BOOST: Bootstrapping Offline and On-policy Successes to Train Real-World Robot Policies

Anthony Liang[*1] **Jesse Zhang**[*2] **Arjun Guru**[2] **Zhanyi Sun**[3]

Aditya Shah[2] **Chaoxiang Zhang**[2] **Abhishek Gupta**[2]

[1]**Toyota Research Institute** [2]**University of Washington** [3]**Stanford University**
anthony.liang@tri.global
[*]Equal contribution

Abstract: Reinforcement learning (RL) can adapt pre-trained robot policies to new environments, but robot interaction is costly. Off-policy RL reuses experience to address this issue, yet its standard updates overlook useful supervision in each action's outcome. We propose using *supervised objectives* derived from transition outcomes to speed up off-policy RL. We introduce **BOOST**, an off-policy residual RL method with three supervised objectives for adapting pre-trained flow policies. A dynamics head grounds the critic in next-observation prediction; a steering head reproduces demonstrated and successful actions through the base policy's input noise; and residual imitation matches these actions unless the critic prefers the current proposal. All three reuse existing data without updating the base policy's weights. In simulation, **BOOST** outperforms prior off-policy fine-tuning baselines overall across ten manipulation tasks spanning four simulation suites. Real-world experiments show that **BOOST** can adapt a pre-trained world-action model (WAM) for dexterous bimanual manipulation.

1 Introduction

Adapting pre-trained robot manipulation policies via reinforcement learning (RL) directly in the deployment environment has become increasingly feasible due to continual improvements in base policy performance [1, 2, 3, 4, 5, 6], with a growing body of work demonstrating how to fine-tune policies with RL on physical robots [7, 8, 9, 10, 11, 12, 13, 14]. Real-world RL is usually described as a data-limited problem: each rollout takes time and human supervision, so the policy cannot rely on the millions of environment steps that make simulated RL work. Yet existing methods may be *wasting useful supervision* contained in the data they already collect. Every transition carries several immediately useful signals—dynamics information, whether it ultimately led to success, and how the taken action compares to the current policy's action. Therefore, the central question is: *how to design RL algorithms that extract the most improvement signal from each transition?*

The standard answer to data scarcity is off-policy RL, which can learn from offline demonstrations and a replay buffer of previously collected trajectories. Most off-policy RL approaches for adapting pre-trained policies share three components: a critic Q, a pre-trained base policy π^{base}, and an RL policy π^{RL} (sometimes identical to π^{base}). Each component is typically trained with a main source of supervision: Q learns by temporal-difference (TD) bootstrapping; π^{base} clones demonstrations; and π^{RL} optimizes Q via policy gradients [15, 16].

However, they all learn slowly: TD learning is slow to converge [17, 18, 19], π^{base} often stays frozen, and π^{RL} updates are often delayed to avoid over-optimizing the slow Q-function [20, 21]. To accelerate sample efficiency, each component needs a second source of training signal, extracted from the transitions already collected, that can be learned via fast-converging *supervised* updates.

We argue that this training signal can be derived from one source—the *outcome* of each action—measured in three ways: what it did next, whether it succeeded, and how it scored.

Following this intuition, we present **BOOST**, an off-policy residual RL algorithm designed for fine-tuning pre-trained flow policies. For each transition, we ask a simple question: *what did taking action a lead to?* First, we ground the TD-learned Q-function with a *dynamics head* that predicts the next state, anchoring it to environment dynamics rather than only to its own moving target [22, 23]. If a led to success, we steer the frozen π^{base} to reproduce a—via behavior cloning in π^{base}'s input-noise space—to improve π^{base} and thus make residual RL easier with limited data. Finally, we train π^{RL} with a *residual imitation* loss that pulls its proposal toward demonstrated and successful actions, using Q to release imitation when the current proposal is preferred, while providing a direct supervised action target rather than relying solely on critic gradients. Therefore, **BOOST** provides each component with a quick-to-learn supervised objective that extracts additional useful signal from each transition, ultimately improving sample efficiency.

We demonstrate **BOOST**'s improvements in sample efficiency across 10 tasks in 4 simulation suites, including adapting a $\pi_{0.5}$ generalist policy. Additionally, we apply **BOOST** to a bimanual YAM system, raising the success rate of a pre-trained generalist policy from 65% to 90% on a precise screwdriver task within 100 online episodes. Finally, our ablations and analysis demonstrate that **BOOST** works especially well when the base policy performance is weak, which is precisely a difficult case for most other RL algorithms.

2 Related Work

RL fine-tuning of pretrained generative robot policies. Noise-parameterized policies have become the standard approach for learning robot manipulation from demonstrations [24, 25, 2], motivating a growing literature on improving pre-trained policies with RL. One line directly fine-tunes the pre-trained diffusion or flow policy. Within this line, some methods apply PPO-style optimization over the multi-step generative process [26, 27, 12], others derive policy-optimization objectives specifically for diffusion or flow-matching policies [28, 29], or retain diffusion- or flow-matching objectives while reweighting the training loss using value or reward information [30, 31].

A separate family avoids applying RL updates directly to the iterative generator, instead optimizing a simpler policy over its latent inputs or action outputs. This includes optimizing latent-space policies over the generator's input noise [9], distilling the generative policy into a one-step actor for value-based RL [32], and learning residual corrections to the base policy's actions [33, 34]. Recent work extends residual RL to pre-trained diffusion-based policies [35, 36, 13, 11]. Much of this literature focuses on adapting RL objectives to diffusion or flow policies. **BOOST** instead makes data reuse a central part of the learning algorithm, using demonstrations and online experience to provide additional supervised signals for both policy and critic learning. This extracts more training signal from each collected transition, targeting the sample-efficiency bottleneck in real-world RL. This data-efficiency issue is increasingly important for VLA post-training, where recent work uses RL to fine-tune large pre-trained robot policies from limited interaction [37, 38, 39, 14, 10, 40]. **BOOST** is especially applicable in this setting due to the use of lightweight noise steering and residual policies to quickly adapt such large policies.

Demonstration-augmented online RL. Demonstration data has long been used to reduce the sample cost of online RL. Early off-policy methods retain expert transitions throughout online learning [41, 42, 43]; some additionally add behavior-cloning losses toward demonstrated actions [41, 43]. Offline-to-online RL broadens this setting to using suboptimal prior data to initialize and accelerate subsequent online learning. Prior approaches use advantage-weighted policy regression on replayed data [44], balanced offline–online replay with stabilized critic learning [21], or calibrated offline Q-functions that avoid overestimation during early online fine-tuning [45]. Complementary work augments model-free RL with auxiliary prediction objectives, including reward and next-state-embedding prediction [46, 22] and multi-step future-latent prediction [47]. Across these lines of work, offline and online experience is reused through actor–critic updates, behavior-

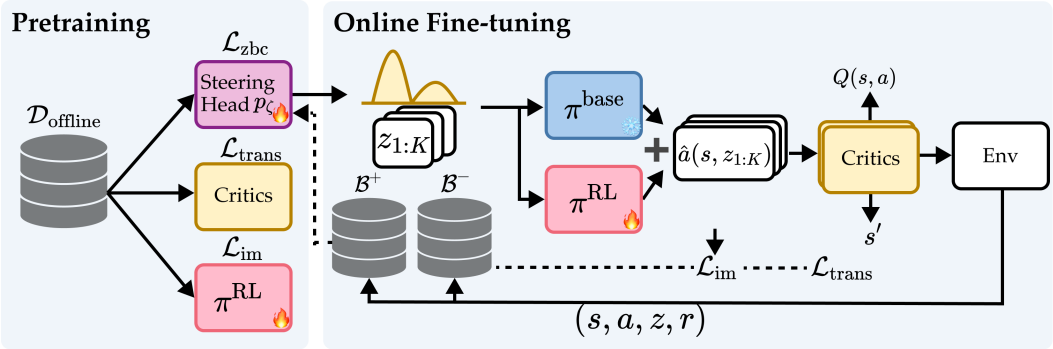


Figure 1: **BOOST training and acting loop. (1) Pre-training:** demonstrations train critic features through next-state prediction, the residual through action imitation, and the steering head through inverted action-to-noise targets. **(2) Online fine-tuning:** sample K noises from $p_\zeta(\cdot \mid s)$, generate base actions, add residual corrections to form $\hat{a}(s, z)$, and execute the highest-valued chunk. All transitions supply dynamics targets; demonstrations and successful episodes supply residual imitation targets (dotted) and steering targets (dashed). The base policy's weights remain frozen throughout.

cloning or imitation losses, and auxiliary prediction objectives. Complementary to these methods, **BOOST** adds additional supervised objectives to off-policy RL by re-using both demonstration and online interaction data: all transitions provide next-observation targets for critic learning; demonstrations and online successes provide value-gated imitation targets for the residual; and the same successful actions are inverted into input-noise targets for the steering head. Thus, the same experience directly improves the critic, residual actor, and the base policy's future action proposals.

