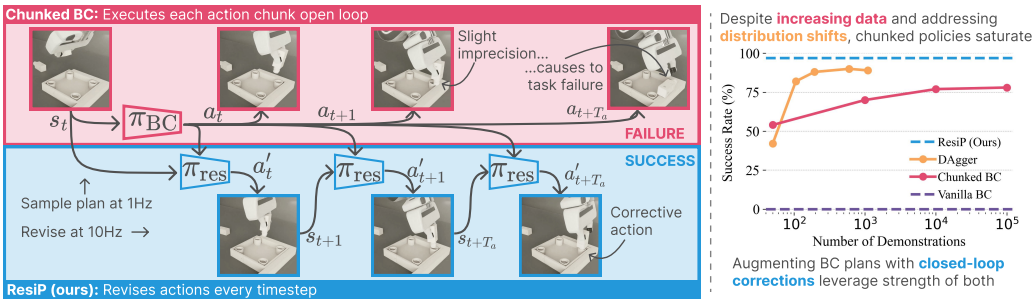


Figure 1: **Left:** (Top) Tasks like assembly require long-horizon coordination and high precision, at which current BC methods fail due to their chunk-level open-loop nature. (Bottom) Combining a BC trajectory planner with an RL-trained closed-loop residual policy results in surprisingly robust and reactive behaviors. **Right:** Chunking improves performance over standard policy architectures. Still, performance saturates despite increasing data and addressing distribution shifts with DAgger [1]. Combining chunking with closed-loop corrections (`ResiP`) combines the strength of each.

Abstract: Recent advances in behavior cloning (BC), like action-chunking and diffusion, have led to impressive progress. Still, imitation alone remains insufficient for tasks requiring reliable and precise movements, such as aligning and inserting objects. Our central insight is that chunked BC policies function as trajectory planners, enabling long-horizon tasks. Still, since they execute action chunks open loop, they lack the fine-grained reactivity necessary for reliable execution. Further, we find that the performance of BC policies saturates despite increasing data. We present a simple yet effective method, `ResiP` (*Resi*dual for *P*recise Manipulation), that sidesteps these challenges by augmenting a frozen, chunked BC model with a fully closed-loop residual policy trained with reinforcement learning (RL). The residual policy is trained via on-policy RL, addressing distribution shifts and reactivity without altering the BC trajectory planner. Evaluation on high-precision manipulation tasks demonstrates strong performance of `ResiP` over BC methods and direct RL fine-tuning. Videos, code, data, and supplementary materials are available at `https://residual-assembly.github.io`.

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

1 Introduction

Robotic manipulation tasks requiring both long-horizon planning and high-precision control, such as assembly, remain a significant challenge in robotics [2, 3, 4, 5, 6]. Consider the furniture assembly task in Fig. 1 (Left), where a robot sequentially grasps, orients, inserts, and screws in four legs. This task involves executing a specific sequence of skills over hundreds of timesteps, each dependent on the previous stage. Small imprecisions at any point can lead to failure, highlighting the need for reliable execution of diverse skills in sequence.

8th Conference on Robot Learning (CoRL 2024), Munich, Germany.

	Methods	one_leg		round_table		lamp		mug-rack	peg-in-hole
		Low	Med	Low	Med	Low	Med	Low	Low
IL	MLP-S	0	0	0	0	0	0	0	0
IL	MLP-C	45	10	5	2	**8**	1	21	2
IL	DP	**54**	**29**	**12**	**4**	7	**2**	**29**	**3**
RL	MLP-C + PPO	70	28	38	6	32	2	23	4
RL	DP + IDQL	52	13	18	3	11	1	31	3
RL	ResiP (ours)	**98**	**76**	**94**	**77**	**97**	**70**	**88**	**99**

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 DP performs the best among the evaluated fine-tuning techniques.

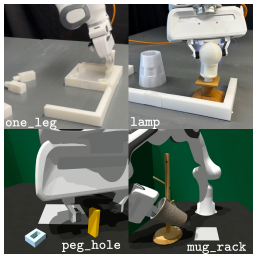


Figure 2: Examples of tasks used for method evaluations.

Behavior cloning (BC) is favored for teaching robots manipulation skills [7, 8, 9, 10, 11, 12, 13, 14, 6, 15]. Recent innovations like diffusion models [16, 17, 18, 19, 20] and action chunking [21, 6, 17, 19] enable learning complex behaviors from demonstrations [22, 15]. However, BC shows limitations in high-precision tasks. In Fig. 1 (Left), a diffusion-based BC model achieves ∼50% success with few demonstrations but plateaus at ∼80% even with 100,000 (Fig. 1 (Right)). Other studies report similar saturation [23, 24, 25].

We hypothesize this saturation occurs due to two issues: (1) BC suffers from compounding errors due to distributional shifts [23]. While Dataset Aggregation (DAgger) [1] can mitigate this, it requires an on-demand, often unavailable expert. (2) Modern BC policies, especially those using action chunking, act more like open-loop "planners" than reactive controllers. In tasks like furniture assembly, "bottleneck" states [22] (e.g., insertions) require precise actions at specific times. If these critical moments fall within an action chunk, the BC planner cannot make real-time adjustments, leading to task failure in high-precision tasks.

Reinforcement Learning (RL) can learn corrective behaviors through interaction and exploration [26, 27], effectively training robust reactive controllers for various tasks [28, 29, 30, 31, 32]. However, applying RL to complex, long-horizon, precise tasks like robotic assembly requires extensive reward engineering [33] and vast amounts of data, which is costly, especially for real data.

We aim to leverage both BC and RL: BC learns a "planner" capturing high-level structure from a few demonstrations, while RL augments it with fine-grained corrections. While combining BC and RL has a long history [34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46], recent BC advancements (e.g., diffusion models, action chunking) complicate RL fine-tuning, often leading to instability or requiring significant architectural modifications [47, 15, 48].

We propose ResiP (*Resi*dual for *P*recise Manipulation), shown in Fig. 3, to combine modern chunked BC and on-policy RL controllers. We augment a frozen, chunked BC model with a small, single-step residual policy trained via on-policy RL. This residual policy predicts corrective actions based on each state and BC-predicted action. This approach addresses BC limitations by (1) being on-policy, overcoming distributional shift, (2) enabling real-time adjustments for improved robustness and precision, and (3) using a sparse task-completion reward, avoiding reward engineering.

We evaluate ResiP on high-precision manipulation tasks from FurnitureBench [5] and Factory [49] benchmarks. ResiP outperforms pure BC and traditional RL fine-tuning, achieving ∼97% success from 50 demonstrations, compared to BC's ∼80% with 100k demonstrations. Using teacher-student distillation, visual domain randomization, and co-training, we successfully transfer policies from simulation to a real robot using RGB observations.

2 Methods

Imitation Learning We begin by training a "base" policy using 50 simulated demonstrations, $\mathcal{D}$sim, obtained through teleoperation. Each trajectory τ_i in $\mathcal{D}$sim contains system states s_t and robot actions a_t. We train the base policy π_{base} using Behavior Cloning (BC), maximizing

$\mathbb{E}(a_t, s_t) \sim \mathcal{D}\text{sim}\,[\log \pi(a_t|s_t)]$. We employ a Diffusion Policy [19], known for strong performance on complex tasks with limited data [22]. Our policy predicts action chunks $\mathbf{a_t} = [a_t, ..., a_{t+T_a}]$ of length T_a, executing a subset $[a_t, ..., a_{t+T_{\text{exec}}}]$ with $T_{\text{exec}} \leq T_a$, like recent advancements [19, 6].

Online Reinforcement Learning with Residual Policies Given an initial base policy π_{base}, we aim to improve it with RL. However, directly fine-tuning diffusion models with RL is challenging [50, 51, 47, 52] due to multi-step inference and unavailability of action log-probabilities.

