LEARNING QUALITY-CONTROLLABLE ROBOT POLICIES WITH ITERATIVE HINDSIGHT RELABELING

Anonymous authors
Paper under double-blind review

ABSTRACT

Robot policy evaluation often relies on a binary success signal for task completion. In real-world deployment, however, desired execution behaviors are also defined by quality attributes such as execution speed and motion smoothness, which need to be dynamically controlled according to task requirements. To address this, we introduce Quality-Controllable Policy (QCP), a robot policy that enables inference-time control over both individual quality attributes and their combinations. To extend the controllable quality range beyond initial demonstrations, we introduce an iterative hindsight relabeling loop: QCP collects online rollouts under target quality labels, adds successful trajectories to the dataset, relabels the accumulated trajectories in hindsight according to their measured quality, and finetunes the policy. Experiments on Robomimic and RoboEval demonstrate that QCP consistently improves both task execution quality and success rates across iterations, producing successful rollouts that exceed the quality of the initial demonstrations. Finally, we validate QCP on real-world manipulation tasks, showing that controlling speed increases throughput in conveyor operations and controlling transport stability improves success when carrying wine glasses, showing that quality-controllable policies yield practical gains in physical deployment.

1 INTRODUCTION

Robot foundation models (RFMs) have shown strong generalization across tasks, environments, and objects (Black et al., 2025; Bjorck et al., 2025; Ye et al., 2026). These policies are usually trained with behavior cloning (BC) on successful demonstrations, which treats every successful trajectory as equally desirable regardless of how the task was executed. As a result, the learned policy reproduces the mixture of execution behaviors in the data, whether fast or slow, smooth or jerky, and offers no way to steer toward the behavior a deployment actually requires. In practice, however, the desired behavior of a robot is not defined by task success alone (Wang et al., 2025; Liu et al., 2026; Mao et al., 2026) and different tasks call for different *quality attributes*. For example, a robot deployed on a conveyor line should prioritize execution speed, whereas a robot carrying wine glasses on a tray must prioritize transport stability.

To address this, prior work has proposed quality-controllable policies whose execution behavior can be adjusted along a chosen quality axis at inference time (Zhan et al., 2020; Yang et al., 2025; Petitbois et al., 2026). However, such policies are trained on static demonstration datasets, which typically cover only a bounded quality range and lack the high-performance regimes needed in deployment (e.g., maximum speed or ideal smoothness). Our goal is therefore not only to learn a quality-controllable policy, but also to continuously expand its controllable range beyond the initial demonstrations through online data collection and policy updates.

In this paper, we introduce *Quality-Controllable Policy (QCP)*, which addresses this limitation of BC by making execution quality an explicit conditioning input (Figure 1). For each demonstration, we measure the target quality attributes with rule-based score functions and quantize the scores into discrete quality labels. The policy is then trained to generate actions conditioned on both the observation history and these labels. Rather than averaging over execution behaviors, QCP learns task completion from all successful trajectories while distinguishing how each one was executed. At inference time, the user specifies a target label to steer execution behavior, for example, requesting the highest speed label for conveyor operations.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

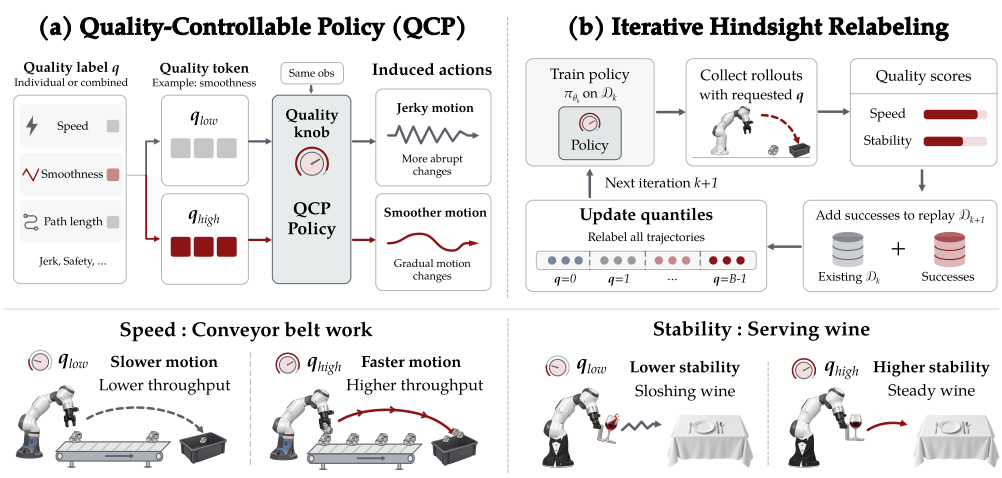

Figure 1: **Overview of QCP and iterative hindsight relabeling.** *Left:* QCP controls execution quality at inference time through conditioning on quantized quality tokens. *Right:* Iterative hindsight relabeling shifts the quality distribution of successful rollouts toward higher quality, expanding the controllable range beyond the initial demonstrations.

To extend the controllable quality range beyond the demonstrations, we pair QCP with *iterative hindsight relabeling*. At each iteration, we collect rollouts conditioned on the highest quality label, keep the successful ones, relabel them by the quality they actually achieved, and add them to the dataset. We then recompute the quality labels of all trajectories and retrain the policy. As higher-quality rollouts accumulate, the highest label comes to represent progressively higher quality, allowing the policy to reach operating regimes absent from the initial demonstrations. Because quality is measured by rule-based score functions, this loop requires no manual quality annotation.

Experiments on Robomimic (Mandlekar et al., 2022) and RoboEval (Wang et al., 2025) show that QCP with iterative hindsight relabeling steadily improves both target quality and task success across iterations, outperforming quality-filtered BC (Emmons et al., 2022) and RL baselines (Mao et al., 2024; Petitbois et al., 2026; Wagenmaker et al., 2025; Dong et al., 2026). Iterative relabeling pushes the quality distribution well beyond the range of the initial demonstrations, reaching an IQR-normalized Wasserstein distance above 1.5. Beyond single-attribute control, QCP reliably controls multiple attributes jointly, improving six quality attributes simultaneously in RoboEval. Finally, in real-world experiments, controlling execution speed substantially increases throughput in conveyor operations, and controlling transport stability improves success when carrying wine glasses on a tray.

2 Related Work

Policy learning with quality control Decision Transformer and RvS condition action prediction on desired returns or goals to improve task performance (Chen et al., 2021; Emmons et al., 2022). Beyond task performance, prior work also studies how to learn different ways of executing a task, representing behavioral styles through learned trajectory representations or explicit labels (Petitbois et al., 2025; Zhan et al., 2020; Yang et al., 2025). Learning a requested style, however, can conflict with task performance. SORL addresses this tension by combining trajectory clustering with task advantage weighting (Mao et al., 2024), while SCIQL combines task and style advantages to account for both task performance and style alignment (Petitbois et al., 2026). QCP combines quantized quality conditioning with iterative hindsight relabeling. Our experiments show simultaneous gains in execution quality and task success, demonstrating that this tradeoff can be overcome.

Online improvement of behavioral quality Freeform Preference Learning (FPL) learns a reward model for the desired quality axes from human comparisons of robot trajectories and trains policies conditioned on the predicted rewards (Torne et al., 2026). This supports online improvement through repeated rollout collection and policy training, but requires additional human comparisons at each round. In text generation, Quark repeatedly samples under the highest reward token, rela-

bels accumulated samples by reward quantiles, and retrains the language model conditioned on the assigned labels (Lu et al., 2022). QCP applies a similar loop to robot execution quality, using automatically computed quality labels and accumulated successful trajectories to extend the controllable range beyond the initial demonstrations without additional human annotation.

3 Main Method

In real-world deployment, users often care about quality metrics beyond binary task success, such as execution speed or motion smoothness. To account for these, we propose Quality-Controllable Policy (QCP), which adjusts its execution behavior along user-specified quality axes by conditioning on quantized quality levels (Section 3.1). Because demonstration data typically covers only a limited range of quality, we extend QCP's controllable quality range through iterative hindsight relabeling, alternating between collecting online rollouts, relabeling trajectories by their achieved quality, and updating the policy (Section 3.2). Figure 1 illustrates the overall framework.

3.1 Quality-Controllable Policy

Quality scores and quantization. Let $\mathcal{D}$ be a dataset of successful demonstrations, where each trajectory $\tau = (o_0, a_0, \ldots, o_{T-1}, a_{T-1}, o_T)$ consists of observations o_t and executed actions a_t, with T the number of executed actions. We assume access to a rule-based score function $m(\tau) \in \mathbb{R}$ that measures a quality axis of interest. The simplest example is $m(\tau) = \frac{1}{T}$ as a proxy for execution speed, and our experiments additionally consider more complex scores such as normalized action differences for smoothness and total joint path length for efficiency (Appendix A for detailed definitions).