3 BOOST: Off-Policy RL with Supervised Anchors

We now introduce **BOOST**, an off-policy residual RL fine-tuning algorithm that extracts *supervised* objectives from existing trajectory outcomes, enabling more sample-efficient learning. **BOOST** adds three supervised anchors to off-policy residual RL: next-state prediction for the critic, successful-action cloning in the base's noise space, and value-gated action imitation for the residual. Figure 1 summarizes the training and acting loop.

3.1 Preliminaries

Problem setting. We aim to enable RL improvement of a policy $\pi^{\text{base}}(a|s)$ pre-trained on expert offline demonstrations $\mathcal{D}_{\text{offline}}$, taking (chunked) actions a given state $s = \{s_{\text{lang}}, s_{\text{img}}, s_{\text{proprio}}\}$ consisting of language, image, and proprioception. The goal is to maximize environment return with minimal interaction, using a residual policy π^{RL} that modifies π^{base}'s action outputs.

Noise-parameterized policies. While **BOOST**'s residual RL formulation works with arbitrary base policies, our experiments assume noise-parameterized flow or diffusion policies [48], common in modern policy architectures [24, 3, 6], that map noise $z \sim N(0, I)$ to actions, i.e., $\pi^{\text{base}}(a \mid s, z)$.

Residual RL. We build on DICE [13], an off-policy residual RL algorithm. DICE freezes π^{base}, conditions the residual on its input noise z, and trains a residual policy π^{RL} off-policy with a critic ensemble over N critics. To act, DICE samples K noises $z_{1:K} \sim N(0, I)$, corrects each base proposal with the residual, and executes the chunk with the highest conservative value $\bar{Q}$. We use the following notation throughout:

$$\begin{aligned} &\textbf{Corrected chunk} && \hat{a}(s, z) = \pi^{\text{base}}(s, z) + \pi^{\text{RL}}(s, z), && \triangleright \textit{Base proposal plus residual correction} \\ &\textbf{Ensemble value} && \bar{Q}(s, a) = \min_{i \in 1 \ldots N} Q^i(s, a). && \triangleright \textit{Most conservative of } N \textit{ critics} \end{aligned} \tag{1}$$

DICE trains the residual with a TD3+BC-style [49] objective that maximizes the conservative value of the corrected chunk and adds a norm penalty on the residual magnitude, $\mathcal{L}_{\mathrm{DICE}}(\theta) = \mathbb{E}_{s\sim\mathcal{D},\, z\sim N(0,I)}\big[-\bar{Q}(s,\hat{a}(s,z)) + \beta\|\pi^{\mathrm{RL}}(s,z)\|_2^2\big]$, and the critics with a chunk-level TD loss. We write the one-chunk target for simplicity:

$$\mathcal{L}_{\mathrm{Q}}(\phi) = \mathbb{E}_{\mathcal{D}}\Big[\sum_{i=1}^{N}\big(Q^i(s_t,a_t) - y\big)^2\Big], \quad y = \sum_{j=0}^{H-1} r_{t+j} + \gamma\, \mathbb{E}_{z'}\big[\bar{Q}'(s_{t+H}, \hat{a}(s_{t+H}, z'))\big], \tag{2}$$

where a_t is an H-step action chunk, Q' is the target critic, $z' \sim N(0, I)$, and γ discounts between chunks. $\mathcal{D}$ mixes offline and online data. DICE samples batches RLPD [21]-style from offline and online data; we adopt these choices and add one supervised objective per component.

3.2 RL Fine-Tuning with BOOST

Overview. **BOOST** retains DICE's residual RL updates and adds supervision from three action outcomes: (1) **next state → critic:** predict chunk-end observations; (2) **success → base:** learn an input-noise distribution that reproduces demonstrated and successful actions, keeping base weights frozen; (3) **value → residual:** imitate demonstrated or successful actions, releasing imitation when the critic prefers the current proposal. Online replay contains both successes and failures. For notational clarity, we explicitly write out two replay buffers: $\mathcal{B}^+$ contains successful trajectories, and $\mathcal{B}^-$ contains failed trajectories.

When offline data $\mathcal{D}_{\mathrm{offline}}$ is available, **BOOST** begins with a pre-training phase that trains the residual with residual imitation, the critic with its dynamics objective, and the steering head on demonstration latents.[1] Online fine-tuning combines these objectives with DICE's off-policy RL objectives. We detail each supervised objective below.

Next state: grounding the critic. TD learning can converge slowly [17, 18] and produce collapsed, low-rank critic features [19] that limit critic expressivity. Under sparse rewards in particular, these features are shaped almost entirely by bootstrapped targets. Therefore, next-state prediction supplies a direct supervised dynamics target—prior work links it to improved performance [51, 52] and reduced feature rank collapse [23]. We train one single-layer (i.e., affine transformation) **dynamics head** g_{ξ_i} per critic on its penultimate features $\psi_{\phi_i}(s,a)$ to predict the state $s' = s_{t+H}$ at the end of the action chunk, applied to all transitions:[2]

$$\mathcal{L}_{\mathrm{trans}}(\phi,\xi) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}_{\mathrm{offline}}\cup\mathcal{B}^+\cup\mathcal{B}^-}\Big[\sum_{i=1}^{N}\big\|g_{\xi_i}\big(\psi_{\phi_i}(s,a)\big) - s'\big\|^2\Big]. \tag{3}$$

Success: steering the frozen base. Poor base proposals are difficult to correct with regularized residuals. We therefore clone demonstrated and successful actions in the frozen base's input-noise space. Instead of updating the base policy's weights, we invert the frozen base to recover noise targets $z^\star = \mathrm{invert}_{\pi^{\mathrm{base}}}(s,a)$ that reconstruct these actions [53, 54]. A small flow-matching **steering head** p_ζ learns to generate these targets from Gaussian noise. Its velocity field v_ζ is trained with the following loss, where τ denotes flow time rather than an environment step:

$$\mathcal{L}_{\mathrm{zbc}}(\zeta) = \mathbb{E}_{(s,a)\sim\mathcal{D}_{\mathrm{offline}}\cup\mathcal{B}^+,\, z_0\sim N(0,I),\, \tau\sim U[0,1]}\big\|\underbrace{v_\zeta(z_\tau \mid s,\tau)}_{\text{Predicted velocity}} - \underbrace{(z^\star - z_0)}_{\text{Target velocity}}\big\|^2, \qquad z_\tau = \underbrace{(1-\tau)\,z_0 + \tau\, z^\star}_{\text{Interpolated noise}}. \tag{4}$$

Flow matching accommodates multimodal noise targets, but relying on learned steering too early can overfit to a few trajectories. Following prior work on Gaussian exploration [55], we blend a

[1] We opt not to pre-train Q with TD learning due to prior findings that doing so generally does not correlate with improved fine-tuning [50, 14].

[2] With image inputs, the target is the stored next observation representation: frozen image features and proprioception, plus privileged critic features where used.