Some methods upweight high-quality model outputs [53, 54, 55, 18, 56], but focus on extracting existing behaviors rather than learning new, corrective ones. Action chunking complicates policy gradient optimization by increasing the effective action space, leading to training instability (see Sec. 3.1). This also affects other BC architectures like ACT [6, 15]. Even successful RL fine-tuning results in trajectory planners lacking closed-loop reactive control (Fig. 6). Additionally, fine-tuning large pre-trained models can lead to capability forgetting if states seen during pre-training are not frequently visited [57].

We propose training a residual Gaussian MLP policy π_{res}, which takes as input both the system state and the action predicted by π_{base}, producing an action correction: $a_t = a_t^{\text{base}} + \alpha \cdot a_t^{\text{res}}$, with $\alpha \leq 1$. The combined policy is denoted $\boldsymbol{\pi}$. This approach allows treating the diffusion model as a black box [58], enables different prediction horizons T_a for the two policies, and removes the need for explicit regularization to stay near the pre-trained policy [59, 60][1].

We train π_{res} using standard PPO [27], augmenting the state space with the base action. For each action in the chunk $\mathbf{a_t}$ output by the base model, 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 $i = 0, ..., T_a - 1$. This approach incorporates an inductive bias for learning *locally* corrective actions [61], motivated by observations that most BC policy failures occur due to slight imprecisions near "bottleneck" transitions in skills like grasping and insertion (e.g., Fig. 1 (Left)).

3 Experiments and Results

3.1 Improving Imitation Learning with Residual Learning

Why Action Chunking and Diffusion Policies? While simple MLPs have shown strong performance in many RL domains [62, 63, 31], single-action MLP policies (MLP-S) fail across all our tasks, as seen in Tab. 1. Introducing action chunking (MLP-C) drastically improves performance. However, Diffusion Policies (DP) generally outperform MLPs, especially in intermediate-difficulty tasks (e.g., 10% to 29% success rate for `one_leg` on medium randomness). DP shows smoother, faster actions, although this occasionally hinders performance (e.g., in `lamp` on low randomness). All methods struggle with the hardest tasks (<5% success rate), indicating a need for stronger BC methods. The `peg-in-hole` task, with a ~0.2 mm clearance insertion, proved particularly challenging (2-3% success rate), supporting the hypothesis that BC methods struggle with high-precision requirements. DP demonstrates greater robustness to noise compared to MLP-C, making it more suitable as a base policy for residual fine-tuning.

Online Policy Fine-Tuning Comparison Next, we investigate the effectiveness of our residual RL approach, `ResiP`, in improving the performance of policies trained with BC. We train residual policies using PPO [27] 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 [53] 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 [64].

[1] This decoupling also means that ACT and other BC methods can serve as the base model without changing the method – an advantage of residual policy learning emphasized by prior robotics work [58, 61].

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 54% to 98%. Even more drastically, for the `peg-in-hole` task, `ResiP` improved the success rate from 3% to 99%. However, we observe performance saturates at lower success rates (e.g., $\sim$ 70%) for tasks with higher part randomization. We hypothesize that the base model's performance limits the residual policy as it is designed to act locally, constrained by the base.

Fig. 1 (Left) illustrates the common failure mode of BC policies and its RL-based solution. BC policies often push down prematurely, misaligning the leg with the hole and causing object shifts that lead to policy divergence due to out-of-distribution grasp poses. The residual policy corrects these errors through small sideways translations and delays downward motions until proper alignment. Additionally, the residual policy demonstrates superior initial grasping, facilitating more accurate downstream alignment, whereas BC policies often grasp at angles that complicate insertion.

In addition to improved success rates, we observed 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 [65], inhibiting Q-learning and constraining the potential for policy improvement. In contrast, 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 [66, 67]. Finally, shown in Fig. 6, `ResiP` exhibits better resilience to random force perturbations (12 percentage point drop) compared to alternative chunked methods (19-26 percentage point drop for BC, DAgger, chunked residual).

3.2 Distillation Performance and Real-World Application

Distillation Performance Distilling trajectories from the RL agent outperforms training directly on demonstrations, with a vision-based policy reaching 73% success on `one_leg` compared to 50% when trained directly on human demos (Fig. 5). However, a performance gap remains between the RL-trained teacher (98%) and distilled student policy (73%). This gap is not primarily due to the change in modality from state to image-based inputs. Scaling up the distillation dataset size improves performance, but a gap persists even with 100k trajectories (78% student vs. 98% teacher success rate, Fig. 1 Right). Despite this, the improvement with more data underscores the advantage of using simulation for large-scale synthetic datasets.

Real-World Application We evaluate a sim-to-real policy trained on a mixture of real-world demonstrations (10/40) and simulation data from the trained residual RL policy, comparing it to a baseline trained only on real-world demos. For the `one_leg` task, we test on two sets of 10 trials each, randomizing part poses and obstacle poses in the other. Results in Tab. 2 show that incorporating simulation data improves real-world performance (increasing task completion from 20-30% to 50-60%). The sim-to-real policy exhibits smoother behavior with fewer erratic movements. Fig. 7 shows examples of successful and unsuccessful attempts. The supplementary video provides more qualitative experiments on sim-to-real transfer and generalizations enabled by synthetic data.

4 Limitations and Conclusion

`ResiP`, while effective for fine-tuning BC-trained policies for precise manipulation tasks, has limitations. Its local residual policies struggle with large-scale deviations and high initial scene randomness. Sim-to-real transfer for RGB policies remains challenging, with performance gaps in distillation and distribution shifts. Future work could explore better sim-to-real techniques, mechanisms for correcting large-scale errors, and generalization to broader initial states. Despite these challenges, `ResiP` successfully uses RL to train residual policies that produce locally corrective actions on complex base models, demonstrating strong performance in fine-tuning imitation-learned assembly policies. Through teacher-student distillation and sim-to-real co-training, we show that precise behaviors can be transferred to real-world assembly policies operating from vision.

References

[1] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[2] K. Kimble, K. Van Wyk, J. Falco, E. Messina, Y. Sun, M. Shibata, W. Uemura, and Y. Yokokohji. Benchmarking protocols for evaluating small parts robotic assembly systems. *IEEE robotics and automation letters*, 5(2):883–889, 2020.

[3] F. Suárez-Ruiz and Q.-C. Pham. A framework for fine robotic assembly. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 421–426. IEEE, 2016.

[4] Y. Lee, E. S. Hu, and J. J. Lim. Ikea furniture assembly environment for long-horizon complex manipulation tasks. In *2021 ieee international conference on robotics and automation (icra)*, pages 6343–6349. IEEE, 2021.

[5] 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.

[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] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

[8] 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`.

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

[10] 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.

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

[12] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 5628–5635. IEEE, 2018.

[13] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[14] 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.

[15] M. Drolet, S. Stepputtis, S. Kailas, A. Jain, J. Peters, S. Schaal, and H. B. Amor. A comparison of imitation learning algorithms for bimanual manipulation. *IEEE Robotics and Automation Letters*, 2024.

[16] J. Ho, A. Jain, and P. Abbeel. Denoising Diffusion Probabilistic Models, Dec. 2020. URL `http://arxiv.org/abs/2006.11239`. arXiv:2006.11239 [cs, stat].

[17] 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].

[18] 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].

[19] 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].

[20] T. Pearce, T. Rashid, A. Kanervisto, D. Bignell, M. Sun, R. Georgescu, S. V. Macua, S. Z. Tan, I. Momennejad, K. Hofmann, and S. Devlin. Imitating Human Behaviour with Diffusion Models, Mar. 2023. URL http://arxiv.org/abs/2301.10677. arXiv:2301.10677 [cs, stat].

[21] L. Lai, A. Z. Huang, and S. J. Gershman. Action chunking as policy compression. *PsyArXiv*, 2022.

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

[23] S. Ross and D. Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668. JMLR Workshop and Conference Proceedings, 2010.

[24] A. Yu, G. Yang, R. Choi, Y. Ravan, J. Leonard, and P. Isola. Lucidsim: Learning agile visual locomotion from generated images. In *8th Annual Conference on Robot Learning*, 2024.