We then quantize the scores into B equal-frequency bins and use the bin index as a discrete quality label:

$$q(\tau) = \left\lceil B \cdot \hat{F}(m(\tau)) \right\rceil - 1 \in \{0, \ldots, B-1\}, \qquad \hat{F}(s) = \frac{1}{|\mathcal{D}|} \sum_{\tau' \in \mathcal{D}} \mathbf{1}\{m(\tau') \le s\}, \tag{1}$$

where $\hat{F}$ is the empirical CDF of scores in $\mathcal{D}$, so that $q(\tau) = j$ if and only if $m(\tau)$ lies between the j/B and $(j+1)/B$ empirical quantiles. Every trajectory in $\mathcal{D}$ is thus paired with its label $q(\tau)$, and we train a quality-conditioned policy on this labeled dataset.[1]

Quality-conditioned policy training. We train a policy π_θ that, given a history of L observations $h_t = (o_{t-L+1}, \ldots, o_t)$ and a quality label q, predicts a chunk of H future actions $A_t = (a_t, \ldots, a_{t+H-1})$. We construct training samples (h_t, A_t, q) by slicing each trajectory $\tau \in \mathcal{D}$ at every time step t and attaching its label $q = q(\tau)$. Denoting the resulting set of training samples by $\mathcal{D}_{\text{tr}}$, the policy is trained with the following quality-conditioned behavior cloning objective:

$$\mathcal{L}(\theta) = -\mathbb{E}_{(h_t, A_t, q) \sim \mathcal{D}_{\text{tr}}} \left[\log \pi_\theta(A_t \mid h_t, q) \right]. \tag{2}$$

We model π_θ as a flow-matching policy (Lipman et al., 2023), which optimizes equation 2 through a regression objective. Given noise $\epsilon \sim \mathcal{N}(0, I)$ and flow time $u \sim \mathcal{U}[0, 1]$, we form the interpolated action chunk $A_t^u = (1-u)\epsilon + uA_t$ and train a velocity field v_θ to predict the target $A_t - \epsilon$:

$$\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{(h_t, A_t, q) \sim \mathcal{D}_{\text{tr}},\, u,\, \epsilon} \left[\left\| v_\theta\big(A_t^u, u, h_t, q\big) - (A_t - \epsilon) \right\|^2 \right], \tag{3}$$

where the norm is taken over all action dimensions and time steps in the chunk. We consider two cases for conditioning on quality q. For simple policies trained from scratch, an embedding layer maps q to a learned vector $\mathbf{e}_q \in \mathbb{R}^d$, which is concatenated with the encoded observation history to condition the velocity network.[2] For vision-language-action (VLA) models such as $\pi_{0.5}$ (Black et al., 2025), we instead insert the label into the language prompt as text (e.g., "quality level: 3"), similar to $\pi_{0.7}$ (Ai et al., 2026), and fine-tune the model with its original training recipe. This enables the model to interpret quality levels through language conditioning without architectural changes.

[1] See Section 5.2 for a comparison with continuous score conditioning, and Appendix G for a comparison with assigning a separate quality label to shorter segments of each trajectory.

[2] For joint control of multiple quality attributes, we use policies trained from scratch. We compute a separate quality label for each attribute and map each label through a separate embedding layer. The embeddings jointly condition the velocity network.

Inference during deployment. At inference time, the quality label becomes a control knob: the user specifies a target bin $q^\star \in \{0, \ldots, B-1\}$, and actions are generated by integrating the learned velocity field $v_\theta(\cdot, u, h_t, q^\star)$ from $u = 0$ to $u = 1$ starting from Gaussian noise. We find that QCP responds reliably to this input, shifting its behavior toward the requested bin and attaining a quality score within the corresponding range (Figure 3). Unless noted, we condition on the highest bin $q^\star = B - 1$ throughout our evaluation and in the iterative training procedure of Section 3.2.

3.2 Iterative Hindsight Relabeling

A policy trained only on demonstrations is largely bounded by the quality present in $\mathcal{D}$. To push beyond this range, we introduce a simple iterative pipeline that alternates between rollout collection, hindsight relabeling, and policy updates. The key observation is that, owing to stochasticity in action generation, rollouts conditioned on the highest bin often exceed the best demonstrations. By folding these rollouts into the dataset and relabeling all trajectories, the policy's controllable quality range expands over iterations.

Algorithm 1 QCP with Iterative Hindsight Relabeling

Require: Demonstrations $\mathcal{D}$, score m, B bins, K collection rounds.
1: Set $\mathcal{D}_0 \leftarrow \mathcal{D}$, $q^\star \leftarrow B - 1$.
2: Label $\mathcal{D}_0$ and train π_{θ_0} as in Section 3.1.
3: **for** $k = 0, \ldots, K-1$ **do**
4: Collect rollouts with $\pi_{\theta_k}(\cdot \mid h_t, q^\star)$.
5: Compute $m(\tau)$ for each rollout.
6: Append successful rollouts to $\mathcal{D}_k$ to obtain $\mathcal{D}_{k+1}$.
7: Compute $\hat{F}$ from scores in $\mathcal{D}_{k+1}$.
8: Recompute $q(\tau)$ for all $\tau \in \mathcal{D}_{k+1}$ using Equation 1.
9: Train $\pi_{\theta_{k+1}}$ from scratch on $\mathcal{D}_{k+1}$ using Equation 3.
10: **end for**

We initialize the dataset as $\mathcal{D}_0 = \mathcal{D}$ and train an initial policy π_{θ_0} on it as in Section 3.1. At each iteration k, we execute π_{θ_k} conditioned on the highest bin $q^\star = B - 1$ to collect a batch of rollouts, and keep only those that succeed at the task.[3] Although every rollout is requested at the highest bin, the realized scores vary. We therefore relabel each successful rollout in hindsight by its realized score $m(\tau)$ rather than the requested label, and append it to $\mathcal{D}_k$ to obtain $\mathcal{D}_{k+1}$. We then recompute the quality labels $q(\tau)$ for all trajectories in $\mathcal{D}_{k+1}$, including the original demonstrations, by applying equation 1 with the empirical CDF taken over $\mathcal{D}_{k+1}$. As higher-scoring rollouts accumulate, the bin boundaries shift upward, progressively raising the quality represented by the highest bin while the original demonstrations preserve the low-quality range and thus widen the controllable range.

Finally, we train $\pi_{\theta_{k+1}}$ on the relabeled $\mathcal{D}_{k+1}$ as in Section 3.1. Rather than fine-tuning π_{θ_k}, we reinitialize the policy at every iteration, from random weights for policies trained from scratch and from the pretrained checkpoint for VLAs. This is because relabeling changes which quality range each label refers to, so the label-to-behavior mapping learned in the previous iteration no longer matches the new targets and would need to be unlearned.[4] We repeat rollout collection, relabeling, and retraining for a fixed number of iterations, as summarized in Algorithm 1.

4 Experiments

We investigate the performance of QCP with iterative hindsight relabeling by focusing on the following four research questions:

(1) Does iterative hindsight relabeling consistently improve the target quality of QCP while maintaining task success? (Section 4.2)

(2) Does iterative hindsight relabeling expand the controllable quality range of QCP? (Section 4.3)

(3) Can QCP jointly control and improve multiple quality attributes? (Section 4.4)

(4) Does our approach remain effective on real-world robot tasks? (Section 4.5)

[3] We also tried including failed rollouts as additional low-quality data, but found that this degraded performance.

[4] Reinitialization may also help avoid the loss of plasticity and primacy bias observed when networks are trained continually on a growing dataset in deep RL (Nikishin et al., 2022), although we do not study this effect directly.

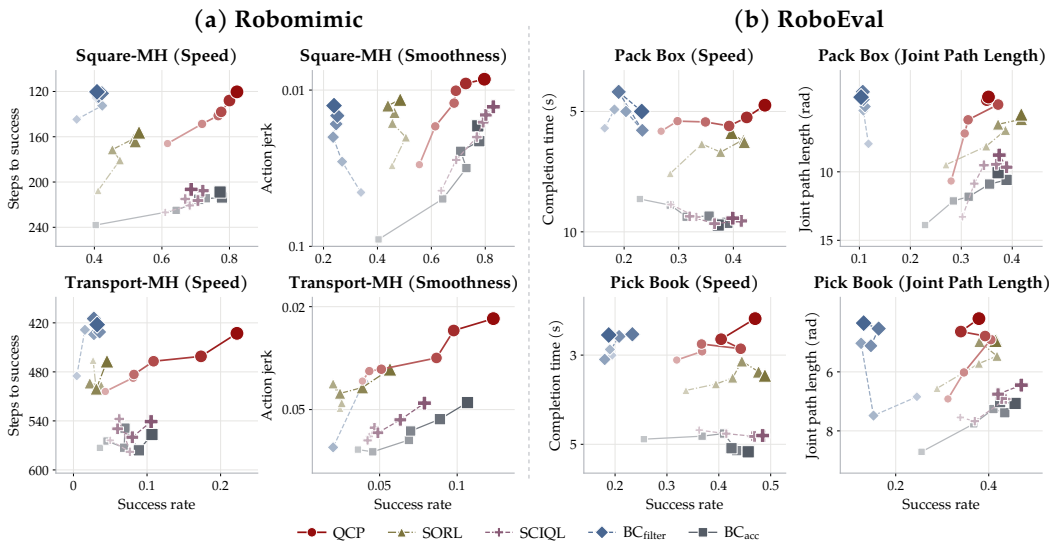

Figure 2: Comparison of QCP with BC baselines ($\text{BC}_{\text{filter}}$, BC_{acc}) and behavioral quality learning methods (SORL, SCIQL) across iterations on (a) Robomimic and (b) RoboEval. Each plot shows task success rate against a quality metric, with the y-axis oriented so that higher is better. Darker and larger markers correspond to later iterations. QCP improves quality and task success together more effectively than the baselines. Results are averaged over three random seeds.

4.1 Experimental setup

Tasks and datasets. Our main simulation experiments use Square and Transport tasks from Robomimic (Mandlekar et al., 2022) and Pack Box and Pick Book tasks from RoboEval (Wang et al., 2025). For Robomimic, we use the Multi-Human (MH) datasets, whose demonstrations are collected by operators of varying proficiency and thus span a wide range of execution quality. For RoboEval, we use the demonstration datasets with randomized object positions and orientations. For the real-world experiments in Section 4.5, we use a Franka Research 3 arm with the DROID hardware setup (Khazatsky et al., 2024), and collect 400 teleoperated demonstrations for the conveyor task and 50 for the wine-glass transport task to form each task's initial dataset $\mathcal{D}_0$.