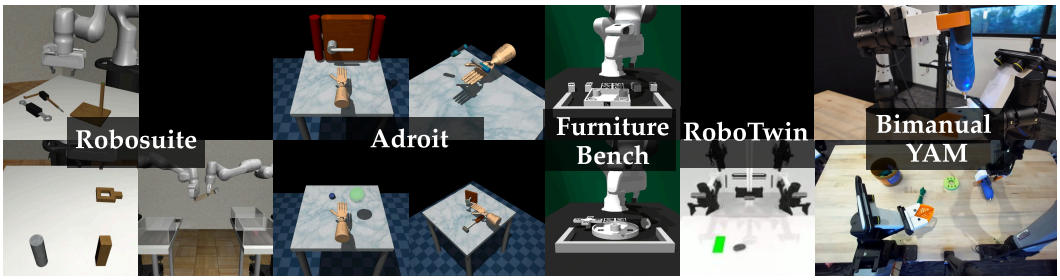


Figure 2: **Evaluation environments** across 4 simulation suites and a real robot environment. The suites test image-based, dexterous, long-horizon, and generalist-policy RL improvement.

head sample $\tilde{z}$ with independent Gaussian noise: $z = \alpha\tilde{z} + (1-\alpha)\epsilon$, where $\epsilon \sim N(0, I)$. Increasing α from 0 to 1 gradually introduces learned steering. Throughout, $p_\zeta(\cdot \mid s)$ denotes the resulting noise distribution, with the blend schedule and the head's exponential moving average (EMA) used for sampling left implicit. Appendix B gives the schedule details.

Value: anchoring the residual. In off-policy RL algorithms, actor updates are often delayed to avoid over-optimizing the critic [20, 21, 56, 13, 57], especially when it is trained without conservatism [58]. To reduce reliance on critic gradients, we directly supervise the corrected action with demonstrated and successful actions. For each (s, a) from $\mathcal{D}_{\text{offline}} \cup \mathcal{B}^+$, we sample $z \sim p_\zeta(\cdot \mid s)$ and form the corrected action $\hat{a}(s, z)$. The conservative gate keeps imitation unless the live minimum proposal value exceeds the data action's reference value Q_{ref}, allowing RL to improve beyond the data:

$$\mathcal{L}_{\text{im}}(\theta) = \mathbb{E}_{(s,a)\sim\mathcal{D}_{\text{offline}}\cup\mathcal{B}^+,\, z\sim p_\zeta(\cdot|s)}\Big[\underbrace{w(s,a,z)}_{\text{imitation gate}} \big\| \underbrace{\hat{a}(s,z) - a}_{\text{residual imitation}} \big\|^2 \Big]. \tag{5}$$

Here $w = \mathbb{I}[Q_{\text{ref}}(s,a) \geq \bar{Q}(s, \hat{a}(s,z))]$. During residual pre-training, however, $w = 1$ for all samples, since the critic is not yet well calibrated.

Summary. **BOOST** improves π^{base}, π^{RL}, and Q with supervised objectives that help them adapt quickly to new transitions to improve sample efficiency. **BOOST** retains DICE's best-of-K selection, gradually replacing Gaussian proposals with learned steering as α increases:

$$\underbrace{z_{1:K} \overset{\text{iid}}{\sim} p_\zeta(\cdot \mid s)}_{\text{Steer } K \text{ noises}}, \quad \underbrace{a_k = \hat{a}(s, z_k)}_{\text{Correct proposal (eq. (1))}}, \quad \underbrace{k^\star = \arg\max_{k \in 1\ldots K} \bar{Q}(s, a_k)}_{\text{Select best of } K}, \quad \underbrace{a^\star = a_{k^\star}}_{\text{Execute chunk}}, \tag{6}$$

Learning combines the off-policy objectives with one supervised objective per component:

$$\begin{aligned} \text{critic:} \quad & \mathcal{L}_{\text{Q}}(\phi) + \mathcal{L}_{\text{trans}}(\phi, \xi), \\ \text{residual:} \quad & \mathbb{E}_{s\sim\mathcal{D},\, z\sim p_\zeta(\cdot|s)}\big[-\bar{Q}(s, \hat{a}(s,z)) + \beta \|\pi^{\text{RL}}(s,z)\|_2^2 \big] + \lambda\, \mathcal{L}_{\text{im}}(\theta), \\ \text{steering head:} \quad & \mathcal{L}_{\text{zbc}}(\zeta), \end{aligned} \tag{7}$$

where colored components denote changes from DICE. For hyperparameters, see Appendix B, and Algorithm 1 for algorithm pseudocode.

4 Experiments

Our experiments investigate three questions: (1) Does **BOOST** learn faster and reach higher success than prior fine-tuning methods, and on which tasks are its gains largest? (2) How robust is **BOOST** to weaker base policies, and how much does learned steering contribute? (3) Can **BOOST** adapt

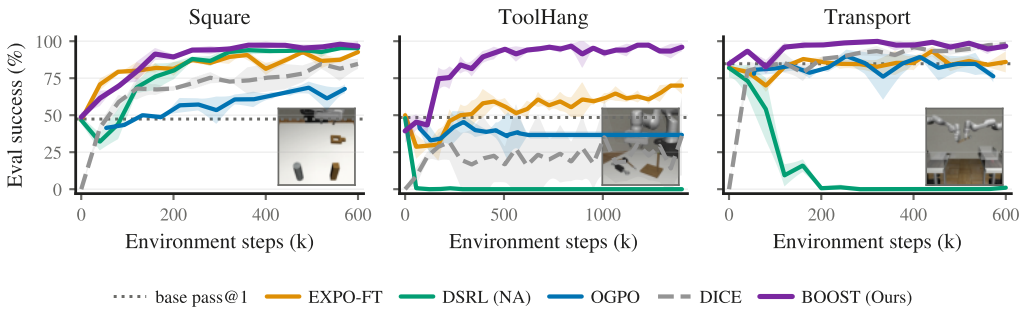


Figure 3: **Robosuite Results.** **BOOST** is the only method above 90% on all three Robosuite tasks and gets there 2–10× faster than the closest baseline.

pre-trained generalist policies to bimanual manipulation in simulation and on a real robot? We compare fine-tuning methods across Robosuite, Adroit, and FurnitureBench, then study generalist-policy adaptation with $\pi_{0.5}$ on RoboTwin and Flex-π on a real bimanual YAM robot.

4.1 Experimental Setup

Simulated environments. We evaluate on four simulation suites addressing different challenges in RL (see Figure 2). **Robosuite** [59]: image-based `Square`, `ToolHang`, and two-arm `Transport`, which require precise peg insertion, a 700-step multi-stage assembly that threads and hangs a tool, and bimanual hand-off, respectively. **Adroit** [60]: `Door`, `Hammer`, `Pen`, and `Relocate` with a 24-DoF hand from state, testing dexterous manipulation. **FurnitureBench** [61]: image-based `One Leg` and `Round Table` assembly in FurnitureSim, long-horizon tasks that chain grasping, lifting, aligning, and inserting table legs with millimeter-level tolerance over up to 700 steps. **RoboTwin** [62]: bimanual manipulation with a generalist policy ($\pi_{0.5}$) base on a `Place Mouse Pad` task.

Real robot. We use Flex-π [6], a 6B-parameter world-action model, on a bimanual YAM robot fine-tuned to use an electric screwdriver with one arm to screw in its own gripper held by the other. This is a precise, contact-rich bimanual task with sub-millimeter alignment clearance. **BOOST** and DICE run on top of the frozen Flex-π policy in its action-only mode (faster inference).

Baselines. We compare against residual, noise-steering, and full base policy fine-tuning baselines. DICE proposes K candidate action chunks from prior noise, corrects each with a learned residual, and selects with the critic ensemble, without the dynamics head, the imitation loss, or the steering head. EXPO-FT is an off-policy residual method: a small edit policy perturbs base samples, the critic scores base and edited candidates, and the base flow itself is fine-tuned each iteration. DSRL (NA) is a noise-aliasing form of DSRL: an action critic Q_A is trained by temporal-difference learning on real actions with a noise critic Q_W distilled from Q_A, and the actor maximizes Q_W without updating the frozen base. OGPO [12] fully fine-tunes the flow policy online to optimize $Q(s, a)$ and stabilizes the update with a success-buffer behavior-cloning term.

Every baseline shares the same frozen flow-matching base policy, image features, demonstrations, interaction budget, and evaluation protocol. Per-task details and hyperparameters are in Appendix B; the final numbers behind every figure are in Appendix C.1. All simulation results are averaged over 3 seeds and reported with their standard deviation, and the dotted line in every plot is the frozen base policy's success rate before fine-tuning.