[25] T. Z. Zhao, J. Tompson, D. Driess, P. Florence, S. K. S. Ghasemipour, C. Finn, and A. Wahid. ALOHA unleashed: A simple recipe for robot dexterity. In *8th Annual Conference on Robot Learning*, 2024. URL https://openreview.net/forum?id=gvdXE7ikHI.

[26] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[27] 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].

[28] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[29] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on robot learning*, pages 651–673. PMLR, 2018.

[30] G. B. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal. Rapid locomotion via reinforcement learning. *The International Journal of Robotics Research*, 43(4):572–587, 2024.

[31] 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.

[32] 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.

[33] Y. Park, G. B. Margolis, and P. Agrawal. Automatic environment shaping is the next frontier in rl. *arXiv preprint arXiv:2407.16186*, 2024.

[34] 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.

[35] 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.

[36] 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.

[37] P. J. Ball, L. Smith, I. Kostrikov, and S. Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, pages 1577–1594. PMLR, 2023.

[38] S. Schaal. Learning from demonstration. *Advances in neural information processing systems*, 9, 1996.

[39] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6292–6299. IEEE, 2018.

[40] M. Nakamoto, S. Zhai, A. Singh, M. Sobol Mark, Y. Ma, C. Finn, A. Kumar, and S. Levine. Cal-ql: Calibrated offline rl pre-training for efficient online fine-tuning. *Advances in Neural Information Processing Systems*, 36, 2024.

[41] I. Uchendu, T. Xiao, Y. Lu, B. Zhu, M. Yan, J. Simon, M. Bennice, C. Fu, C. Ma, J. Jiao, et al. Jump-start reinforcement learning. In *International Conference on Machine Learning*, pages 34556–34583. PMLR, 2023.

[42] H. Hu, S. Mirchandani, and D. Sadigh. Imitation bootstrapped reinforcement learning. *arXiv preprint arXiv:2311.02198*, 2023.

[43] Q. Zheng, A. Zhang, and A. Grover. Online decision transformer. In *international conference on machine learning*, pages 27042–27059. PMLR, 2022.

[44] J. Kober and J. Peters. Imitation and reinforcement learning. *IEEE Robotics & Automation Magazine*, 17(2):55–62, 2010.

[45] R. Ramrakhya, D. Batra, E. Wijmans, and A. Das. Pirlnav: Pretraining with imitation and rl finetuning for objectnav. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17896–17906, 2023.

[46] A. Singh, H. Liu, G. Zhou, A. Yu, N. Rhinehart, and S. Levine. Parrot: Data-driven behavioral priors for reinforcement learning. *arXiv preprint arXiv:2011.10024*, 2020.

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

[48] A. Z. Ren, J. Lidard, L. L. Ankile, A. Simeonov, P. Agrawal, A. Majumdar, B. Burchfiel, H. Dai, and M. Simchowitz. Diffusion policy policy optimization. *arXiv preprint arXiv:2409.00588*, 2024.

[49] 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.

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

[51] 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.

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

[53] 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].

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

[55] 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.

[56] 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.

[57] 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.

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

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

[60] 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.

[61] 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.

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

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

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

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

[66] 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.

[67] 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.

[68] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[69] R. Tedrake. *Robotic Manipulation*. Course Notes for MIT 6.421, 2024. URL `http://manipulation.mit.edu`.

[70] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023. doi:10.1109/LRA.2023.3270034.

[71] 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.

[72] 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.

[73] 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.

[74] 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].

[75] 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.

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

[77] 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].

[78] 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.

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

[80] 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.

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

[82] S. Haldar, J. Pari, A. Rai, and L. Pinto. Teach a robot to fish: Versatile imitation from one minute of demonstrations. *arXiv preprint arXiv:2303.01497*, 2023.

[83] S. Lee, Y. Wang, H. Etukuru, H. J. Kim, N. M. M. Shafiullah, and L. Pinto. Behavior generation with latent actions. *arXiv preprint arXiv:2403.03181*, 2024.

[84] N. M. Shafiullah, Z. Cui, A. A. Altanzaya, and L. Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022.

[85] 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.

[86] 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.

[87] 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.

[88] 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.

[89] 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.

[90] 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.

[91] 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.

[92] 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.

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

[94] 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.

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

[96] 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].

[97] 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.

A Problem Setup and Approach Overview

We aim to develop a method for robots to reliably execute long-horizon manipulation tasks with minimal human effort. This section outlines our problem formulation and key assumptions.

Assumptions and System Components Given the observed limitations in scaling performance with human demonstrations (Fig. 1 (Right), [24, 25, 23]), we adopt an approach that combines a moderate set of expert demonstrations ($\sim$50 per task) with RL refinement. We assume tasks consist of a fixed set of rigid objects for which we can access CAD models (either given for an assembly or scanned real-world object). Thus, both the parts and the entire assembly process can be accurately simulated. Each task has an underlying success criterion depending on the required alignments between the parts used to implement the sparse task-completion reward. Each task may have long horizons (up to $\sim$750-1000 steps at 10Hz) and require sequencing of behaviors such as corner alignment, 6-DoF grasping, reorientation, insertion, and screwing (see, e.g., Fig. 1 (Left)). Multi-part assembly interactions are simulated using the SDF-based collision geometry representations featured in the Factory [49] extension of NVIDIA's Isaac Gym simulator [68]. We use the tasks `one_leg`, `round_table`, and `lamp` from the FurnitureBench [5] task suite, the `peg-in-hole` task from [49], and mug-hanging task called `mug-rack` defined through scanning of real-world objects and demonstrations.

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 at 10Hz. 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 [69] controller to convert the desired end-effector pose commands into joint position targets, tracked with a low-level PD controller at 1kHz. 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 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, indicating whether two parts of the assembly have achieved their required geometric alignment.

B Extended methods

B.1 Imitation Learning

We start with a few 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 a Diffusion Policy [19] as the base model, which has shown strong empirical performance in handling difficult manipulation tasks with relatively small datasets [22]. Consistent with recent advancements [19, 6], our policy framework enhances its performance by predicting multiple future actions in chunks instead of 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 $T_{\text{exec}} \leq T_a$.

B.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 [50, 51, 47, 52], 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 [53, 54, 55, 18, 56]. 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 $8\times$), which we find to increase RL training instability (see Sec. 3.1). This means fine-tuning other popular BC architectures, like the Action-Chunked Transformer (ACT) [6], also brings technical challenges [15]. Even with successful RL fine-tuning, the improved policy will still be a trajectory planner lacking in closed-loop reactive control (Fig. 6). 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 [57].

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 find that the policy is not sensitive to α in isolation, but rather the resulting exploration noise, $\alpha\mathcal{N}(0, \sigma^{\text{init std}} I)$. 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 [58][2]. This also allows different prediction horizons T_a for the two policies. This flexibility in action horizon is helpful as most RL algorithms optimize single-action policies and allow the residual to be fully closed-loop. 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 [59, 60].

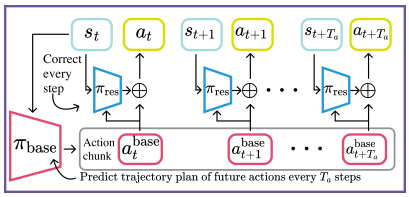


Figure 3: Overview of `ResiP`. A pre-trained chunked base policy predicts an action chunk of T_a future actions. For every timestep, the residual model observes the current state s_t and predicted base action a_t^{base} and corrects.

Given the above, we train the π_{res} using standard PPO [27] 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 [61]. 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 (e.g., Fig. 1 (Left)).

B.3 Sim-to-real pipeline

Ultimately, we are interested in what capabilities our method can enable on real robots. To that end, we devise a sim-to-real pipeline to transfer state-based policies trained in simulation to RGB-based policies deployable on physical hardware. We present the general pipeline in Fig. 4. Once the RL training has converged, the combined policy has attained closed-loop correction behavior. We aim to distill this enhanced performance from the state-based policy into a vision-based policy that operates from RGB images. Following the established teacher-student distillation paradigm [32, 63, 62], we generate a dataset of successful trajectories, $\mathcal{D}_{\text{synth}}$, where the environment's observations, o_t,

[2]This decoupling also means that ACT and other BC methods can serve as the base model without changing the method – an advantage of residual policy learning emphasized by prior robotics work [58, 61].