Baselines. We compare QCP against three groups of baselines. To test whether data accumulation or quality filtering alone can achieve similar gains, we consider BC trained on accumulated successful rollouts (BC_{acc}), and quality-filtered BC using the best 10% of successful trajectories along the target axis ($\text{BC}_{\text{filter}}$). To compare against existing methods for learning desired behavioral styles, we evaluate SORL (Mao et al., 2024) and SCIQL (Petitbois et al., 2026), which learn style-conditioned policies through offline RL. Finally, to compare quality conditioning against direct optimization of quality rewards, we evaluate DSRL (Wagenmaker et al., 2025) and EXPO (Dong et al., 2026), which improve flow matching policies with online RL. For both, we add a weighted quality reward to the task reward and match the total number of environment interactions used by QCP across all iterations. Reward settings and interaction budgets are detailed in Appendix C.

Quality metrics. On Robomimic, we target execution speed by steps to success and action smoothness by action jerk. On RoboEval, we target execution speed by completion time and motion efficiency by total joint path length. We negate or invert qualities which lower values indicate higher quality to obtain the score $m(\tau)$ used in equation 1. In real-world experiments, we target execution speed, measured by end-effector speed, and transport stability, measured by the end-effector tilt from vertical. Detailed definitions are provided in Appendix A.

Training and evaluation. In simulation, we use a simple flow-matching policy with an MLP backbone, trained for 100k steps on Robomimic and 30k steps on RoboEval with a batch size of 1024. In real-world experiments, we fine-tune the $\pi_{0.5}$ vision-language-action (VLA) model (Black et al., 2025) for 10 epochs with a batch size of 32. These settings are fixed across iterations. We use

$B = 4$ quality bins and always condition QCP on the highest bin $q^\star = 3$ during rollout collection.[5] At each iteration, we collect 300 rollouts per task and seed in simulation, and in the real world, as many rollouts as there are demonstrations in the initial dataset $\mathcal{D}_0$. These rollouts serve both as evaluation and as data for the next iteration. We report mean quality over successful rollouts and success rate over all rollouts, both averaged across three seeded runs of the full iterative procedure.

4.2 Iterative improvement over quality and success rate

Figure 2 shows execution quality and task success across iterations. Across tasks and quality attributes, QCP improves both target quality and task success, whereas the baselines tend to improve one at the expense of the other, each for a different reason. BC_{acc} has no quality signal and improves task success alone. $\text{BC}_{\text{filter}}$ and SORL, which train only on trajectories whose label matches the request, discard most of the successful data, at a cost in task success. SCIQL keeps all trajectories but weights them by the sum of a success advantage and a quality advantage, which creates a trade-off between the two (Appendix D). QCP avoids these problems: all successful trajectories are used by conditioning on their quality labels, so even low-quality trajectories contribute to learning task completion, while the requested label selects high-quality behavior at inference.

QCP also achieves higher final quality than the flow matching RL baselines DSRL and EXPO (Table 1), even when we assign high weights to their quality rewards.[6] These results suggest that the evaluated RL methods for flow matching policies, which focus primarily on improving task success rates, do not effectively translate additional quality rewards into improved execution quality. QCP instead provides the quality signal in a supervised manner by conditioning the policy on achieved quality labels, achieving high execution quality together with strong task success.

Table 1: **QCP versus RL baselines.** Results at the final iteration (iteration 6) are averaged across three random seeds. Higher success rates and lower quality metric values are better.

	Sq+Speed		**Sq+Smoothness**		**Tr+Speed**		**Tr+Smoothness**		**Pack+Speed**		**Pack+JPL**		**Book+Speed**		**Book+JPL**	
Method	SR (%)	Speed (steps)	SR (%)	Smoothness ($\times 10^{-3}$)	SR (%)	Speed (steps)	SR (%)	Smoothness ($\times 10^{-3}$)	SR (%)	Speed (s)	SR (%)	JPL (rad)	SR (%)	Speed (s)	SR (%)	JPL (rad)
QCP (ours)	82.2	**120.3**	79.7	**8.5**	**22.2**	**432.8**	**12.3**	**22.2**	45.8	**4.82**	35.3	**6.42**	47.0	**2.43**	37.9	**4.63**
SORL	53.2	156.0	48.4	11.5	4.6	466.8	5.7	34.9	39.7	5.64	41.8	7.11	**48.9**	3.37	41.4	5.15
SCIQL	68.7	206.5	**83.0**	12.8	10.6	540.8	7.9	47.2	39.9	9.24	37.4	9.06	48.4	4.75	**47.0**	6.40
DSRL	37.0	208.8	43.0	46.9	4.0	549.5	7.0	79.4	**60.7**	6.35	**50.7**	66.05	0.0	—	0.0	—
EXPO	**96.0**	162.5	79.0	157.4	11.0	567.7	3.0	176.5	39.0	8.72	43.7	24.43	3.0	5.73	4.3	29.20

4.3 Quality control and range expansion

Quality control across iterations. We first test whether QCP can produce different levels of execution quality on request by evaluating it at each bin in $\{0, 1, 2, 3\}$ at every iteration (Figure 3). Even the initial policy trained only on demonstrations (iteration 1) follows the requested bins: on Square, for example, requesting the highest bin yields faster execution than the lowest, with mean steps to success of 193.7 at $q = 3$ versus 308.1 at $q = 0$, and the intermediate bins fall in between. We further find that this ordering holds for every attribute at every iteration, even though new data comes only from the highest bin. Iterative relabeling thus raises the quality of the highest bin over iterations while preserving the ability to select different execution behaviors from a single policy.

Quality beyond the initial demonstrations. Figure 4 further shows that successful rollout distributions shift *beyond the initial demonstration range* for Square smoothness and RoboEval joint path length. We quantify the distribution shift using the Wasserstein-1 distance between the initial demonstrations and final successful rollouts, normalized by the interquartile range (IQR) of the initial demonstration scores. This distance exceeds 1.5 for smoothness and joint path length, indicating large shifts from the initial demonstration distribution. For speed, the rollout distribution shifts toward shorter completion times, extending beyond the range observed in the initial demonstrations.[7] These results show that online rollout collection and iterative learning enable QCP to expand its capable quality range beyond the initial demonstrations while successfully completing the task.

[5] Section 5.2 evaluates performance with different numbers of quality bins.

[6] Appendix D compares quality reward weights for DSRL and EXPO.

[7] In our observations, the fastest rollouts appear to be close to the maximum physically achievable speed in this setup.

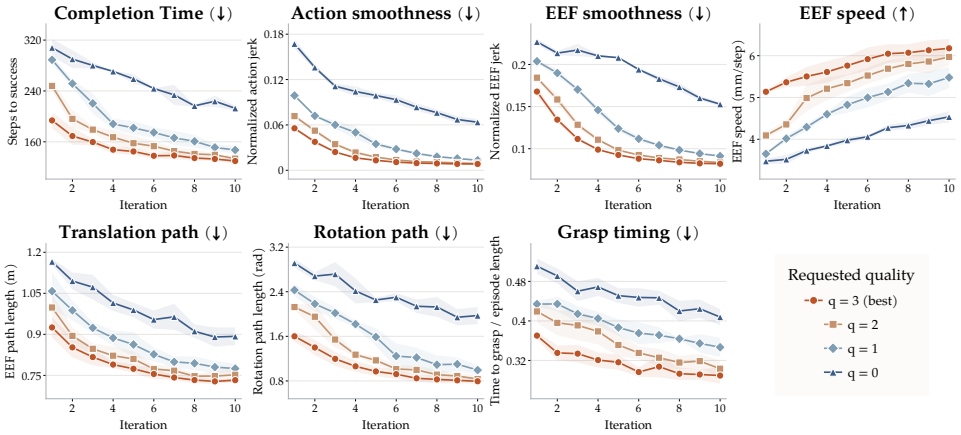

Figure 3: Quality achieved at each requested bin across iterations on Robomimic Square. The ordering of the bins is preserved while the whole range moves toward higher quality (arrows). Mean ± one standard deviation over three seeds.

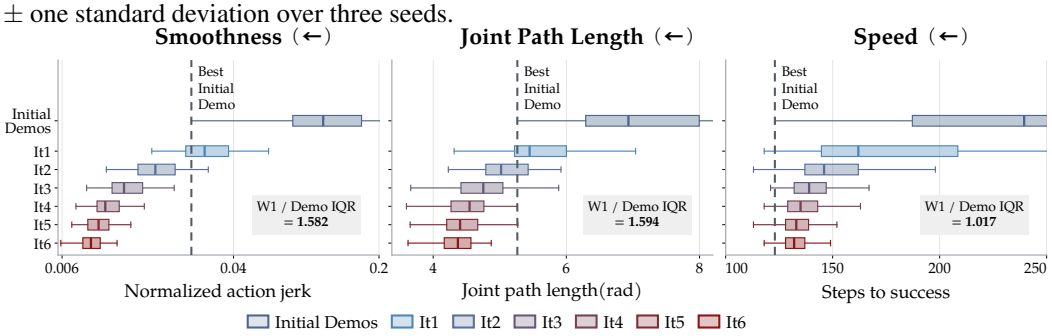

Figure 4: **Quality distributions across iterations.** Starting from the initial demonstration distribution, QCP shifts the quality distribution of successful rollouts toward higher quality, extending beyond the initial demonstration range. Arrows point toward higher quality.