4.2 BOOST Fine-Tunes Faster and Reaches Higher Success Than Prior Methods

Across the ten simulated tasks, **BOOST** consistently achieves competitive final success while learning faster than prior fine-tuning methods. On Robosuite (Figure 3), it is the only method to exceed 90% on all three tasks, reaching that level with 2–10× fewer environment steps than the closest

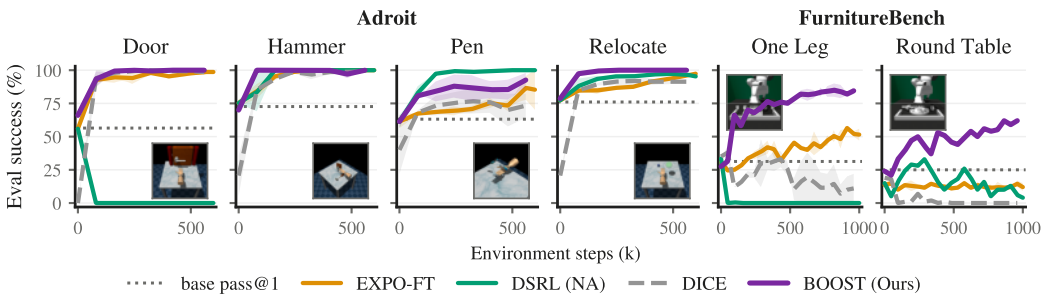


Figure 4: **Adroit and FurnitureBench Results.** BOOST learns quickly and reaches high success on Adroit, with the largest gains over baselines on FurnitureBench. Base pass@1 denotes the frozen base policy's single-attempt success rate.

baseline on `Square` and `Transport`. On Adroit and FurnitureBench (Figure 4), it reaches high success on the dexterous tasks and shows its largest gains on long-horizon assembly.

Where does BOOST help most? The clearest gains occur where the base policy struggles to produce useful behavior. On `ToolHang`, BOOST exceeds 90% within 400k steps, while EXPO-FT plateaus around 70% and DICE makes little progress. On `Pen`, BOOST reaches 94% success, compared with DICE's 73%. The gap is largest on FurnitureBench `One Leg`: BOOST raises success from 31% to 85%, with gains appearing within 50k steps, while the strongest baseline, EXPO-FT, reaches 52% and DICE and DSRL finish below the frozen base's success rate.

These results also highlight differences between adaptation strategies. DSRL learns quickly on some tasks but collapses on others; EXPO-FT is the strongest baseline overall, while OGPO learns more slowly under the same interaction budget. Together, the comparisons suggest that residual corrections and learned steering can offer a sample-efficient alternative to full-policy fine-tuning. We next examine how steering contributes when starting from a weaker base policy.

4.3 Learned Steering Matters More with a Weaker Base Policy

We repeat the Robosuite experiments from an earlier base checkpoint, with success rates of 37%, 41%, and 38% on `Square`, `ToolHang`, and `Transport`, respectively (Figure 5). Despite the weaker starting point, BOOST reaches 94–99% success across all three tasks. Baselines are less consistent: on `Square`, EXPO-FT falls from 93% to 68%, while on `Transport`, DICE falls from 98% to 61%.

This result comes mainly from the noise steering head. Removing the steering head lowers success on all three tasks. This suggests that learned steering is most useful when the base's proposals are poor: it changes the input-noise distribution to reproduce successful behavior, giving the residual better actions to correct over time and making up for lower base policy performance.

4.4 BOOST Adapts Pre-Trained Generalist Policies

We next study how BOOST adapts pre-trained generalist policies to specific bimanual manipulation tasks, using $\pi_{0.5}$ in simulation and Flex-π on a physical robot. Both settings retain the base policy's weights and learn lightweight steering and residual corrections.

RoboTwin with $\pi_{0.5}$. We evaluate adaptation of $\pi_{0.5}$ [63] on a bimanual manipulation task in RoboTwin 2.0 [62]. This setting tests whether BOOST can adapt a generalist vision-language-action policy to task-specific behavior without updating its weights. Figure 6 compares BOOST with DICE and DSRL on `Place Mouse Pad`. Within 150k environment steps, BOOST raises $\pi_{0.5}$'s success from 46% to

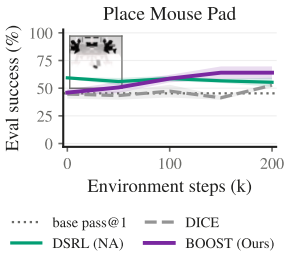


Figure 6: **RoboTwin adaptation** with $\pi_{0.5}$.

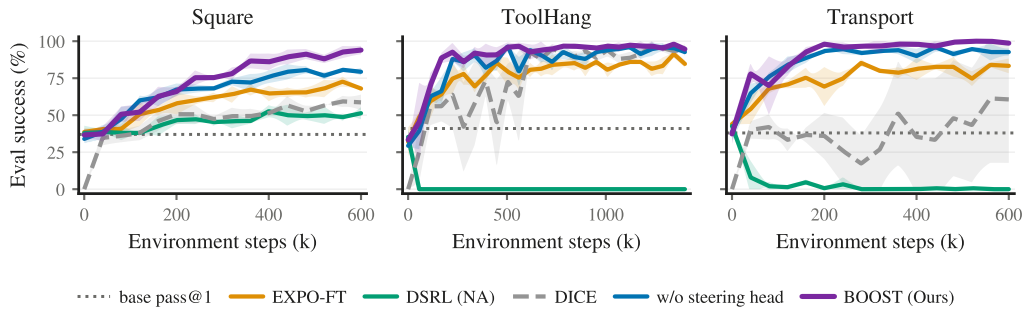


Figure 5: **BOOST remains effective with a weaker base policy.** Starting from an earlier checkpoint, **BOOST** reaches 94–99% success on Robosuite. Removing steering costs 15 percentage points on `Square` and 6 on `Transport`, compared with 2 and 0 from the main checkpoint.

64%, whereas DICE stays at the base policy's level and DSRL stays flat near 57% throughout.

Real robot with Flex-π. We adapt Flex-π [6], a pre-trained world-action model, on the bimanual YAM robot. The task requires one arm to use an electric screwdriver to screw in the gripper held by the other, testing precise, contact-rich coordination. Figure 8 in Appendix C.2 shows synchronized views of the task from all three cameras. Setup details are in Appendix B.3 (Table 8). We run **BOOST** for 104 online episodes performing 5 gradient steps per executed action chunk at 30 Hz, measuring final success and training success rates throughout in Figure 7). **BOOST** improves Flex-π's base performance from 65% success rate to 90% over 104 RL episodes, measured over 20 evaluation episodes per checkpoint, and improves both median time per success (13.5 s $\rightarrow$ 10.2 s) and throughput (2.59 successes / min $\rightarrow$ 3.82 successes / min).

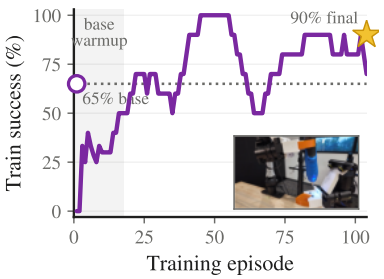


Figure 7: **Real-robot adaptation of Flex-π.** Rolling 10-episode training success on the bimanual YAM screwdriver task; eval success rises from 65% to 90% in 104 episodes.

5 Conclusion

We presented **BOOST**, an off-policy residual RL approach for fine-tuning pre-trained policies that turns the outcomes of the policy's own experience into supervised objectives for every component of the learner: a dynamics head that grounds the critic in dynamics, a residual imitation loss that anchors the residual to demonstrated and successful actions, and a steering head that folds successes back into the base policy's input noise. We demonstrated in four simulated manipulation suites and on a real bimanual YAM robot that **BOOST** fine-tunes faster and reaches higher success than prior residual and off-policy methods, with the largest gains on long-horizon and dexterous tasks where the base policy performs poorly. We see **BOOST** as a step toward generalist robot policies that keep improving from every success they collect in the real world.

References

[1] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, et al. RT-1: Robotics transformer for real-world control at scale. In *Robotics: Science and Systems (RSS)*, 2023. arXiv:2212.06817.

[2] M. J. Kim, K. Pertsch, S. Karamcheti, et al. OpenVLA: An open-source vision-language-action model. In *Conference on Robot Learning (CoRL)*, 2024.