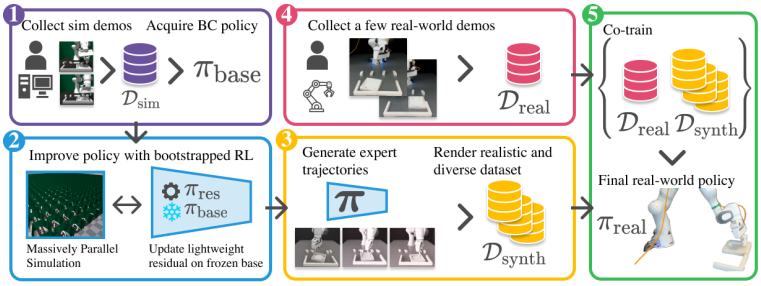


Figure 4: **Sim-to-real pipeline.** (1) Beginning with a policy trained with BC in simulation, (2) we train residual policies with RL and sparse rewards. (3) We then distill the resulting behaviors into a policy operating from RGB images. (4) By combining synthetic data with a small set of real demonstrations, (5) we deploy RGB-based policies in real.

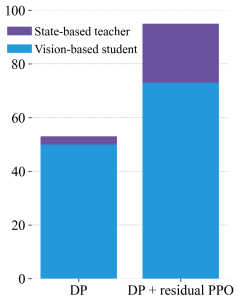


Figure 5: Comparison of distilled performance from BC and RL-based teacher.

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 investigate the effect of combining real-world demonstrations with simulation trajectories from our RL-trained residual policies. Co-training with real and synthetic data improves motion quality and success rate on the `one_leg` task.

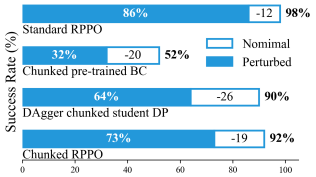


Figure 6: Perturbations show `ResiP`'s resilience: 12% drop vs. 19-26% for chunked methods.

replace the states, i.e., $\mathcal{D}_{\text{synth}} = \{\tau_{\text{synth},1}, ..., \tau_{\text{synth},N}\}$ and $\tau_{\text{synth},i} = \{(o_1, a_1), ..., (o_T, a_T)\}$. For real-world transfer, we enhance the synthetic dataset $\mathcal{D}_{\text{synth}}$ by re-rendering its trajectories in Isaac Sim [70]. 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 with BC.

C Extended Experiments

C.1 Improving Imitation Learning with 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 [62, 63, 31], and offer a natural starting point for RL fine-tuning after BC pre-training. However, MLP policies trained to directly output single action control instead of a trajectory plan through an action chunk (MLP-S) 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 (DP) 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 made subsequent fine-tuning far easier. In one case, `lamp` on low ran-

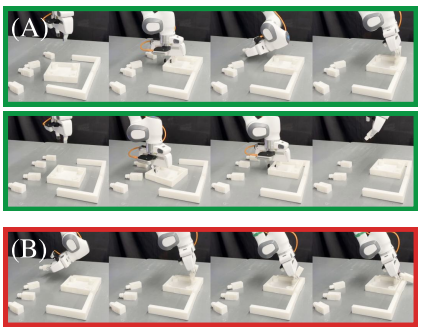


Figure 7: **(A)** Examples of successful real world assembly from RGB. Co-training with simulation data reduces jerkiness and improves insertion robustness by containing a higher diversity of part poses and insertion locations (see Table 2). **(B)** Example failure: difficulty adjusting the insertion pose when grasps lead to unseen in-hand part poses.

domness, MLP-C outperformed DP. In qualitative evaluations, we find that DP has smoother and faster actions, which is generally beneficial. Still, it seems to hurt performance in this case, as it tends to retract before the gripper closes. We also find that all methods struggle with the hardest tasks, on which MLP-C and DP both achieve less than 5% success rate, indicating that there is still a need for stronger BC methods. The `peg-in-hole` task, despite its relatively short horizon of $\sim$100 timesteps, proved particularly challenging for BC methods. This task involves a $\sim$0.2 mm clearance insertion, resulting in a 2-3% success rate. This poor performance on a short yet precise task lends credence to the hypothesis that BC methods are ill-equipped to handle high-precision requirements. Another interesting finding is that the DP is significantly more robust to noise, an important property for an effective base policy for residual fine-tuning. In contrast, we observed performance collapse when using MLP-C as the base. Finally, as shown in Fig. 6, the effect of fully closed-loop control becomes clear when we introduce randomly sampled force perturbations to the episode rollouts. Performance for the `ResiP` policy drops 12 percentage points, compared to 19-26 percentage points for alternative chunked methods (BC, DAgger, chunked residual).

Online Policy Fine-Tuning Comparison Next, we investigate the effectiveness of our residual RL approach, `ResiP`, in improving the performance of policies trained with BC. We train residual policies using PPO [27] 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 [53] 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 [64].

The results in Fig. 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 54% to 98%. Even more drastically, for the `peg-in-hole` task, `ResiP` improved the success rate from 3% to 99%. However, we observe performance saturates at lower success rates (e.g., $\sim$ 70%) for tasks with higher part randomization. We hypothesize that the base model's performance limits the residual policy as it is designed to act locally, constrained by the base.

Fig. 1 (Left) qualitatively shows the most common failure mode of the BC policies and how the RL policy overcomes it. 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. The residual policy regularly corrects these errors by performing small sideways translations while canceling premature downward motions. It typically only allows the leg to be inserted once it is 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. On the other hand, the BC policies more often grasp the object at angles that make insertion more difficult.

In addition to improved success rates, we observed 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 [65], 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 [66, 67].

C.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. A vision-based policy distilled from the RL-0trained

teacher reached 73% on one_leg, outperforming the 50% achieved by training the vision policy directly on human demos, see Fig. 5. 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. 1 (Right), we see that performance increases with more data but still does not reach the performance achieved by the teacher policy, with a gap of 20 percentage points between the best student policy (78% success rate) and the teacher policy (98% success rate) at 100k trajectories (with only a minor improvement from 77% 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.

C.3 Real-World Application

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 only 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 (i.e., insertion location in the workspace).

The results, presented in Fig. 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. 7 shows examples of successful and unsuccessful task attempts. See the supplementary video for more qualitative experiments of sim-to-real transfer and generalizations enabled by synthetic data.

D Implementation Details

D.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 [71]
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 [71]
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 [72] 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 [19] 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 [71]
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 [73]
Encoder Weights	R3M [74]
Encoder Parameters	2×11 million
Encoder Projection Dim	128
Diffusion backbone architecture	Transformer (similar to [19])
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 wrist 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	$320 \times 240 \rightarrow 224 \times 224$

D.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 [71, 75]. 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[71, 75, 76]. 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 [71]) 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 [77, 19]. This normalization method is widely accepted for diffusion models. In [77], 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.