4.4 Multi-attribute quality control and improvement

We next evaluate whether QCP can control multiple quality attributes in the requested directions and improve them through iterative learning.

Bidirectional control of two quality attributes. On Robomimic Square, we train a separate QCP policy for each of the four combinations of faster or slower execution and smaller or larger lift amplitude.[8] Figure 5 shows that both attributes change in the requested directions over iterations. Across all four combinations, the final policies achieve quality levels close to those of policies trained for each attribute alone, showing that QCP can pursue both objectives together with comparable quality.

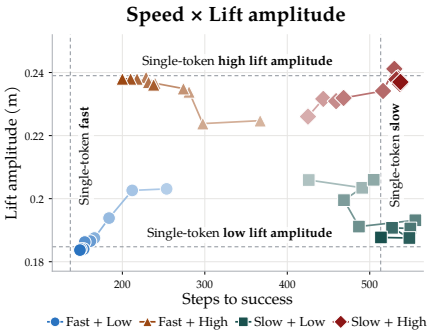

Figure 5: QCP jointly adjusts execution speed and lift amplitude in the requested directions over iterations.

Joint improvement across six attributes. We further evaluate whether QCP can jointly improve up to six quality attributes across four RoboEval tasks (Figure 6). We condition each policy on six quality labels and request the best bin for every attribute during iterative learning (see Appendix E for experimental details). The final policies achieve better quality than initial BC and BC_{acc} for most quality attributes across the evaluated tasks. These results suggest that QCP can jointly improve multiple aspects of execution even when a single policy is conditioned on up to six quality attributes.

[8] Lift amplitude is the difference between the maximum and minimum end-effector height during execution.

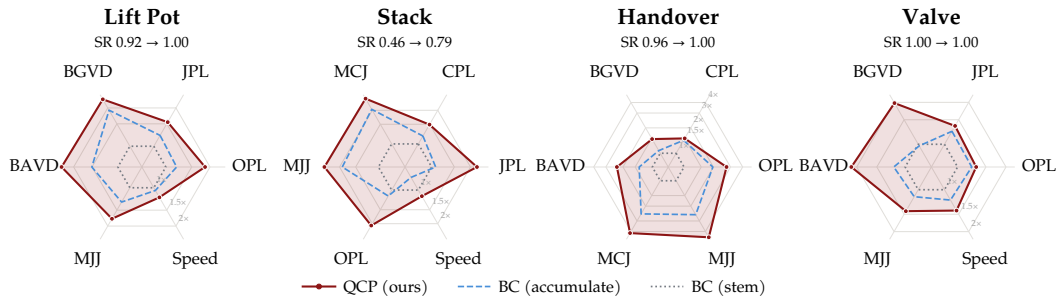

Figure 6: **Joint improvement of six quality attributes on RoboEval.** Final QCP and BC_{acc} policies are compared with initial BC on four tasks. Each axis shows the initial BC score divided by the method's score, so the initial BC polygon lies on the $1\times$ ring and outward indicates better quality. Success rates of initial BC and final QCP are shown above each task.

4.5 Quality control for real robot tasks

We evaluate whether controlling execution quality leads to practical gains in real-world robot tasks. We consider two tasks with distinct quality requirements: execution speed for the conveyor task, measured by end-effector speed, and transport stability for the wine-glass task, measured by the tray's tilt angle from horizontal (Appendix A).

Serving wine. The robot must carry a tray holding wine glasses from one table to another without dropping any glass. This task is challenging even for human operators: only 50 of 362 teleoperated attempts (13.8%) succeeded during data collection, as keeping the tray level while moving proved difficult. Conditioning QCP on the highest stability bin reduces the mean tilt angle from 0.0620 rad for BC to 0.0498 rad, a 19.7% reduction, while the success rate improves from 24% to 34% over iterations (Figure 7a). In contrast, $\text{BC}_{\text{filter}}$ improves tilt angle and success rate through iteration 2 but collapses at iteration 3, suggesting that retaining lower quality data provides broader coverage needed to sustain task success. Notably, even the initial QCP policy, trained only on demonstrations, already achieves a higher success rate than BC, and the final policy succeeds far more often than teleoperation did during data collection.

Conveyor belt work. Twenty cans are placed on a conveyor belt moving at a fixed speed, one every four seconds, and throughput is defined as the number of cans successfully grasped per minute. Achieving high throughput under this fixed schedule requires both fast execution and a high success rate. Relative to BC, QCP increases mean end-effector speed from 186.6 to 229.5 mm/s and throughput from 7.1 to 8.7 cans per minute, improving both by 23% (Figure 7b). $\text{BC}_{\text{filter}}$ achieves higher end-effector speed than QCP but substantially lower throughput, indicating a trade-off between execution quality and task success. Notably, even the initial QCP policy already exceeds the throughput of BC, while the final QCP policy reaches a speed of 229.5 mm/s, surpassing the 150.7 mm/s of human teleoperation, where high-speed control is challenging.

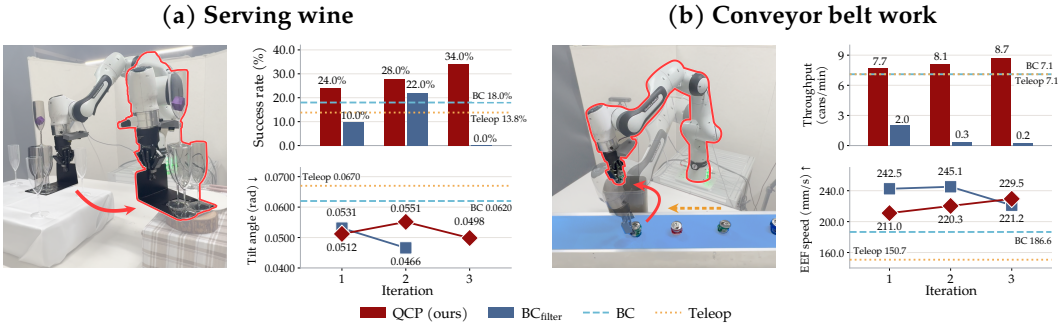

Figure 7: **Quality control on real robots.** QCP achieves higher task success than BC baselines while (a) reducing the tray tilt angle in wine serving and (b) increasing end-effector speed in conveyor work.

5 Analysis

5.1 Effect of quality conditioning on rollout distributions

We examine whether the distribution shift in Figure 4 comes from the iterative hindsight relabeling loop or from quality conditioning alone, since a conditioned policy might reach rare behaviors simply by sampling more. To separate the two, we spend the same rollout budget in two ways under the best quality bin $q^\star = 3$: 1,800 rollouts from the QCP policy frozen at iteration 1, or 300 rollouts at each of iterations 1 to 6 with retraining in between (Figure 8). The fixed policy reproduces its iteration-1 distribution regardless of sample count, and none of its successful rollouts reaches the median smoothness of the final iteration. The improvement thus comes from policy updates on the relabeled data, which move the best bin to a quality level the previous policy could not produce.

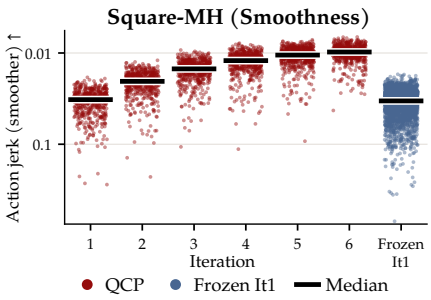

Figure 8: Iterative versus fixed policy rollouts at equal sample counts.

5.2 Continuous quality scores versus quantized quality labels

We examine why QCP uses quantized quality labels rather than continuous scores. With four quantile bins, roughly the best quarter of the training trajectories share the best label, giving the policy multiple examples of the requested behavior. Learning from multiple trajectories with the same label can provide a more stable training signal than conditioning on a score represented by only a few examples. The importance of training support is also evident in return conditioning, where Decision Transformer (Chen et al., 2021) and RvS (Emmons et al., 2022) can be unreliable for requests outside the training distribution (Brandfonbrener et al., 2022). On Square, quantized quality labels achieve faster execution with higher task success than a fixed continuous target at every iteration, and increasing the number of bins beyond four buys a small gain in speed at a growing cost in task success (Figure 9). These results support simple quantization as an effective representation for quality control. Experimental details are provided in Appendix F.

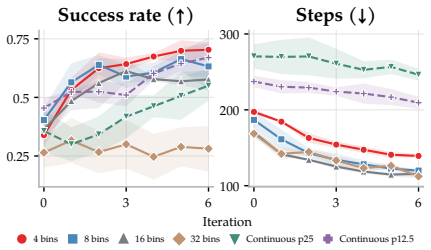

Figure 9: Quantized labels with 4 to 32 bins versus continuous score conditioning on Square speed.

6 Conclusion

We presented QCP, a robot policy that uses quantized quality labels to control individual execution attributes and their combinations at inference time. Iterative hindsight relabeling extends the controllable quality range beyond the initial demonstrations through online rollout collection and policy updates, without additional human annotation. Experiments on Robomimic and RoboEval show that QCP can improve execution quality and task success while retaining quality control across iterations. Real robot experiments further demonstrate that QCP can be directed to pursue the execution quality desired for a given task.

Reproducibility Statement

Appendix A defines every quality score with its unit, Appendix B lists the architecture, optimizer, and per-experiment settings, Appendix C specifies each baseline and its reward and interaction budget, Appendices E and F gives the settings of the multi-attribute and analysis experiments and Appendix G gives the settings for subtrajectory-based QCP ablation studies. Simulation experiments use the public Robomimic and RoboEval benchmarks and datasets. Code, configuration files, and the labeled datasets will be released upon publication.