[3] H. Fang, J. Duan, D. Clay, S. Wang, S. Liu, W. Huang, X. Fan, W.-C. Tsai, S. Chen, Y. R. Wang, S. Xing, J. Cho, J. S. Park, A. Eftekhar, P. Sushko, K. Farley, A. Wadhwa, C. Harrison, W. Han, Y.-C. Lee, E. VanderBilt, R. Hendrix, S. Ellawela, L. Ngoo, J. Chai, Z. Ren, A. Farhadi, D. Fox, and R. Krishna. Molmoact2: Action reasoning models for real-world deployment. *arXiv preprint arXiv:2605.02881*, 2026.

[4] S. Ye, Y. Ge, K. Zheng, S. Gao, S. Yu, G. Kurian, S. Indupuru, Y. L. Tan, C. Zhu, J. Xiang, A. Malik, K. Lee, W. Liang, N. Ranawaka, J. Gu, Y. Xu, G. Wang, F. Hu, A. Narayan, J. Bjorck, J. Wang, G. Kim, D. Niu, R. Zheng, Y. Xie, J. Wu, Q. Wang, R. Julian, D. Xu, Y. Du, Y. Chebotar, S. Reed, J. Kautz, Y. Zhu, L. J. Fan, and J. Jang. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

[5] P. Intelligence, B. Ai, A. Amin, R. Aniceto, A. Balakrishna, G. Balke, K. Black, G. Bokinsky, S. Cao, T. Charbonnier, V. Choudhary, F. Collins, K. Conley, G. Connors, J. Darpinian, K. Dhabalia, M. Dhaka, J. DiCarlo, D. Driess, M. Equi, A. Esmail, Y. Fang, C. Finn, C. Glossop, T. Godden, I. Goryachev, L. Groom, H. Habeeb, H. Hancock, K. Hausman, G. Hussein, V. Hwang, B. Ichter, C. Jacobsen, S. Jakubczak, R. Jen, T. Jones, G. Kammerer, B. Katz, L. Ke, M. Khadikov, C. Kuchi, M. Lamb, D. LeBlanc, B. LeCount, S. Levine, X. Li, A. Li-Bell, V. Lialin, Z. Liang, W. Lim, Y. Lu, E. Luo, V. Mano, N. Marwaha, A. Mongush, L. Murphy, S. Nair, T. Patterson, K. Pertsch, A. Z. Ren, G. Schelske, C. Sharma, B. Shi, L. X. Shi, L. Smith, J. T. Springenberg, K. Stachowicz, W. Stoeckle, J. Tang, J. Tanner, S. Tekeste, M. Torne, K. Vedder, Q. Vuong, A. Walling, H. Wang, J. Wang, X. Wang, C. Whalen, S. Whitmore, B. Williams, C. Xu, S. Yoo, L. Yu, W. Zhang, Z. Zhang, and U. Zhilinsky. $\pi_{0.7}$: a steerable generalist robotic foundation model with emergent capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

[6] G. Yan, J. Liu, Y. Fan, L. Cai, M. Liao, J. Zhang, and D. Fox. Flex-π: A multi-stream world-action model with compute flexibility. *arXiv preprint arXiv:2608.10860*, 2026.

[7] J. Luo, C. Xu, J. Wu, and S. Levine. Precise and dexterous robotic manipulation via human-in-the-loop reinforcement learning. *Science Robotics*, 10(105):eads5033, 2025.

[8] P. Yin, T. Westenbroek, S. Bagaria, K. Huang, C.-A. Cheng, A. Kolobov, and A. Gupta. Rapidly adapting policies to the real-world via simulation-guided fine-tuning. In *International Conference on Learning Representations (ICLR)*, 2025.

[9] A. Wagenmaker, M. Nakamoto, Y. Zhang, S. Park, W. Yagoub, A. Nagabandi, A. Gupta, and S. Levine. Steering your diffusion policy with latent space reinforcement learning. *Conference on Robot Learning*, 2025.

[10] P. Intelligence, A. Amin, R. Aniceto, A. Balakrishna, K. Black, K. Conley, G. Connors, J. Darpinian, K. Dhabalia, J. DiCarlo, D. Driess, M. Equi, A. Esmail, Y. Fang, C. Finn, C. Glossop, T. Godden, I. Goryachev, L. Groom, H. Hancock, K. Hausman, G. Hussein, B. Ichter, S. Jakubczak, R. Jen, T. Jones, B. Katz, L. Ke, C. Kuchi, M. Lamb, D. LeBlanc, S. Levine, A. Li-Bell, Y. Lu, V. Mano, M. Mothukuri, S. Nair, K. Pertsch, A. Z. Ren, C. Sharma, L. X. Shi, L. Smith, J. T. Springenberg, K. Stachowicz, W. Stoeckle, A. Swerdlow, J. Tanner, M. Torne, Q. Vuong, A. Walling, H. Wang, B. Williams, S. Yoo, L. Yu, U. Zhilinsky, and Z. Zhou. $\pi^*_{0.6}$: A vla that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025.

[11] P. Dong, Q. Li, D. Sadigh, and C. Finn. EXPO: Stable reinforcement learning with expressive policies. *arXiv preprint arXiv:2507.07986*, 2025.

[12] S. Patil, M. Nakamoto, M. Agarwal, S. Saxena, J. Zhang, G. Anantharaman, C. Winston, C. Pan, D. Chen, N.-C. Huang, Z. Temel, O. Kroemer, S. Levine, A. Gupta, H. Dai, P. Shah, and M. Simchowitz. OGPO: Sample efficient full-finetuning of generative control policies. In *International Conference on Machine Learning*, 2026.

[13] Z. Sun and S. Song. From prior to pro: Efficient skill mastery via distribution contractive rl finetuning. In *International Conference on Machine Learning (ICML)*, 2026.

[14] P. Dong, K.-H. Hung, T. Gao, D. Sadigh, and C. Finn. Expo-ft: Sample-efficient reinforcement learning finetuning for vision-language-action models. *arXiv preprint arXiv:2605.25477*, 2026.

[15] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[16] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, 2018.

[17] L. C. Baird. Residual algorithms: Reinforcement learning with function approximation. In *International Conference on Machine Learning*, 1995.

[18] J. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997. doi: 10.1109/9.580874.

[19] A. Kumar, R. Agarwal, T. Ma, A. Courville, G. Tucker, and S. Levine. DR3: Value-based deep reinforcement learning requires explicit regularization. In *International Conference on Learning Representations*, 2022.

[20] X. Chen, C. Wang, Z. Zhou, and K. W. Ross. Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations (ICLR)*, 2021.

[21] P. J. Ball, L. Smith, I. Kostrikov, and S. Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning (ICML)*, 2023.

[22] S. Fujimoto, P. D'Oro, A. Zhang, Y. Tian, and M. Rabbat. Towards general-purpose model-free reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2025.

[23] J. Park, M. Park, and D. Lee. Pretraining a shared q-network for data-efficient offline reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 38: 130527–130555, 2026.

[24] K. Black, N. Brown, D. Driess, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[25] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[26] A. Ren, J. Lidard, L. Ankile, A. Simeonov, P. Agrawal, A. Majumdar, B. Burchfiel, H. Dai, and M. Simchowitz. Diffusion policy policy optimization. In *International Conference on Learning Representations*, volume 2025, pages 77288–77329, 2025.

[27] K. Lei, H. Li, D. Yu, Z. Wei, L. Guo, Z. Jiang, Z. Wang, S. Liang, and H. Xu. Rl-100: Performant robotic manipulation with real-world reinforcement learning. *arXiv preprint arXiv:2510.14830*, 2025.

[28] S. Ding, K. Hu, S. Zhong, H. Luo, W. Zhang, J. Wang, J. Wang, and Y. Shi. GenPO: Generative diffusion models meet on-policy reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=BmRNz1TpCc.

[29] D. McAllister, S. Ge, B. Yi, C. M. Kim, E. Weber, H. Choi, H. Feng, and A. Kanazawa. Flow matching policy gradients. In *International Conference on Learning Representations*, volume 2026, pages 36352–36372, 2026.

[30] H. Ma, T. Chen, K. Wang, N. Li, and B. Dai. Efficient online reinforcement learning for diffusion policy. *arXiv preprint arXiv:2502.00361*, 2025.