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

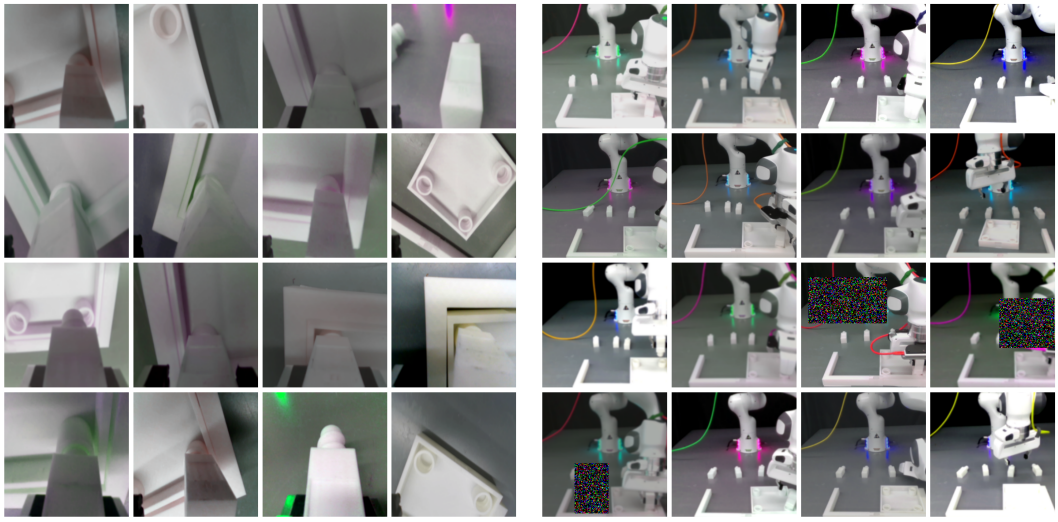


Figure 8: **Left:** Examples of augmentations of the wrist camera view, consisting of color jitter and Gaussian blur. **Right:** Examples of augmentations for the front view also consist of color jitter and Gaussian blur augmentations and random cropping.

E Tasks and Environment

E.1 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`	`mug-rack`	`peg-in-hole`
Mean episode length	∼500	∼700	∼600	∼200	∼75
# Parts to assemble	2	3	3	2	2
Num rewards	1	2	2	1	1
Moving object	✗	✗	✓	✗	✗
# Precise insertions	1	2	1	1	1
# Screwing sequences	1	2	1	0	0
Precise grasping	✗	✓	✗	✗	✗
Insertion occlusion	✗	✓	✗	✓	✗
Insertion tolerance	0.5cm	0.5cm	0.5cm	1.5cm	0.2mm

E.2 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 [5]. However, we found that their method of directly sampling random poses often leads to initial part configurations

Figure 9: Examples of initial scene layouts for `one_leg`, `lamp`, and `round_table` with different levels of initial part pose and obstacle fixture randomness

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 [5]). 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. 9 shows examples of our different randomness levels for each task in simulation.

E.3 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 [5] 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.

E.4 Task description, `mug-rack`

This task involves the robot picking up a coffee mug and hanging it by the handle on one of two pegs on a rack. See Fig. 10 for task illustration and the attached supplementary video, `mug-rack-resip-1.mp4`, for example rollouts. This task is interesting for two main reasons. First, we don't have any CAD models for the objects. Instead, we used scanned imports of real-world objects (obtained with the ARCode app on the iPhone App Store). Second, the task has inherent multi-modality in that the mug can be hung in one of two ways for each of the two pegs.

In practice, we find that the diffusion and residual policy system works well for this task. First, the base diffusion model captures the task's multimodality and sometimes hangs the mug on both pegs. Furthermore, the residual RL procedure keeps this multimodality intact as the base model is frozen. We achieve ∼30% success rate in pre-training and ∼85% with residual fine-tuning.

(a) Example task initialization of the `mug-rack` task.

(b) Example of hanging the mug on the lower rack.

(c) Example of hanging the mug on the upper rack.

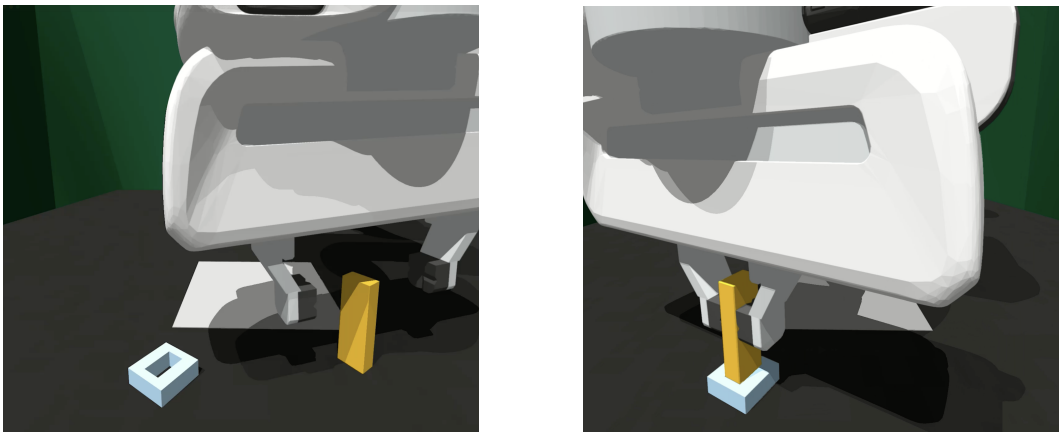
Figure 10: Overview of the new `mug-rack` task we add to showcase the real-to-sim capabilities that one can leverage in tandem with our pipeline. This also shows how reward signals can be inferred directly from data instead of being hand-designed.

E.5 Task description, `peg-in-hole`

To push the limits of precision in simulation, controller, and policy, we pick one of the insertion tasks from the Factory task suite [49], which involves grasping a peg and inserting in a hole with a

0.2mm clearance, i.e., 25x tighter than the FurnitureBench [5] tasks. See Fig. 11 for task illustration and the supplementary video `peg-hole-resip-1.mp4` for example rollouts.

Our approach worked out of the box on this task as well, using the same hyperparameters as for the FurnitureBench tasks. Here, we achieve ∼3% success rate in pre-training and ∼99% in fine-tuning. Good performance at this task is essentially entirely dominated by the ability to locally adjust the peg until it lines up with the hole, and the high final success rate achieved by our approach reflects that the local nature of the corrections learned by our residual policy is well aligned with such task scenarios.



(a) Example task initialization of the `peg-in-hole` task.

(b) Example of task completion when the peg is fully inserted.

Figure 11: Overview of the new `peg-in-hole` task we add to push the requirement for precision. We find that the pipeline as presented works well with the same hyperparameters used for the furniture tasks.

F RGB 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. 12. 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. 13 for examples of front-view images obtained from our domain randomization and re-rendering procedure.

G Visualization of Residual Policy Actions

We hypothesize that the strength of the residual policy is that it can operate locally and make corrections to the base action predicted by the pretrained policy operating on the macro scale in the scene. We show an example of this behavior in Fig. 14. Here, we visualize the base action with the red line, the correction predicted by the residual in blue, and the net action of the combined policy in green.

We find that the residual has indeed learned to correct the base policy's actions, which often leads to failure. One common example is for the base policy to be imprecise in the approach to the hole

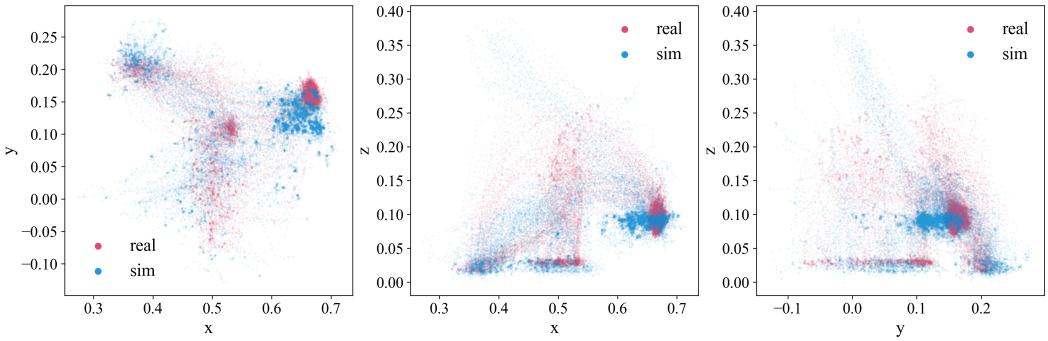


Figure 12: Plots of the x, y, z action coordinates in the demo datasets for the `one_leg` task in the real world and the simulator. That is, each dot represents one action from one of the 40/50 trajectories. Red is from real-world demos, and blue is from the simulator. **Left:** Top-down view, showing 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.