Ethics Statement

This work uses public simulation benchmarks and an in-laboratory robot arm; no human subjects or personal data are involved. Quality-controllable policies raise no risks beyond those of any learned manipulation policy, and the speed attribute in particular should be deployed with the same safety limits as any fast robot motion.

LLM Usage Statement

Large language models were used to assist with editing the text and with writing the scripts that produce the figures and tables. All experiments, analyses, and claims were designed, run, and verified by the authors.

References

Bo Ai, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Greg Balke, Kevin Black, George Bokinsky, Shihao Cao, Thomas Charbonnier, Vedant Choudhary, Foster Collins, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Maitrayee Dhaka, Jared DiCarlo, Danny Driess, Michael Equi, Adnan Esmail, Yunhao Fang, Chelsea Finn, Catherine Glossop, Thomas Godden, Ivan Goryachev, Lachlan Groom, Haroun Habeeb, Hunter Hancock, Karol Hausman, Gashon Hussein, Victor Hwang, Brian Ichter, Connor Jacobsen, Szymon Jakubczak, Rowan Jen, Tim Jones, Gregg Kammerer, Ben Katz, Liyiming Ke, Mairbek Khadikov, Chandra Kuchi, Marinda Lamb, Devin LeBlanc, Brendon LeCount, Sergey Levine, Xinyu Li, Adrian Li-Bell, Vladislav Lialin, Zhonglin Liang, Wallace Lim, Yao Lu, Enyu Luo, Vishnu Mano, Nandan Marwaha, Aikys Mongush, Liam Murphy, Suraj Nair, Tyler Patterson, Karl Pertsch, Allen Z. Ren, Gavin Schelske, Charvi Sharma, Baifeng Shi, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Will Stoeckle, Jiaming Tang, Jimmy Tanner, Shalom Tekeste, Marcel Torne, Kyle Vedder, Quan Vuong, Anna Walling, Haohuan Wang, Jason Wang, XuDong Wang, Chris Whalen, Samuel Whitmore, Blake Williams, Charles Xu, Sukwon Yoo, Lili Yu, Wuming Zhang, Zhuoyang Zhang, and Ury Zhilinsky. $\pi_{0.7}$: A steerable generalist robotic foundation model with emergent capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

Johan Bjorck, Valts Blukis, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Xiaowei Jiang, Kaushil Kundalia, Jan Kautz, Zhiqi Li, Kevin Lin, Zongyu Lin, Loic Magne, Yunze Man, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Jing Wang, Qi Wang, Shihao Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Seonghyeon Ye, Zhiding Yu, Yizhou Zhao, Zhe Zhang, Ruijie Zheng, and Yuke Zhu. GR00T N1.5: An improved open foundation model for generalist humanoid robots. NVIDIA Research Blog, 2025. URL `https://research.nvidia.com/labs/gear/gr00t-n1_5/`.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: A vision-language-action model with open-world generalization. In *Conference on Robot Learning*, 2025.

David Brandfonbrener, Alberto Bietti, Jacob Buckman, Romain Laroche, and Joan Bruna. When does return-conditioned supervised learning work for offline reinforcement learning? In *Advances in Neural Information Processing Systems*, 2022.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems*, 2021.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Robotics: Science and Systems*, 2023.

Perry Dong, Qiyang Li, Dorsa Sadigh, and Chelsea Finn. EXPO: Stable reinforcement learning with expressive policies. In *International Conference on Learning Representations*, 2026.

Scott Emmons, Benjamin Eysenbach, Ilya Kostrikov, and Sergey Levine. RvS: What is essential for offline RL via supervised learning? In *International Conference on Learning Representations*, 2022.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, Peter David Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Yecheng Jason Ma, Patrick Tree Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, Huy Ha, Arhan Jain, Abraham Lee, Youngwoon Lee, Marius Memmel, Sungjae Park, Ilija Radosavovic, Kaiyuan Wang, Albert Zhan, Kevin Black, Cheng Chi, Kyle Beltran Hatch, Shan Lin, Jingpei Lu, Jean Mercat, Abdul Rehman, Pannag R Sanketi, Archit Sharma, Cody Simpson, Quan Vuong, Homer Rich Walke, Blake Wulfe, Ted Xiao, Jonathan Heewon Yang, Arefeh Yavary, Tony Z. Zhao, Christopher Agia, Rohan Baijal, Mateo Guaman Castro, Daphne Chen, Qiuyu Chen, Trinity Chung, Jaimyn Drake, Ethan Paul Foster, Jensen Gao, Vitor Guizilini, David Antonio Herrera, Minho Heo, Kyle Hsu, Jiaheng Hu, Muhammad Zubair Irshad, Donovon Jackson, Charlotte Le, Yunshuang Li, Kevin Lin, Roy Lin, Zehan Ma, Abhiram Maddukuri, Suvir Mirchandani, Daniel Morton, Tony Nguyen, Abigail O'Neill, Rosario Scalise, Derick Seale, Victor Son, Stephen Tian, Emi Tran, Andrew E. Wang, Yilin Wu, Annie Xie, Jingyun Yang, Patrick Yin, Yunchu Zhang, Osbert Bastani, Glen Berseth, Jeannette Bohg, Ken Goldberg, Abhinav Gupta, Abhishek Gupta, Dinesh Jayaraman, Joseph J Lim, Jitendra Malik, Roberto Martín-Martín, Subramanian Ramamoorthy, Dorsa Sadigh, Shuran Song, Jiajun Wu, Michael C. Yip, Yuke Zhu, Thomas Kollar, Sergey Levine, and Chelsea Finn. DROID: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023.

Mengyuan Liu, Juyi Sheng, Peiming Li, Ziyi Wang, Tianming Xu, Tiantian Xu, and Hong Liu. Eval-Actions: Fine-grained execution quality evaluation for robotic manipulation. *arXiv preprint arXiv:2601.18723*, 2026.

Ximing Lu, Sean Welleck, Jack Hessel, Liwei Jiang, Lianhui Qin, Peter West, Prithviraj Ammanabrolu, and Yejin Choi. Quark: Controllable text generation with reinforced unlearning. In *Advances in Neural Information Processing Systems*, 2022.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *Conference on Robot Learning*, 2022.

Yanbo Mao, Jianlong Fu, Ruoxuan Zhang, Hongxia Xie, and Meibao Yao. Beyond success: Refining elegant robot manipulation from mixed-quality data via just-in-time intervention. In *Conference on Computer Vision and Pattern Recognition*, 2026.

Yihuan Mao, Chengjie Wu, Xi Chen, Hao Hu, Ji Jiang, Tianze Zhou, Tangjie Lv, Changjie Fan, Zhipeng Hu, Yi Wu, Yujing Hu, and Chongjie Zhang. Stylized offline reinforcement learning: Extracting diverse high-quality behaviors from heterogeneous datasets. In *International Conference on Learning Representations*, 2024.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International Conference on Machine Learning*, 2022.

Mathieu Petitbois, Rémy Portelas, Sylvain Lamprier, and Ludovic Denoyer. Offline learning of controllable diverse behaviors. *arXiv preprint arXiv:2504.18160*, 2025.

Mathieu Petitbois, Rémy Portelas, and Sylvain Lamprier. Offline reinforcement learning of high-quality behaviors under robust style alignment. In *International Conference on Machine Learning*, 2026.

Marcel Torne, Anubha Mahajan, Abhijnya Bhat, and Chelsea Finn. Freeform preference learning for robotic manipulation. *arXiv preprint arXiv:2606.32027*, 2026.

Andrew Wagenmaker, Mitsuhiko Nakamoto, Yunchu Zhang, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning. In *Conference on Robot Learning*, 2025.

Yi Ru Wang, Carter Ung, Christopher Tan, Grant Tannert, Jiafei Duan, Josephine Li, Anh Le, Rishabh Oswal, Markus Grotz, Wilbert Pumacay, Yuquan Deng, Ranjay Krishna, Dieter Fox, and Siddhartha Srinivasa. RoboEval: Where robotic manipulation meets structured and scalable evaluation. *arXiv preprint arXiv:2507.00435*, 2025.

Hanlin Yang, Jian Yao, Weiming Liu, Qing Wang, Hanmin Qin, Hansheng Kong, Kirk Tang, Jiechao Xiong, Chao Yu, Kai Li, Junliang Xing, Hongwu Chen, Juchao Zhuo, Qiang Fu, Wei Yang, and Haobo Fu. Diverse policies recovering via pointwise mutual information weighted imitation learning. In *International Conference on Learning Representations*, 2025.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, Ayaan Malik, Kyungmin Lee, William Liang, Nadun Ranawaka, Jiasheng Gu, Yinzhen Xu, Guanzhi Wang, Fengyuan Hu, Avnish Narayan, Johan Bjorck, Jing Wang, Gwanghyun Kim, Dantong Niu, Ruijie Zheng, Yuqi Xie, Jimmy Wu, Qi Wang, Ryan Julian, Danfei Xu, Yilun Du, Yevgen Chebotar, Scott Reed, Jan Kautz, Yuke Zhu, Linxi Fan, and Joel Jang. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

Eric Zhan, Albert Tseng, Yisong Yue, Adith Swaminathan, and Matthew Hausknecht. Learning calibratable policies using programmatic style-consistency. In *International Conference on Machine Learning*, 2020.

A Quality Score Functions

Every quality attribute is measured on one complete trajectory, from the initial observation to the step at which the task is completed. Both simulators run at a control rate of 20 Hz, so one step corresponds to 0.05 s. Either direction of an attribute can be the preferred one depending on the task; for each attribute we state below the direction preferred in our experiments. The score $m(\tau)$ of Section 3.1 is the measurement itself when a higher value is preferred and its reciprocal when a lower value is preferred, as in $m(\tau) = 1/T$ for speed, so that the highest bin $q^\star = B - 1$ always denotes the preferred end of the attribute; the figures and tables report the raw measurements.