[31] S. Pfrommer, Y. Huang, and S. Sojoudi. Reinforcement learning for flow-matching policies. *arXiv preprint arXiv:2507.15073*, 2025.

[32] S. Park, Q. Li, and S. Levine. Flow q-learning. In *International Conference on Machine Learning (ICML)*, 2025.

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

[34] 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 *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[35] X. Yuan, T. Mu, S. Tao, Y. Fang, M. Zhang, and H. Su. Policy decorator: Model-agnostic online refinement for large policy model. *arXiv preprint arXiv:2412.13630*, 2024.

[36] L. Ankile, Z. Jiang, R. Duan, G. Shi, P. Abbeel, and A. Nagabandi. Residual off-policy rl for finetuning behavior cloning policies. *arXiv preprint arXiv:2509.19301*, 2025.

[37] Y. Guo et al. ConRFT: A reinforced fine-tuning method for vla models via consistency policy. *arXiv preprint arXiv:2502.05450*, 2025.

[38] S. Tan, K. Dou, Y. Zhao, and P. Krähenbühl. Interactive post-training for vision-language-action models. *arXiv preprint arXiv:2505.17016*, 2025.

[39] H. Li, Y. Zuo, J. Yu, Y. Zhang, Z. Yang, K. Zhang, X. Zhu, Y. Zhang, et al. SimpleVLA-RL: Scaling vla training via reinforcement learning. *arXiv preprint arXiv:2509.09674*, 2025.

[40] G. Lu, W. Guo, C. Zhang, Y. Zhou, H. Jiang, Z. Gao, Y. Tang, and Z. Wang. Vla-rl: Towards masterful and general robotic manipulation with scalable reinforcement learning. *arXiv preprint arXiv:2505.18719*, 2025.

[41] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[42] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[43] 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.

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

[45] 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 (NeurIPS)*, 2023.

[46] C. Gelada, S. Kumar, J. Buckman, O. Nachum, and M. G. Bellemare. Deepmdp: Learning continuous latent space models for representation learning. In *International conference on machine learning*, pages 2170–2179. PMLR, 2019.

[47] M. Schwarzer, A. Anand, R. Goel, R. D. Hjelm, A. Courville, and P. Bachman. Data-efficient reinforcement learning with self-predictive representations. *arXiv preprint arXiv:2007.05929*, 2020.

[48] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Robotics: Science and Systems (RSS)*, 2023. arXiv:2303.04137.

[49] S. Fujimoto and S. S. Gu. A minimalist approach to offline reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. arXiv:2106.06860.

[50] Z. Zhou, A. Peng, Q. Li, S. Levine, and A. Kumar. Efficient online reinforcement learning fine-tuning need not retain offline data. In *The Thirteenth International Conference on Learning Representations*, 2025.

[51] J. Lyu, Z. Lin, S. Fujimoto, K. Yang, Y. Chen, S. Yang, Z. Lu, and D. Ye. Debiased model-based representations for sample-efficient continuous control. In *Forty-third International Conference on Machine Learning (ICML)*, 2026.

[52] D. Tarasov, V. Kurenkov, A. Nikulin, and S. Kolesnikov. Revisiting the minimalist approach to offline reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2305.09836.

[53] S. Park, H. Bharadhwaj, and S. Tulsiani. Demodiffusion: One-shot human imitation using pre-trained diffusion policy. In *International Conference on Robotics and Automation (ICRA)*, 2026.

[54] A. Tang, W. Chen, A. Wagenmaker, C. Finn, and S. Levine. Improving robotic generalist policies via flow reversal steering. 2026.

[55] M. M. Hong, J. Zhang, A. Nagabandi, and A. Gupta. Tmrl: Diffusion timestep-modulated pre-training enables exploration for efficient policy fine-tuning. In *Robotics: Science and Systems (RSS)*, 2026.

[56] D. Palenicek, F. Vogt, J. Watson, I. Posner, and J. Peters. XQC: Well-conditioned optimization accelerates deep reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026.

[57] D. Kim, Y. Lee, M. Park, K. Kim, I. M. A. Nahendra, T. Seno, S. Min, D. Palenicek, F. Vogt, D. Kragic, J. Peters, J. Choo, and H. Lee. Flashsac: Fast and stable off-policy reinforcement learning for high-dimensional robot control. *arXiv preprint arXiv:2604.04539*, 2026.

[58] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *Advances in neural information processing systems (NeurIPS)*, 2020.

[59] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, et al. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 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. In *Robotics: Science and Systems (RSS)*, 2018.

[61] M. Heo, Y. Lee, D. Lee, and J. J. Lim. FurnitureBench: Reproducible real-world benchmark for long-horizon complex manipulation. In *Robotics: Science and Systems*, 2023.

[62] T. Chen, Z. Chen, B. Chen, Z. Cai, Y. Liu, Q. Liang, Z. Li, X. Lin, Y. Ge, Z. Gu, W. Deng, Y. Wang, Y. Mu, W. Wang, Y. Cheng, Z. Li, and P. Luo. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

[63] Physical Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, M. Y. Galliker, D. Ghosh, L. Groom, K. Hausman, B. Ichter, S. Jakubczak, T. Jones, L. Ke, D. LeBlanc, S. Levine, A. Li-Bell, M. Mothukuri, S. Nair, K. Pertsch, A. Z. Ren, L. X. Shi, L. Smith, J. T. Springenberg, K. Stachowicz, J. Tanner, Q. Vuong, H. Walling, H. Wang, and U. Zhilinsky. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

[64] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4RL: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Appendix

Contents

A Training Algorithm **15**

B Experiment and Implementation Details **15**

B.1 Environment Details 15

B.2 **BOOST** Implementation Details 16

B.3 Generalist-Policy Experiments 18

B.4 Baseline Implementation Details 19

C Additional Results **20**

C.1 Full Results Tables 20

C.2 Real-Robot Task Sequence 20

A Training Algorithm

BOOST retains the off-policy residual RL loop: collect experience, update the critic and residual from mixed offline/online data, and update target networks. Algorithm 1 highlights only the additions: supervised pretraining, steered proposals, and one supervised objective per component. The base policy remains frozen throughout. As in Section 3.2, $p_\zeta(\cdot \mid s)$ denotes the proposal noise distribution, including the Gaussian blend and the EMA acting head.

Algorithm 1 **BOOST** on top of off-policy residual RL

Require: frozen base π^{base}, demonstrations $\mathcal{D}_{\text{offline}}$, residual RL learner, candidates K
initialize dynamics heads, steering head, and success buffer $\mathcal{B}^+ \leftarrow \emptyset$
cache demonstration noise targets by inverting the frozen base
pretrain on $\mathcal{D}_{\text{offline}}$: critic features with $\mathcal{L}_{\text{trans}}$, residual with ungated $\mathcal{L}_{\text{im}}$, steering head with $\mathcal{L}_{\text{zbc}}$; synchronize its EMA copy
while interaction budget remains **do**
    Act: sample K noises from $p_\zeta(\cdot \mid s)$; generate, correct, and select chunks (Eq. 6)
    store experience in online replay; add successful episodes to $\mathcal{B}^+$ and cache their inverted noise targets
    Critic: augment TD updates on $\mathcal{D}$ with $\mathcal{L}_{\text{trans}}$ (Eq. 3)
    Residual: augment policy updates with value-gated $\mathcal{L}_{\text{im}}$ on $\mathcal{D}_{\text{offline}} \cup \mathcal{B}^+$ (Eq. 5)
    Steering: once $\mathcal{B}^+$ is nonempty, update the steering head with $\mathcal{L}_{\text{zbc}}$ on $\mathcal{D}_{\text{offline}} \cup \mathcal{B}^+$ and update its EMA copy
    retain residual RL's target updates and gradually increase steering weight α
end while

Chunk-level TD targets. The one-chunk target in Section 3 extends to n chunks as

$$y = \sum_{j=0}^{n-1} \gamma^j R^{\text{chunk}}_{t+jH} + \gamma^n (1-d)\, \mathbb{E}_{z'}\left[\bar{Q}'(s_{t+nH}, \hat{a}(s_{t+nH}, z'))\right], \tag{8}$$

where $R^{\text{chunk}}_t = \sum_{\ell=0}^{H-1} r_{t+\ell}$, n is shortened at episode boundaries, and d suppresses terminal bootstrapping. Discounting is per chunk. The expectation averages four next-proposal values with $z' \sim p_\zeta(\cdot \mid s_{t+nH})$; the dynamics target remains one chunk ahead, regardless of n.