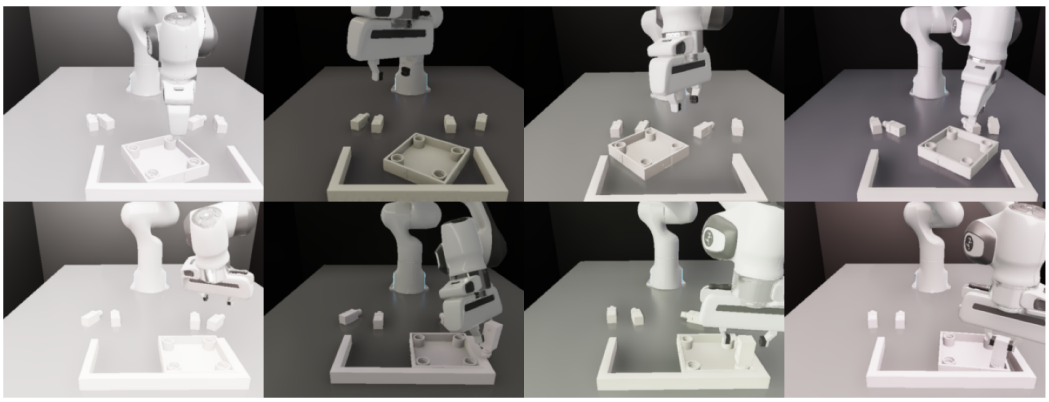


Figure 13: Examples of the randomization applied when rendering out the simulation trajectories used for co-training for the real-world policies.

during insertion, pushing down with the peg not aligned with the hole, causing the peg to shift in the gripper, which leads to a grasp-pose unseen in the training data and the policy inevitably fails. The residual policy counteracts the premature push-down and correct the placement towards the hole, improving task success. See video examples of this behavior on the accompanying website: `https://residual-assembly.github.io/`.

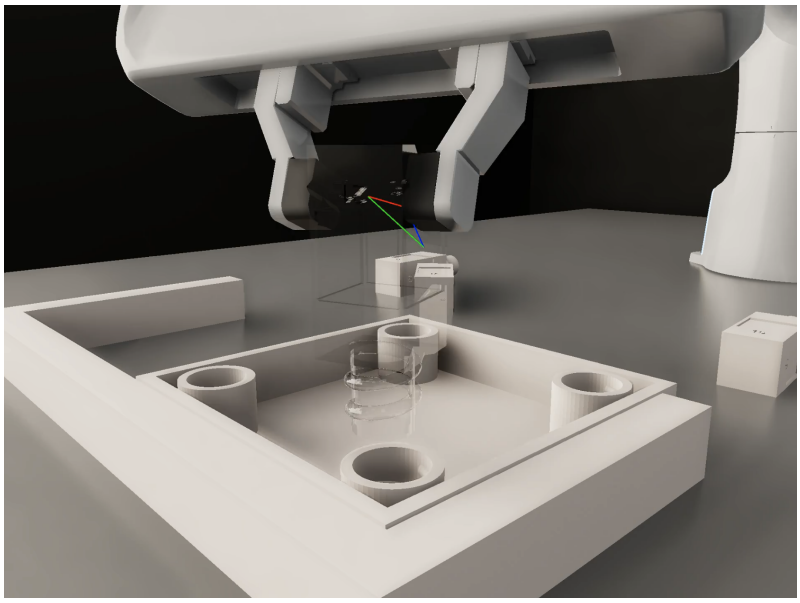


Figure 14: Visualization of the effect of the residual policy during insertion, the phase requiring the most precision. The red line shows the action commanded by the base policy. The blue is the correction predicted by the residual, and the green is the net action. The residual learns to correct actions that typically lead to failure.

H Real-World Results

H.1 Quantitative Results Failure Mode Breakdown

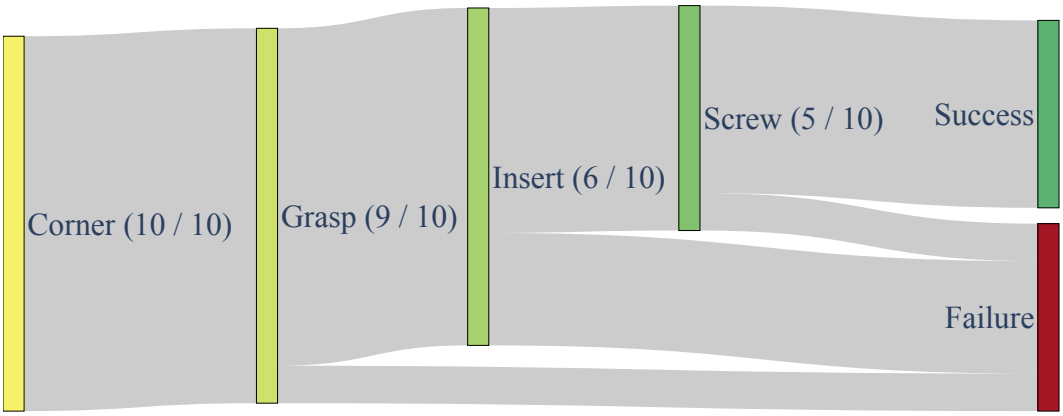


Figure 15: Sankey diagram for the success rate and failure points for the real-world rollouts with 40 real and 350 simulation demos.

The diagram in Fig. 15 shows how successful and failed completion of individual sub-skills along the `one_leg` task amount to our overall final success rates reported in Fig. 2 (bottom row, corresponding to "40 real + 350 sim" with random initial part poses and a fixed obstacle pose).

H.2 Extension of Pipeline to Unseen Settings

Here, we conduct further qualitative experiments to evaluate whether our simulation-based co-training pipeline can make policies more robust to real-world parts with visual appearances that are

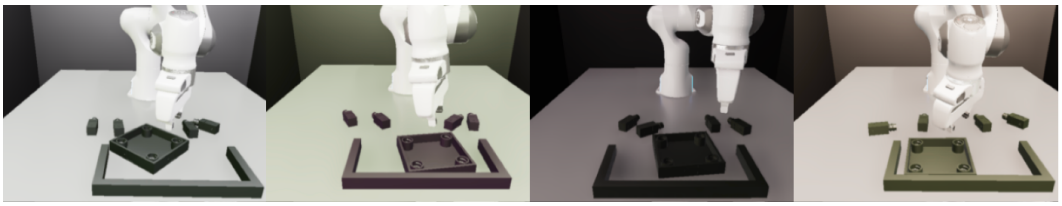


Figure 16: Randomizing the visual appearance of the scene in the simulator allows for more fine-grained control and varying attributes that are hard to isolate in standard image augmentation techniques. Here, we illustrate how we can easily cover a larger space of part appearances without jittering the colors of everything else in the scene in tandem.

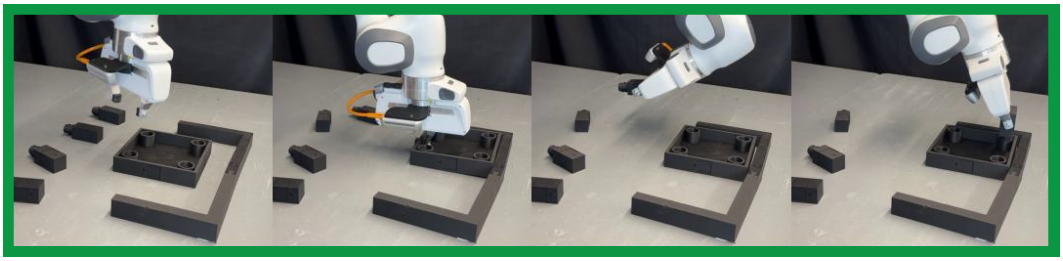


Figure 17: An example of a successful rollout of a policy co-trained on 40 real-world demos containing only white parts and 400 synthetic demos with part colors randomized.

unseen in real world demos. To test this, we 3D printed the same set of parts used in the `one_leg` task in black, and rolled out various policies on these black parts (rather than the white-colored parts used throughout our other experiments). This setting is especially relevant in industrial domains where parts can come in a variety of colors to which the assembly system must be invariant (e.g., the same piece of real-world furniture usually comes in many colors).