Quantization. For every score, the bin edges are the $1/B, \dots, (B-1)/B$ quantiles of the scores of the successful trajectories in the current dataset, recomputed at every iteration. A trajectory whose score coincides with an edge is assigned to the better bin. For the joint conditioning experiments, each attribute is quantized independently.

Robomimic. Let L be the number of executed actions, $a_t \in \mathbb{R}^{6|\mathcal{A}|}$ the arm part of the action at step t (the position and rotation deltas of every arm in $\mathcal{A}$, gripper dimensions excluded), $p_t^r \in \mathbb{R}^3$ the end-effector position of arm r, and q_t^r its orientation quaternion. Let $s_t = \frac{1}{|\mathcal{A}|}\sum_r \|p_{t+1}^r - p_t^r\|_2$ denote the arm-averaged displacement per step. Square uses one arm and Transport two.

- **Speed (steps to success):** $T = L$, in steps. Lower is preferred in the main experiments; Section 4.4 also requests slower execution.
- **Action smoothness (normalized action jerk):** $\frac{1}{L-1}\sum_t \|a_{t+1} - a_t\|_2^2 \,/\, \frac{1}{L}\sum_t \|a_t\|_2^2$, dimensionless. Lower is better.
- **End-effector smoothness (normalized EEF jerk):** $\frac{1}{|\mathcal{A}|(L-2)}\sum_{r,t}\|p_{t+2}^r - 2p_{t+1}^r + p_t^r\|_2 \,/\, \frac{1}{L-1}\sum_t s_t$, dimensionless. Lower is better.
- **End-effector speed:** $\frac{1}{L-1}\sum_t s_t$, in m per step. Higher is better.
- **Translation path length:** $\frac{1}{|\mathcal{A}|}\sum_r \sum_t \|p_{t+1}^r - p_t^r\|_2$, in m. Lower is better.
- **Rotation path length:** $\sum_t \frac{1}{|\mathcal{A}|}\sum_r 2\arccos|\langle q_{t+1}^r, q_t^r\rangle|$, in rad. Lower is better.
- **Grasp timing:** the index of the first step with a closing gripper command divided by L, averaged over arms. Lower is better.
- **Lift amplitude:** $\frac{1}{|\mathcal{A}|}\sum_r \big(\max_t z_t^r - \min_t z_t^r\big)$ with z_t^r the height of the end-effector, in m. Section 4.4 requests both a smaller and a larger lift amplitude.

RoboEval. The following scores use the metric implementation of RoboEval (Wang et al., 2025). Let p_t^r be the wrist position of arm r, q_t^r its orientation quaternion, and $\theta_t^r \in \mathbb{R}^7$ its joint configuration.

- **Completion time (CT, "Speed"):** the number of control steps until success divided by 20 Hz, in simulated seconds. Lower is better. RoboEval reports wall-clock time; we use simulated time so that the score does not depend on the machine.
- **Joint path length (JPL):** $\sum_r \sum_t \|\theta_{t+1}^r - \theta_t^r\|_2$, in rad, summed over both arms. Lower is better.
- **Cartesian path length (CPL):** $\sum_r \sum_t \|p_{t+1}^r - p_t^r\|_2$, in m, summed over both arms. Lower is better.
- **Orientation path length (OPL):** $\sum_r \sum_t 2\arccos|\langle q_{t+1}^r, q_t^r\rangle|$, in rad, summed over both arms. Lower is better.
- **Mean joint jerk (MJJ):** the mean over arms and steps of $\|\dddot{\theta}_t^r\|_2$, with velocity, acceleration, and jerk obtained by successive finite differences with Δt, in rad/s^3. Lower is better.
- **Mean Cartesian jerk (MCJ):** the same construction on p_t^r, in m/s^3. Lower is better.
- **Bimanual arm velocity difference (BAVD):** $\frac{1}{T}\sum_t \big|\, \|\dot{\theta}_t^L\|_2 - \|\dot{\theta}_t^R\|_2 \big|$ using the simulator joint velocities, in rad/s. Lower is better.
- **Bimanual gripper vertical difference (BGVD):** $\frac{1}{T}\sum_t |z_t^L - z_t^R|$ with z the wrist height, in m. Lower is better.

Real robot. Let $\mathbf{p}_t$ be the end-effector position at timestep t, $\mathbf{z}_t^{\text{eef}}$ the unit z-axis of the end-effector frame, $\mathbf{z}^{\text{world}}$ the unit z-axis of the world frame, and T the episode length.

- **End-effector speed (conveyor work):**

$$\bar{v} = \frac{1}{T}\sum_{t=1}^{T} \|\dot{\mathbf{p}}_t\|_2 \,, \tag{4}$$

  in mm/s, computed over successful grasp segments. Higher is better.

- **Tilt angle (wine serving):**

$$\bar{\theta} = \frac{1}{T}\sum_{t=1}^{T} \left|\arccos\left(\mathbf{z}_t^{\text{eef}} \cdot \mathbf{z}^{\text{world}}\right)\right| , \tag{5}$$

  in rad, computed over successful episodes. Since the tray is horizontal when it is rigidly held by an end-effector aligned with $\mathbf{z}^{\text{world}}$ (Figure 7a), $\bar{\theta}$ directly reflects the uprightness of the wine glasses. Lower is better.

B Training and Evaluation Details

Policy and optimization. All simulation policies share one architecture and optimizer. The velocity network is an MLP with three layers of width 512 (expansion factor 1, no dropout) and a 128-dimensional flow-time embedding; a two-layer MLP encodes the observation history of $L = 2$ steps. The quality label is mapped to a learned embedding and concatenated with the encoded observation before the velocity network. Policies are trained from scratch at every iteration with AdamW (learning rate 10^{-4}, weight decay 10^{-5}, gradient-norm clipping at 10), a cosine learning-rate schedule over the training steps, batch size 1024, and an exponential moving average of the weights with rate 0.995. Actions are sampled with nine Euler steps of the learned velocity field. Conditioned on an observation history of $L = 2$ steps, the policy predicts a chunk of H actions that starts at the first observation of the history; the $L - 1$ actions aligned with past observations are discarded, the next $H - L$ actions are executed, and the last one is dropped before replanning. Both choices follow the original implementation of Diffusion Policy (Chi et al., 2023). The quality bins are recomputed as adaptive quartiles of the successful trajectories in the current dataset at every iteration, and rollouts are collected with the best bin requested. Table 2 lists the settings that differ between experiments.

Table 2: **Per-experiment settings.** Steps are gradient steps per iteration. "Policies" counts trained policies including the one trained on the initial dataset. The action chunk is written as chunk length / executed steps.

	Main results (Figs. 2, 4, Tab. 1)			Fig. 3	Fig. 5	Fig. 6	Fig. 8	Fig. 9
	Square	Transport	RoboEval	Square	Square	RoboEval	Square	Square
Steps per iteration	100k	100k	30k	30k	30k	30k	30k	30k
Episode limit (steps)	400	700	400	400	800	400	800	400
Action chunk	6 / 4	10 / 8	10 / 8	10 / 8	10 / 8	10 / 8	10 / 8	10 / 8

Reported statistics. Success rates are computed over all rollouts of an evaluation batch. Quality is computed as the mean over the successful rollouts of the batch, first within each seed and then averaged across seeds with equal weight.

Real robot experiments. We use a Franka Research 3 arm with the DROID hardware and camera setup (Khazatsky et al., 2024). The policy is $\pi_{0.5}$ (Black et al., 2025) fine-tuned with its original recipe for 10 epochs with batch size 32 at every iteration, starting from the pretrained checkpoint each time; the quality label is inserted into the language prompt as text. For wine serving, the initial dataset contains the 50 successful episodes among 362 teleoperated attempts; each iteration collects 50 rollouts, and success requires that the tray reaches the target table with no glass fallen. For conveyor work, the initial dataset contains 400 successful grasp segments from 720 teleoperated segments; each iteration collects 20 rollouts of 20 arriving cans, that is 400 segments, with cans arriving at one can every four seconds on a belt of fixed speed. Throughput is the success fraction

multiplied by the arrival rate of 20 cans per minute. Speed is measured on successful segments and tilt on successful episodes. The quality label requested at collection is the best bin, $\mathrm{BC}_{\text{filter}}$ retrains on the best 20% of successful trajectories along the target axis at every iteration, and BC denotes $\pi_{0.5}$ fine-tuned on the initial dataset without a quality label.

C Baseline Details

All baselines use the network, optimizer, training steps, rollout budget, retraining from scratch, success filter, and seeds of QCP (Appendix B); they differ only in how the training set is formed and in the learning objective.

BC_{acc}. BC_{acc} runs the loop of Algorithm 1 without a quality label: rollouts are collected by the unconditioned policy, successful ones are appended to the dataset, and a plain behavior-cloning policy is retrained on the accumulated set.

$\mathrm{BC}_{\text{filter}}$. $\mathrm{BC}_{\text{filter}}$ additionally ranks the successful trajectories of the accumulated set (demonstrations included) by the target score at every iteration and keeps the best 10%. At every iteration the filter is applied again to the whole accumulated set, not to the previously filtered set, so the filtered training set grows with the accumulated data rather than shrinking over iterations; the setting does not force the dataset to become smaller. On Robomimic at least 20 trajectories are kept, and on RoboEval the filter is implemented as a sampling weight of 0 or 1 without a floor. In the real robot experiments the filter keeps the best 20%, because the initial datasets are small (50 episodes for wine serving) and a 10% filter would leave too few trajectories to train on.