Implementation details. Critic pretraining updates the trunks and independent affine dynamics heads, not the scalar value heads. Inverted steering targets are cached, with online inversion warm-started at the executed latent. Steering uses episode-balanced online sampling and an EMA acting head. Exploration warm-up, update frequencies, loss normalization, residual constraints, and task-specific sampling and steering schedules are detailed in Appendix B.

B Experiment and Implementation Details

B.1 Environment Details

All methods on a task use the same code, data, and evaluation protocol, and start from the same frozen flow-matching base policy. We report the base policy's pass@1 as base success in the tables below. **BOOST** uses 4 critics, while DICE, DSRL, and EXPO-FT use 10. We leave equalizing the ensemble size to future work.

Robosuite. Robosuite [59] provides contact-rich tabletop tasks with image observations. We use `Square` (peg insertion onto a square nut), `ToolHang` (a long-horizon assembly that threads and hangs a tool on a frame), and two-arm `Transport` (a bimanual hand-off and placement), with 50 human demonstrations per task, a budget of 600k environment steps (1.4M for `ToolHang`), and evaluation over 50 episodes. All methods use the same pre-extracted image features. Table 1 lists the tasks.

Table 1: Robosuite tasks.

Task	Obs	Horizon	Cameras	Budget (env steps)	Base success
`Square`	image	400	2	600k	47%
`ToolHang`	image	700	2	1.4M	49%
`Transport` (two-arm)	image	700	4	600k	85%

Adroit. Adroit [60] provides dexterous manipulation with a 24-DoF anthropomorphic hand and state observations. We use `Door`, `Hammer`, `Pen`, and `Relocate`, with the human and expert trajectories distributed with the Minari ports of the D4RL datasets [64], a horizon of 200 steps, a 600k-step budget, and evaluation every 80k steps over 50 episodes. Table 2 lists the tasks.

Table 2: Adroit tasks.

Task	Obs dim	Action dim	Demos	Base success
`Door`	39	28	25 human	50%
`Hammer`	46	26	5 expert	76%
`Pen`	45	24	25 human	54%
`Relocate`	39	30	10 expert	74%

FurnitureBench. FurnitureBench [61] is a long-horizon assembly benchmark. We use its simulator, FurnitureSim, at the low-randomization level with image observations from a front and a wrist camera encoded by a frozen ResNet featurizer. Actions are 10-D absolute end-effector targets at 10 Hz executed in 8-step chunks, and an episode succeeds only when the furniture is fully assembled. All methods are trained with the same stage reward, which gives 0.2 for each of grasp, lift, align, and insert and 1 for assembly. The critic and residual act on normalized delta actions relative to the current end-effector pose, the critic receives object-relative privileged features, and the residual is additionally conditioned on the base chunk. We train for 1M environment steps and evaluate every 48k steps over 100 episodes. Table 3 lists the tasks.

Table 3: FurnitureBench tasks.

Task	Horizon	n-step return	Budget	Base success
`One Leg`	700	5	1M	31%
`Round Table`	700	5	1M	25%

B.2 BOOST Implementation Details

Shared hyperparameters for all methods on Robosuite are listed in Table 4, and FurnitureBench hyperparameters in Table 6.

Table 4: Shared per-task hyperparameters on Robosuite. All methods on a task use the same base policy, image features, budget, data mix, and evaluation protocol. Slash-separated entries are **BOOST** / DICE / DSRL / EXPO-FT.

Hyperparameter	`Square`	`ToolHang`	`Transport`
env steps	600k	1.4M	600k
episode horizon	400	700	700
cameras	2	2	4
parallel envs	8	8	4
action chunk	4	8	8
base flow steps	10	20	10
action candidates K	16	16	16
grad. steps / iter	20/20/20/20	10/10/10/20	10/10/10/20
critic ensemble	4/10/10/10	4/10/10/10	4/10/10/10
n-step return	3	5	3
batch size	256	256	256
discount γ	0.99	0.99	0.99
target rate κ	0.01	0.01	0.01
critic hidden	1024^3	1024^3	1024^3
lr (DICE family / EXPO-FT)	$10^{-4}/3{\cdot}10^{-4}$	$10^{-4}/3{\cdot}10^{-4}$	$10^{-4}/3{\cdot}10^{-4}$
expert demos used in online RL	50	50	50

BOOST hyperparameters. Table 5 lists the **BOOST**-specific hyperparameters used on Robosuite and Adroit.

Table 5: **BOOST** hyperparameters.

	Hyperparameter	Value
dynamics head	pre-training	10k steps
	auxiliary weight	1
imitation	weight λ	100
	pre-training	20k steps
	online successes per imitation batch	128 (once available)
steering head	demonstration pre-training	3000 steps
	online updates	1 per iteration once successes exist
	EMA rate κ_g	0.995
	trust weight α	$0 \to 1$ over 50k env steps after the first online update
	proposal	$z = \alpha\tilde{z} + (1-\alpha)\epsilon$, $\epsilon \sim \mathcal{N}(0, I)$

The steering head p_ζ is a conditional flow-matching MLP with three blocks of width 256. We integrate its velocity field v_ζ with 8 fixed Euler steps, independent of the base policy's solver. We train the head on the horizon-mean inverse latent and repeat each sample over the chunk before mixing. The inverse latent $z^\star = \mathrm{invert}_{\pi^{\mathrm{base}}}(s, a)$ is the preimage of a recorded action under the frozen base, $\pi^{\mathrm{base}}(s, z^\star) = a$. We compute it on the base policy's Euler grid with Anderson-accelerated fixed-point iteration,

$$z^{(k+1)} = \mathrm{Anderson}\big(z^{(k)};\, \pi^{\mathrm{base}}(s, \cdot),\, a\big), \quad k < 10, \qquad z^{(0)} = \begin{cases} z_{\mathrm{exec}} & (s,a) \in \mathcal{B}^+, \\ \epsilon \sim N(0, I) & (s,a) \in \mathcal{D}_{\mathrm{offline}}. \end{cases} \tag{9}$$

For online successes we initialize at the executed latent z_{exec}, which yields the preimage closest to the one actually executed. For demonstrations we initialize from the prior. We compute inverse latents once per transition and cache them.

We update the actor and the target networks every second iteration, performing all gradient steps of the iteration at once. The actor's value term is normalized by the detached mean absolute Q and applied only to online transitions, while the BC penalty on the residual norm uses both demonstrations and online data. We keep the soft-Q filter of DICE [13] on this penalty (Section 3.1). The penalty

on candidate k is dropped when the corrected chunk has higher value than the base proposal and the recorded action's value underestimates its Monte-Carlo return,

$$\mathcal{G}_k(s) = \mathbb{I}\big[\bar{Q}(s, \hat{a}_k) > \bar{Q}(s, \pi^{\text{base}}(s, z_k))\big]\, \mathbb{I}\big[\bar{Q}(s, a) - \widehat{G}(s) < \epsilon_Q\big], \tag{10}$$

so that the penalty becomes $\beta \sum_k (1 - \mathcal{G}_k) \|\pi^{\text{RL}}(s, z_k)\|_2^2$, where $\hat{a}_k = \hat{a}(s, z_k)$ is the corrected chunk of candidate k, $\widehat{G}(s)$ is the Monte-Carlo return of the recorded transition, and $\epsilon_Q = -0.5$ (-0.55 on FurnitureBench) after a short warm-up.

FurnitureBench hyperparameters. Table 6 lists the hyperparameters that differ from the **BOOST** values in Table 5 on the FurnitureBench tasks. All other hyperparameters follow the Robosuite values above. The `Round Table` column lists only the values that differ from `One Leg`; a dash marks an unchanged value.

Table 6: FurnitureBench hyperparameters.