When deploying the policy trained on the same 40 demos as in the main experiment, which only had *white*, the policy cannot come close to completing the task. The behavior is highly erratic and triggered the velocity limits of the Franka on every trial we ran. We compare this baseline policy trained on differently colored parts to a policy co-trained on both real and synthetic data from simulation. However, when creating the synthetic dataset for this test, we added in additional randomization of part color, with an emphasis on black or gray colors in this case, as shown in Fig. 17. When we co-train a policy on a mix of the same real-world demos containing *only* white parts as before, with a dataset of 400 synthetic demos with *varying* part colors, the resulting policy can complete the task, as illustrated in Fig. 17 (and even when it fails at the entire task sequence, the predicted motions are much more reasonable than the erratic policy which has overfit to real-world parts of a specific color).

For example videos, please see the accompanying website: `https://residual-assembly.github.io/`. We note, however, that the resulting policy is considerably less reliable than the corresponding policy rolled out with white parts, which illustrates that there is still a meaningful sim2real gap.

I Related Works

Training diffusion models with reinforcement learning The approach in [47, 51] 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 [53], advantage weighted regression [54], or changing the objective into a supervised learning problem with return conditioning [55, 18, 17]. Some have also explored augmenting the de-noising training

objective with a Q-function maximization objective [56] and iteratively changing the dataset with Q-functions [78]. Recent work developed techniques for training diffusion policies from scratch [52], leveraging unsupervised clustering and Q-learning combinations to encourage multi-modal behavior discovery. Our method avoids such complexity involved with directly optimizing diffusion models by using standard PPO to train simple residual policies that 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 been widely successful in robotics. Common frameworks include learning residual policies that correct for errors made by a nominal behavior policy [58, 61, 79, 80, 81, 82, 83, 84] and combining learned components to correct for inaccuracies in analytical models for physical dynamics [85, 86, 87] or sensor observations [88]. Residual policies have been used in insertion applications [89], and recent work has applied residual policy learning to the same FurnitureBench task suite we study in this paper [90]. 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.

J Expanded Related Work

Learning robotic assembly skills Robotic assembly has been used by many as a problem setting for various behavior learning techniques [91, 92, 61, 93, 94]. Enabling assembly that involves multi-skill sequencing (e.g., fixturing $\rightarrow$ grasping $\rightarrow$ insertion $\rightarrow$ screwing) directly from RGB images has remained challenging, especially *without* explicitly defining sub-skill-specific boundaries and supervision. Concurrent work [90] explores a similar framework to ours on FurnitureBench tasks [5], but instead supervises learned policies on a per-skill basis and incorporates 3D point clouds. IndustReal [91] 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 [36, 34]. For instance, demonstrations are often used to support RL in overcoming exploration difficulty and improving sample efficiency [95, 31, 96]. RL can also act as a robustification operator to improve upon base BC behaviors [36, 31], paralleling the RL fine-tuning paradigm that has powered much of the recent advancement in other areas like NLP [97] and vision [47]. Additionally, many successful robotics deployments [32, 63, 62] 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.

K Expanded Conclusions and Limitations

K.1 Conclusion

This work presents an approach for fine-tuning BC-trained policies for precise manipulation tasks using sparse rewards we call `ResiP`. 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 performs well 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 from vision.

K.2 Limitations

The local nature of our residual policies is not well-suited for learning the macro-level behaviors required to recover from large-scale deviations like dropped parts. Our proposed method also 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. Furthermore, despite showcasing the advantage of incorporating simulation data, sim-to-real for RGB policies still presents a challenge. There remains a performance gap in both teacher-student distillation and sim-to-real distribution shifts. Future investigations may include better sim-to-real transfer techniques, exploration mechanisms for discovering how to correct large-scale execution errors, tractable interactive learning for real-world policy distillation, and incorporating inductive biases that help generalize to much broader initial state distributions.

Real-world distillation Our experiments have demonstrated the effectiveness of online learning versus offline or passive learning through behavior cloning. Still, we employ only offline learning in our teacher-student distillation phase for sim-to-real transfer, which will likely upper-bound the performance we can transfer to the real world. Combining our pipeline with techniques for online learning could improve performance significantly. However, at this point, there are significant challenges to overcome to make this practically applicable to the tasks studied herein.

The field is progressing rapidly, and we are excited to investigate how online learning in the real world can be made practical for a broader set of tasks with longer horizons and less obvious ways of performing automatic state resets in follow-up work. This effort further ties into a more general framework for pre-training and adaptation of robot systems where the deployed robot can continue learning and adapting "on the job" after deployment. These investigations complement the methods presented in this paper and are not in scope.

At the same time, our results indicate that making more capable systems only through increasing the collection of real-world demos may also be fundamentally limited unless online learning is introduced as a fine-tuning step in those systems.

Locality of online correction learning Though effective, we re-emphasize that our residual online reinforcement learning framework has the fundamental limitation of being bound to the pre-trained policy and mainly performing locally corrective actions. This limitation is both a strength and a weakness. First, the strong pre-trained prior allows RL to perform the tasks and improve, and having a frozen prior helps stabilize training and prevent collapse. At the same time, the degree to which online learning can generalize to states far from the training set is limited.

Limitations of simulators in contact-rich tasks We have an experiment for a task from the Factory [49] task suite that pushes the simulator's accuracy more than with the original FurnitureBench [5] tasks. This task has a clearance of 0.2mm for the insertion, which shows that the general BC + Residual RL framework also works well in this setting. We did not show, however, that this transfers to the real world, and it would likely be more challenging than in the original tasks for at least two reasons. First, with increased precision requirements, accurate calibration of physics parameters between the actual and simulated environment will likely matter more. Second, performing manipulation from vision when parts are smaller is more challenging.

L Updated RL results section

L.1 Improved task performance

Please see Fig. 1 for our updated success rates achieved using our residual RL + diffusion policy method. In short, we now solve all tasks with >70% success rate, with an average success rate improvement on tasks with low randomness of 73% and 64% for medium randomness.

As we discuss below, in the α parameter ablations (Sec. N.3), the crucial driver of performance with this method (and RL generally) is effective exploration.

We did not come across these improved results earlier because we started out using a conservatively low amount of exploration noise and an unnecessarily high rate for annealing the exploration. The initial conservative hyperparameters were informed by the instability we faced trying to make RL work with a pre-trained MLP policy (i.e., our baseline MLP-C + PPO experiment). However, by experimenting with larger initial noise values and keeping the amount of exploration noise fixed throughout training, we found that the increased exploration ability led to dramatic success rate improvements across low and medium randomness settings in all three assembly tasks.

More aggressive exploration seems possible because we have a robust prior model kept frozen, with a small model optimized online, alleviating some risk of collapse.

M The Surprising Effectiveness of DAgger

DAgger [1] can learn from scratch significantly more efficiently than pure BC regarding gradient steps. DAgger performance compared to BC is in the scaling plot, shown to the right in Fig. 1.

In $\sim$10k steps, it surpasses the BC from 50 human demos trained with $\sim$100k steps. After 10k steps, it has around 800 rollouts in the aggregated dataset. After around 20k gradient steps is when it seems to surpass the best-performing BC distillation runs using more than 10k rollouts and 500k gradient steps, at which point it has $\sim$1.5k demonstrations in the replay buffer.

This result highlights the effectiveness of online and interactive learning as opposed to learning purely passively from an offline dataset. Furthermore, it highlights that the expert we query is an effective teacher. It lastly highlights that for interactive learning to be effective, one needs to have a teacher at the ready to be queried as learning progresses.