SORL and SCIQL Both methods proceed in two stages at every iteration. In the first stage, a value function is learned with implicit Q-learning (Kostrikov et al., 2022) on a chunk-level MDP whose state is the two-step observation history, whose action is the executed action chunk, and whose discount is 0.99 per decision; the task reward is 1 on the chunk in which the task succeeds and 0 otherwise. The value and twin Q networks are two-layer MLPs of width 256 with layer normalization, trained for 60k steps with Adam (learning rate 3×10^{-4}, cosine schedule, batch 1024), expectile 0.7, and Polyak averaging with rate 0.005. Advantages are divided by the 99th percentile of their magnitude, and the resulting weights $\exp(\beta A)$ are clipped at 100.

In the second stage, the actor is trained. It is our quality-conditioned flow policy with the same network and the same number of gradient steps as QCP (100k on Robomimic, 30k on RoboEval), and the weights from the first stage are used as sampling probabilities over the training samples. The two methods differ in how the weights are formed. SORL (Mao et al., 2024) trains on the samples whose label equals the requested one, weighted by $\exp(\beta A^r)$ with A^r the task advantage. SCIQL (Petitbois et al., 2026) relabels every trajectory under each of the four labels, learns a style advantage A^λ from the reward $\mathbf{1}\{z = q(\tau)\}$, and weights samples by $\exp(\beta\,\xi)$ with $\xi = A^\lambda + \sigma(A^\lambda)A^r$, as in the original paper: the style advantage is the primary term, and the task advantage counts only for actions that already match the requested style. Both stages are retrained from scratch at every iteration on the accumulated success-filtered set, and rollouts are collected with the best label requested. The main results use $\beta = 3$; Appendix D reports the other values of β and explains this choice.

DSRL and EXPO Both methods start from a frozen flow-matching BC policy trained on the demonstrations for 100k steps with the network of Appendix B and sampled with nine Euler steps. DSRL (Wagenmaker et al., 2025) learns a soft actor-critic policy over the initial noise of that policy (update-to-data ratio 20, discount 0.999, batch 256, learning rate 3×10^{-4}, twin critics of three layers of width 2048 with layer normalization). EXPO (Dong et al., 2026) learns an edit policy and a critic ensemble on top of the frozen policy (eight candidate actions per decision, edit scale 0.1, update-to-data ratio 5, discount 0.99, batch 256, an ensemble of ten critics with the minimum over two), with half of every batch drawn from the frozen policy's own 300 initial rollouts. Both act on chunks of 4 actions on Square and 8 elsewhere.

The reward has two parts, a task reward and a quality reward:

$$r_t = \mathbf{1}\{\text{task succeeds at step } t\} + \lambda\, r_t^{\mathrm{q}}. \tag{6}$$

The task reward is 1 at the step where the task succeeds, and the episode ends there. The quality reward r_t^{q} is a small penalty given at every step, and λ sets how much it counts relative to the task reward.

For speed, the penalty is the same at every step, $r_t^{\mathrm{q}} = -1/L_{\mathrm{ref}}$, where L_{ref} is the average length of the demonstrations. Summed over an episode, the quality reward equals minus the episode length divided by L_{ref}, so a shorter episode earns a higher return and an episode as long as a typical demonstration loses about 1.

For smoothness and joint path length, the penalty at step t depends on how much the motion at that step exceeds what is typical in the demonstrations:

$$r_t^{\mathrm{q}} = -\frac{1}{L_{\mathrm{ref}}}\Big(\mathrm{clip}\big(\tfrac{j_t - J_{\mathrm{ref}}}{J_{\mathrm{ref}}},\, -1,\, 5\big) - \bar{c}\Big), \tag{7}$$

where j_t is the per-step motion measure ($\|a_t - a_{t-1}\|^2$ over the arm dimensions on Robomimic, the joint-space increment on RoboEval), J_{ref} is its average over the demonstrations, the clipping keeps a single very jerky step from dominating the return, and $\bar{c}$ is the average of the clipped term over the demonstrations, so that a demonstration-like step receives zero penalty on average. Dividing by L_{ref} puts the return on the same scale as the speed reward.

All constants are computed once from the successful demonstrations of each task and then kept fixed: $L_{\mathrm{ref}} = 240$, $J_{\mathrm{ref}} = 0.0354$, $\bar{c} = -0.253$ on Square; $L_{\mathrm{ref}} = 630$, $J_{\mathrm{ref}} = 0.0669$, $\bar{c} = -0.145$ on Transport; $L_{\mathrm{ref}} = 136$, $J_{\mathrm{ref}} = 0.141$ on Pack Box; $L_{\mathrm{ref}} = 71$, $J_{\mathrm{ref}} = 0.137$ on Pick Book. DSRL multiplies the whole reward by $L_{\mathrm{ref}}/\max(1, \lambda)$ so that the size of its returns stays similar across values of λ. Table 1 uses $\lambda = 4$; Appendix D reports the other values.

The policy is evaluated on 100 episodes every 120k environment steps, which is the number of steps of 300 rollouts at the 400-step limit, and the paper reports the sixth evaluation, that is 720k environment steps after the prefill.

D Effect of Quality Reward Weight

The RL baselines have one coefficient that sets how strongly the quality signal enters training: the advantage temperature β of SORL and SCIQL and the quality reward weight λ of DSRL and EXPO. To choose the values used in the main results, we swept these coefficients on Robomimic Square for both attributes with the settings of Appendix C, three seeds per setting, at the same endpoint as Table 1 (Table 3). For each method we selected the setting that gives a good balance of success rate and quality, $\beta = 3$ and $\lambda = 4$, and used it for the main results on all tasks.

Table 3: **Coefficient sweep on Robomimic Square.** Each cell is success rate (%) / quality of successful episodes (steps to success for Speed, action jerk for Smoothness; lower is better), mean ± standard deviation over three seeds. Left: SORL and SCIQL over the advantage temperature β. Right: DSRL and EXPO over the quality reward weight λ. Bold rows are the settings used in the main results ($\beta = 3$, $\lambda = 4$) and reproduce Table 1.

	SORL		SCIQL	
β	Speed	Smoothness	Speed	Smoothness
0	34.0±3.5 / 213.8±17.0	36.7±5.3 / 0.0118±0.0010	77.1±9.0 / 251.2±25.6	72.3±6.0 / 0.0205±0.0017
1	39.0±6.0 / 206.5±18.8	40.3±8.3 / 0.0114±0.0012	78.0±8.7 / 231.0±26.7	79.3±6.2 / 0.0187±0.0018
2	45.0±9.5 / 189.4±9.8	51.3±3.7 / 0.0126±0.0007	74.7±4.3 / 218.9±18.6	79.7±9.8 / 0.0176±0.0018
3	**53.2±7.8 / 156.0±18.1**	**48.4±6.5 / 0.0115±0.0010**	**68.7±5.2 / 206.5±23.2**	**83.0±6.0 / 0.0128±0.0007**
4	41.7±5.7 / 188.1±18.0	43.3±4.9 / 0.0142±0.0008	70.7±8.4 / 212.5±14.6	85.0±4.7 / 0.0185±0.0022
10	48.7±5.5 / 194.2±23.3	44.7±3.7 / 0.0212±0.0017	63.7±7.7 / 208.2±19.0	59.2±6.0 / 0.0163±0.0012
16	47.3±7.4 / 159.5±18.0	44.7±4.6 / 0.0201±0.0015	52.3±5.1 / 195.3±13.8	61.0±9.4 / 0.0177±0.0014

	DSRL		EXPO	
λ	Speed	Smoothness	Speed	Smoothness
0	30.0±8.2 / 196.2±15.6	42.0±5.0 / 0.0463±0.0055	92.0±8.6 / 161.8±16.4	96.0±7.6 / 0.1183±0.0116
1	25.3±8.0 / 192.2±17.4	34.0±4.8 / 0.0455±0.0039	90.0±6.6 / 164.2±13.8	94.0±5.9 / 0.1428±0.0159
4	**37.0±4.2 / 208.8±22.4**	**43.0±8.1 / 0.0469±0.0044**	**96.0±4.1 / 162.5±10.7**	**79.0±10.0 / 0.1574±0.0147**
16	52.0±6.9 / 224.4±20.1	49.0±3.4 / 0.0510±0.0034	97.0±5.4 / 163.8±14.6	68.0±7.4 / 0.1631±0.0176

E Multi-Quality Conditioning Experiment Details

Two-attribute control on Square. Figure 5 uses Square with an 800-step episode limit so that slow executions can complete, ten policies, and three seeds. Each trajectory receives two labels, a speed quartile (steps to success) and a lift-amplitude quartile (Appendix A), both recomputed on the current dataset at every iteration. Each label is mapped through its own embedding and the two embeddings are concatenated with the encoded observation. A separate lineage is trained for each of the four corner commands, namely the lowest or highest speed quartile combined with the lowest or highest lift-amplitude quartile, and every lineage collects its rollouts under its own command. The dashed reference lines in Figure 5 are the extreme iteration means of single-label lineages run with the identical protocol: fastest and slowest speed commands, and lowest and highest lift-amplitude commands. The joint commands reach the two single-label references on both coordinates within a few percent, at a lower success rate for the slow commands.