	Hyperparameter	`One Leg`	`Round Table`
shared	parallel envs	8	16
	n-step return	5	–
	gradient steps per iteration	20	–
	base-only warm-up	48k steps	–
residual	initialization	zero	–
	BC weight β	500	–
imitation	imitation gate critic	target critic	–
	soft-Q filter threshold	-0.55 after 5k steps	–
steering head	updates per iteration	1	2
	batch size	256	–
	learning rate	$3{\cdot}10^{-4}$	–
	demo fraction	$0.5 \rightarrow 0.2$ over 100 successes	–
	trust weight α cap	1.0	0.8

B.3 Generalist-Policy Experiments

RoboTwin 2.0 with $\pi_{0.5}$. We adapt a frozen $\pi_{0.5}$ checkpoint [63], co-trained on RoboTwin 2.0 data, on the bimanual `Place Mouse Pad` task in RoboTwin 2.0 [62] with the ALOHA-AgileX embodiment. The base policy is served as a frozen model that maps an observation and input noise to a 50-step action chunk, of which we execute 25 steps before re-planning. **BOOST** and DICE therefore interact with $\pi_{0.5}$ only through its input noise and its outputs, and neither updates its weights. We use the Robosuite hyperparameters of Table 5 with a continuously updated steering head and an imitation weight of 1000. The reward is the task's binary success signal. We compare **BOOST** with DICE and DSRL, train for 200k environment steps, and evaluate every 50k steps over 50 episodes, reporting eval success averaged over three seeds with its standard deviation (Figure 6). Table 7 summarizes the setup.

Table 7: RoboTwin 2.0 hyperparameters.

Hyperparameter	Value
Base policy	frozen $\pi_{0.5}$, co-trained on RoboTwin 2.0, served
Action chunk / execution	50-step chunk, 25 steps executed
Reward	binary task success
Imitation weight λ	1000
Steering head updates	continuous, once successes exist
Training budget	200k environment steps
Evaluation protocol	every 50k steps, 50 episodes, 3 seeds

Real-world Flex-π. The real-robot experiment adapts Flex-π, run in action-only mode, on the bimanual YAM screwdriver task; see Section 4.4 and the all-camera rollout in Figure 8. Table 8 summarizes the setup.

Table 8: Real-world Flex-π hyperparameters (gripper self-repair, screwdriver phase).

	Hyperparameter	Value
setup	base policy / action mode	frozen Flex-π, action-only
	control rate / episode timeout	30 Hz / 1000 ticks ($\approx$ 33 s)
	training episodes / learner updates	104 (17 warm-up + 87 residual) / 1803
	evaluation protocol	20 episodes per checkpoint
shared	parallel envs	1
	n-step return	3
	gradient steps per robot chunk	5
	base-only warm-up	9k executed ticks ($\approx$ 17 episodes)
	critic ensemble / learning rate	10 (min) / 10^{-4}
	demo fraction in RL batches	$0.5 \rightarrow 0.25$ over 104 episodes
	action selection	best-of-4
residual	initialization	zero
	BC weight β	5000
	actor learning rate	$3{\cdot}10^{-5}$
imitation	demo BC on inverted latents	weight 100, ungated
	imitation gate critic	–
	soft-Q filter threshold	-0.85 after 3k env steps
steering head	updates per new success	20
	batch size	256
	learning rate	$3{\cdot}10^{-4}$
	demo fraction	$1.0 \rightarrow 0.15$ over 10k iterations
	trust weight α cap	1.0 (annealed over 105k env steps)

B.4 Baseline Implementation Details

All baselines share the base policy, demonstrations, budget, and evaluation of **BOOST**.

DICE. Candidates are $K{=}16$ draws from prior noise, each corrected by a randomly initialized residual and selected with a 10-critic ensemble trained by temporal-difference learning.

EXPO-FT. The edit policy is an MLP (256^3) with edit scale 0.1 and learning rate $3{\cdot}10^{-4}$. Each iteration scores 8 base plus 8 edited candidates with the critic, and the base flow itself is fine-tuned every iteration (flow-matching update, lr $3{\cdot}10^{-4}$).

DSRL (NA). An action critic Q_A is trained by temporal-difference learning on real actions, and a noise critic Q_W is distilled from $Q_A(s, \mathrm{decode}(w))$, using 10 heads with a subsample of 2. The actor is an MLP (2048^3, layer norm) that maximizes Q_W with no gradient through the frozen base. The latent bound is $b_W{=}1.5$, with one latent per step broadcast over the chunk, and the reward applies a -1 per-step bias with per-step discounting.

OGPO. We use the authors' image recipe [12]: the whole flow actor is fine-tuned online with a PPO-style objective over the denoising chain, treating each action chunk as one macro-action whose terminal reward is the mean over a 10-critic ensemble (Q aggregated by mean rather than minimum, target rate $\kappa{=}0.05$). Each update samples 32 noisy denoising trajectories from a slow EMA copy of the actor ($\kappa{=}0.005$), which also supplies the old-policy log-probabilities and the best-of-N candidates, and adds a success-buffer behavior-cloning term for stability. The runs use the same frozen image features and the same per-task budget as every other method, with 16 collection environments.

C Additional Results

C.1 Full Results Tables

Tables 9 and 10 list the final eval success of every method on every task behind Figures 3 and 4, averaged over seeds with standard deviations.

Table 9: **BOOST is the only method above 90% on all three Robosuite tasks.** Eval success (%) at the end of training, averaged over 3 seeds with standard deviation (50 evaluation episodes). Bold marks our method.

Method	Square	ToolHang	Transport
EXPO-FT	93 ± 1	70 ± 6	86 ± 7
DSRL (NA)	95 ± 2	0 ± 0	1 ± 1
OGPO	68 ± 3	37 ± 2	76 ± 4
DICE	85 ± 3	36 ± 27	98 ± 2
BOOST (Ours)	$\mathbf{97 \pm 3}$	$\mathbf{96 \pm 3}$	$\mathbf{97 \pm 3}$

Table 10: **BOOST is at or near the ceiling on Adroit and more than doubles the base policy's success on both FurnitureBench tasks.** Eval success (%), averaged over 3 seeds with standard deviation, using 50 evaluation episodes on Adroit and 100 on FurnitureBench. †collapsed to 0% within 192k steps.

	Adroit				FurnitureBench	
Method	Door	Hammer	Pen	Relocate	One Leg	Round Table
base pass@1	50	76	54	74	31	25
EXPO-FT	100 ± 0	100 ± 0	96 ± 2	96 ± 0	52 ± 5	12 ± 3
DSRL (NA)	0 ± 0	100 ± 0	100 ± 0	96 ± 0	$0 \pm 0^{\dagger}$	4
DICE	98 ± 2	100 ± 0	73 ± 11	92 ± 2	11 ± 11	0
BOOST (Ours)	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$	$\mathbf{94 \pm 6}$	$\mathbf{100 \pm 0}$	$\mathbf{85 \pm 3}$	**62**

C.2 Real-Robot Task Sequence

Figure 8 illustrates the bimanual screwdriver task used for Flex-π adaptation in Section 4.4. The synchronized camera views show the arm motion and close-up tool contact.

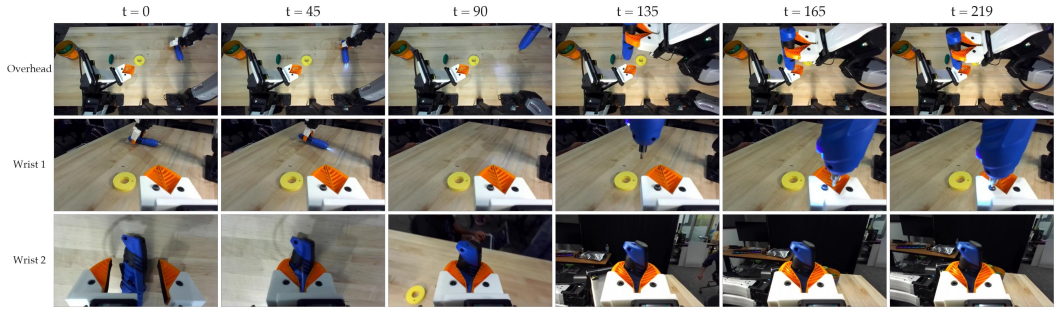


Figure 8: **Dexterous real-robot task rollout.** Rows show the overhead view and two wrist-camera views of the screwdriver approaching and engaging the gripper held by the other arm. This sequence comes from a real training rollout of **BOOST** with Flex-π as the base policy.