N Residual RL ablations

N.1 Effect of fully versus partially closed-loop policies

One differentiating factor of our residual model from some prior work is that the base and residual models make predictions at different frequencies, i.e., every 8 timesteps for the base model and every timestep for the residual model. Making predictions with the most up-to-date information is likely an easier prediction problem, and we expect this to work better than the "standard" setup of letting the residual correct the full output of the base model. When training a residual model that corrects a whole chunk at a time but otherwise uses the same hyperparameters, we observe that training is less sample efficient and performance saturates at a lower success rate. In particular, the chunked residual policy reaches $\sim$85% success rate in about 250 million environment steps, while the one-step residual needs about 75 million.

To further probe the difference between fully closed-loop policies and those using chunking, we evaluate the policies with perturbations added to the parts in the environment throughout the episode. In particular, at each timestep, 1% of parts across the environments will have a random force applied to them. The forces are sampled from the same distribution as the initial part randomization distribution.

We generally see that the partially open-loop policies have a bigger drop in performance when perturbations are introduced, around 20 percentage points compared to 12 for the one-step residual model.

Model	No Perturb	W/ Perturb	Drop in SR
Standard RPPO	98%	86%	12 pp
Chunked RPPO	92%	73%	19 pp
Chunked pre-trained BC	52%	32%	20 pp
DAgger chunked student DP	90%	68%	24 pp

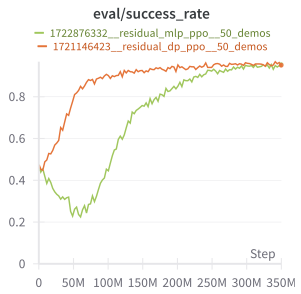
Table 14: Success rates with/without perturbations for different models. SR = Success Rate, pp = percentage points.

N.2 Residual base policy ablation

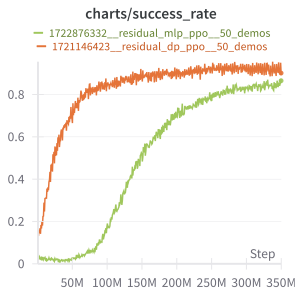
To further tease apart what part of the diffusion policy that provides the most important performance increase, the action chunking or the denoising diffusion process, we run the same residual PPO run for the `one_leg` task as before, but with the best-performing BC MLP model in place of the diffusion policy.

The resulting training dynamics are intriguing. Despite the initial success rate of the base model being close to that of the diffusion model, the success rate drops markedly when exploration noise is introduced. This is especially visible in the training performance in plot 2 below. We also notice that as the residual model explores and learns more, the evaluation performance drops. However, the residual is eventually able to find actions that the MLP responds better to and, in the end, converges to a similar performance as the diffusion-based runs.

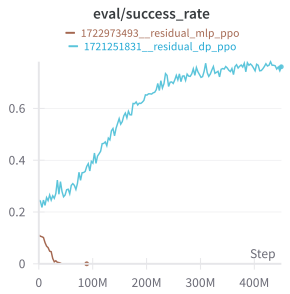
In the harder version of the task with higher initial state randomness, the same initial dynamic plays out, but the training performance drops to zero, causing the learning to collapse. We conclude that any base model achieving a high enough initial success rate can be plugged into our framework (and, based on our BC experiments, a base model with chunking is likely to outperform one without chunking) but that the expressivity and robustness to input noise offered by diffusion de-noising also contributes to downstream performance benefits during residual RL.



(a) Evaluation success rates for RL training for `one_leg`, low randomness.



(b) Success rates in exploration phase of training for `one_leg`, low randomness.



(c) Evaluation success rate for RL training for `one_leg`, medium randomness.

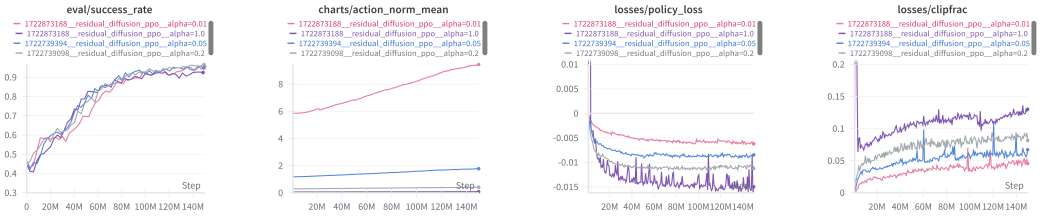
Figure 18: We compare the performance of the diffusion-based residual RL training with using the best-performing MLP as the base model. As we can see, despite having similar pertaining performance (for low randomness), the MLP-based residual model performs poorly compared to the diffusion-based one. On a higher randomness setting, it fails to complete the task.

N.3 Residual action scaling parameter ablation

Two of the reviewers, y2T5 and A6PS, inquired about the significance of the parameter $\alpha = 0.1$. It is a good question and something we should have included in the original submission. The parameter choice is somewhat arbitrary and was informed by some intuitions about the task. I.e., since the intention is for the residual model to make local corrections, we want to imbue it with that inductive bias. In the normalized action space, the workspace is constrained to [-1, 1], and letting a $\sigma = 1$ for the residual Gaussian model correspond to [-0.1, 0.1] on the macro scale seemed reasonable.

In response to the reviewers' requests, we have tested more values of the parameter $\alpha \in \{0.01, 0.05, 0.2, 1.0\}$, but kept the resulting exploration noise on the macro scale fixed (i.e., scaled with the value of α, so $\alpha_1\sigma_1 = \alpha_2\sigma_2$). The result, shown in the figure below, shows a remarkable robustness to this parameter, and all cases have very similar performance.

We note a couple of observations that are still preliminary and that we will investigate further with more runs and harder tasks. First is that $\alpha = 0.2$ seems to perform slightly better than our original $\alpha = 0.1$. Second, different levels of α also result in very different magnitudes of activations at the last layer, which impacts losses. In this experiment, it seems to change the resulting performance much, but we suspect it could make training less stable in harder settings.



(a) Evaluation success rate curves for different levels of α.

(b) Mean norm of predicted action throughout training.

(c) The policy gradient loss development throughout training.

(d) The fraction of time the PPO clipping kicks in throughout training.

Figure 19: We test different values of the residual action scaling parameter α, and test it for values $\alpha \in \{0.01, 0.05, 0.2, 1.0\}$, while adjusting the exploration noise to be such that the macro-level exploration is the same initially. We find that for success rates in this task, the value is not crucial but does cause training dynamics to change, particularly the residual model output norms and policy loss magnitude.

N.4 Data-efficiency ablation

Some reviews have raised the question of whether it would be better to spend the time collecting demonstrations in simulation than collecting them in the real world. We also investigate the flip side of the question of how little data we need to get the pipeline going. In the figure below, we compare using 50 demos for pre-training with 10 (red) and 20 (green).

We naturally see worse performance with fewer demonstrations, but all can improve significantly. Despite this worse performance, this experiment shows that the pipeline is versatile and can use whatever data is available. It also points towards a bootstrapping procedure that can allow reaching the same performance with less data by distilling the initial policy into a stronger base policy and running the procedure again.

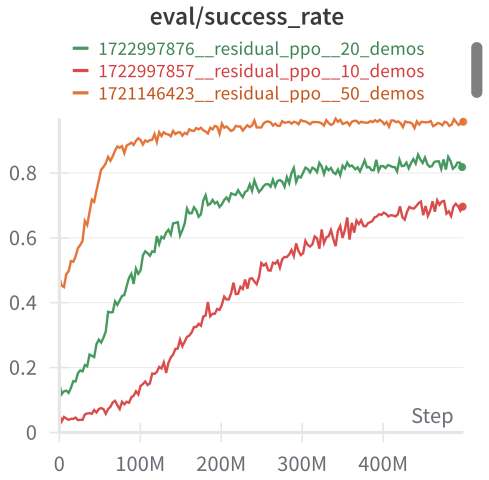


Figure 20: Comparison of online RL optimization for the `one_leg` task on low randomness with BC diffusion policies trained with the original 50 demonstrations (orange), 20 demonstrations (green), and 10 demonstrations (red). We observe worse performance with fewer demonstrations but robust improvement with RL across the board.