Six-attribute improvement on RoboEval. Figure 6 uses the state-based policy of Table 2 on the Lift Pot, Stack Two Blocks, Handover, and Valve tasks of RoboEval with the demonstrations of the fixed-position variant, seven policies, 300 rollouts per iteration, and three seeds. Each trajectory receives six quartile labels, one per attribute, each mapped through its own embedding; all six labels are requested at their best bin during collection. The six attributes are chosen per task from the eight RoboEval metrics of Appendix A by dropping the two that a preliminary eight-attribute run showed to be unaffected by conditioning or to be in conflict with the task (Table 4). The reference policies are the unconditioned BC policy trained on the demonstrations (initial BC) and the unconditioned loop that accumulates its successful rollouts for six iterations (BC_{acc}), both with the same trainer, demonstrations, and seeds. Per-seed values are means over successful rollouts of the evaluation batch and are averaged across seeds. Table 5 lists the values behind the radar plot.

F Analysis Experiment Details

Rollout collection comparison (Section 5.1). The comparison uses the Square smoothness attribute with 30k training steps per iteration, an 800-step episode limit, and three seeds. The frozen arm draws 1,800 rollouts per seed from the checkpoint trained at iteration 1, conditioned on the best bin $q^\star = 3$, with the dataset normalization that checkpoint was trained under, and without any further training. The iterative arm is the loop itself, which draws 300 rollouts per seed at each of iterations 1 to 6 under the same best-bin request, so both arms spend 1,800 rollouts per seed after iteration 1. The purpose of this comparison is to separate the effect of the quality-conditioned archi-

Table 4: **Conditioned attributes per task.** The two dropped attributes and the reason.

Task	Conditioned	Dropped (reason)
Lift Pot	JPL, OPL, CT, MJJ, BAVD, BGVD	CPL (a fixed vertical lift leaves little Cartesian path to vary), MCJ (redundant with MJJ)
Stack	CPL, JPL, CT, OPL, MJJ, MCJ	BAVD (one-arm task, equal arm velocities are not a quality), BGVD (large by design of the task)
Handover	CPL, OPL, MJJ, MCJ, BAVD, BGVD	CT (in conflict with the smoothness attributes), JPL (redundant with CPL and OPL)
Valve	JPL, OPL, CT, MJJ, BAVD, BGVD	MCJ (nearly constant for a wrist rotation), CPL (path fixed by the valve geometry)

Table 5: **Six-attribute quality on RoboEval.** Numerical values of Figure 6: success rate and the six conditioned attributes at iteration 6 for QCP and BC_{acc}, and for the initial BC. Mean ± standard deviation over seeds; lower is better for every attribute; units as in Appendix A.

Lift Pot	SR	JPL	OPL	CT	MJJ	BAVD	BGVD
Initial BC	0.92±0.01	4.37±0.06	3.65±0.04	2.04±0.02	228.8±2.6	0.418±0.004	0.0052±0.0003
BC_{acc}	0.99±0.00	3.59±0.07	3.08±0.06	1.95±0.03	176.6±2.6	0.278±0.015	0.0027±0.0003
QCP	1.00±0.00	2.82±0.09	1.98±0.09	1.72±0.00	130.5±6.4	0.173±0.019	0.0022±0.0002
Stack	SR	CPL	JPL	CT	OPL	MJJ	MCJ
Initial BC	0.46±0.02	1.98±0.03	8.16±0.24	6.64±0.05	5.72±0.11	153.2±3.0	39.2±0.6
BC_{acc}	0.46±0.02	1.68±0.05	7.59±0.22	8.62±0.86	5.05±0.09	80.4±6.7	19.2±1.2
QCP	0.79±0.21	1.33±0.02	3.63±0.06	5.85±0.03	2.75±0.12	58.5±4.1	15.4±0.9
Handover	SR	CPL	OPL	MJJ	MCJ	BAVD	BGVD
Initial BC	0.96±0.00	1.10±0.09	2.24±0.29	169.6±10.6	48.1±3.2	0.293±0.004	0.0253±0.0008
BC_{acc}	0.95±0.04	0.77±0.05	1.13±0.15	67.1±0.8	19.4±0.8	0.215±0.004	0.0236±0.0010
QCP	1.00±0.00	0.73±0.01	0.82±0.05	36.2±1.7	11.5±0.5	0.127±0.013	0.0172±0.0032
Valve	SR	JPL	OPL	CT	MJJ	BAVD	BGVD
Initial BC	1.00±0.00	1.98±0.11	1.68±0.08	2.03±0.08	62.8±0.4	0.486±0.023	0.0094±0.0004
BC_{acc}	1.00±0.00	1.60±0.02	1.37±0.02	1.71±0.02	56.0±3.4	0.413±0.030	0.0096±0.0003
QCP	1.00±0.00	1.46±0.03	1.28±0.02	1.44±0.13	44.1±1.8	0.225±0.017	0.0048±0.0022

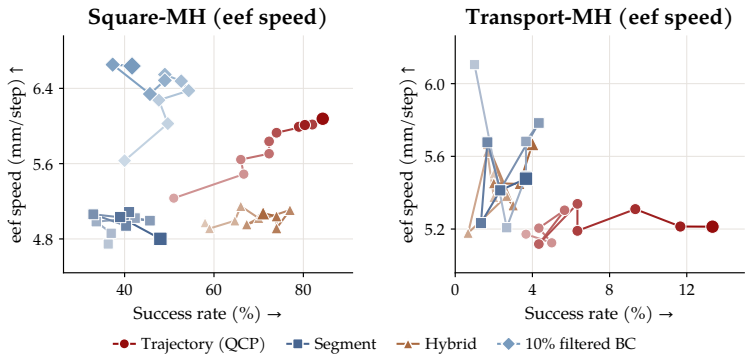

Figure 10: **Quality labels for shorter segments.** Success rate versus mean end-effector (EEF) displacement per step over successful rollouts on Square and Transport. We compare complete trajectory labels (QCP), segment labels, hybrid labels, and 10% filtered BC over iterations 1–10; darker points indicate later iterations. Transport 10% filtered BC has no successful rollouts, so its quality values are undefined and omitted.

tecture from the effect of the iterative hindsight relabeling loop. The frozen arm has the architecture but no further loop iterations, so any additional shift of the iterative arm comes from the loop rather than from conditioning or sampling alone. We freeze the iteration-1 policy rather than the iteration-0 policy so that both arms start from a policy that has already seen one round of self-collected data. Statistics are computed over successful rollouts pooled across seeds; per-seed medians agree with the pooled values to within 0.002. The frozen policy has a success rate of 54.5% and a median action jerk of 0.0335 (interquartile range 0.028 to 0.043) over 2,943 successful rollouts, against 57.0% and 0.0324 at iteration 1 of the loop. By iteration 6 the loop reaches 73.9% and a median of 0.0098 (interquartile range 0.0088 to 0.0109). The smoothest frozen rollout has a jerk of 0.0171, above the 75th percentile of the iteration-6 distribution. The later iterations therefore produce behavior that the iteration-1 policy does not generate at all, not merely behavior that it generates rarely: the quality-conditioned architecture alone does not reach this range, and the additional shift comes from the iterative hindsight relabeling loop.

Quality representation (Section 5.2). This experiment examines how the quality condition should be represented: as a quantized label, and if so with how many bins, or as a continuous score. Figure 9 compares six variants on the Square speed attribute, all run with the same loop (30k training steps and 300 rollouts per iteration, seven policies, three seeds). Four variants use quantized labels with $B = 4$, 8, 16, or 32 equal-frequency bins, recomputed at every iteration, and always request the best bin; with more bins the requested bin covers a smaller share ($1/B$) of the training data. Two variants use a continuous score instead of a label: the score is normalized so that the best and worst initial demonstrations map to 0 and 1, this normalization is fixed for the whole loop, and the requested value is set once to the 25th or the 12.5th percentile of the initial demonstrations. Curves are means over seeds and bands are one standard deviation. At iteration 6, the four-, eight-, sixteen-, and thirty-two-bin variants reach 70.3%, 63.2%, 57.7%, and 28.1% success with 139, 120, 116, and 112 steps; the two continuous variants reach 55.0% and 67.0% success with 246 and 209 steps.

More bins make executions slightly faster but cost success, sharply so beyond sixteen bins, and neither continuous target matches the bins on either axis at any iteration. Because the continuous variants also use a different way of selecting the requested value, this experiment supports quartile labels with per-iteration recomputation as a reliable default rather than ruling out continuous conditioning in general.

G Quality Labels for Shorter Segments

This experiment tests whether a quality label assigned to shorter segments of a trajectory is a better training signal than one label for the complete trajectory. We compare three labeling options. *Trajectory* is QCP as used in the main experiments: one label per trajectory. *Segment* assigns a label to every decision step from the score measured in a window of ± 8 steps around it. *Hybrid* conditions on both the trajectory label and the segment label through two embeddings. A segment score depends strongly on the stage of the task (for example, approaching versus inserting), so for the segment labels the quartile boundaries are computed separately within each of ten intervals of normalized trajectory time, and the best quartile is requested at every step. All options run the same loop otherwise.

Figure 10 shows the comparison on the end-effector speed attribute for Square and Transport, with 10% filtered BC as a reference, over ten iterations with 100k training steps and 300 rollouts per iteration and one seed. At iteration 10 on Square, the trajectory label reaches 84.3% success at 6.08 mm per step, the segment label 48.0% at 4.80, and the hybrid 71.0% at 5.07; on Transport, the trajectory label reaches 13.3% success against 3.7% and 4.0% for the segment and hybrid labels, at similar speeds (5.2 to 5.7 mm per step). The trajectory label is therefore the only option that improves success and speed together on Square, and the only one that raises success on Transport. A segment label provides a more local signal, but the signal reflects the stage of the task as much as the quality of execution even after the stage-wise normalization, and the policy can improve a local score without making progress toward completion. These results support labeling complete trajectories; using finer quality signals may require methods that separate quality from task progress, which we leave to future work.