CROSS-REALIZATION CONSISTENCY: REDUCING RUN-TO-RUN VARIATION IN FURTHER TRAINING OF ROBOT MANIPULATION POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Even with the same data, objective, and training recipe, repeated further training of a robot policy can produce different functional changes because of training randomness. Stabilization approaches based on keeping the updated model close to the original policy can limit how much each run changes, but do not directly encourage similar functional changes across runs. We introduce Cross-Realization Consistency (CRC), which aligns the functional changes of two branches further trained from the same original policy. For independent runs, the shared change cancels in their functional difference, allowing CRC to target run-specific variation without requiring either branch to remain close to the original policy. Across eight MimicGen manipulation tasks using ACT-lite, CRC reduces functional difference by 21% on average and improves directional alignment on all tasks while maintaining average task success. Runs trained with CRC also behave more similarly during execution, in both their actions and end-effector trajectories.

1 INTRODUCTION

Learning robot policies from demonstrations is common in complex manipulation tasks (Mandlekar et al., 2022; Zhao et al., 2023; Chi et al., 2023). However, repeated training can yield different outcomes across runs even with the same training configuration (Codevilla et al., 2019; Zhang et al., 2025). Such run-to-run differences can also be visible during execution, with runs starting from the same simulator state and all succeeding yet following different end-effector trajectories (Figure 1). This variation can come from training randomness, such as initialization and mini-batch order (Dodge et al., 2020; Bouthillier et al., 2021). It then becomes difficult to distinguish whether an observed functional change reflects the further-training procedure or a particular random run (Henderson et al., 2018; D'Amour et al., 2022). We therefore ask whether repeated further training under a fixed procedure produces consistent functional changes across runs.

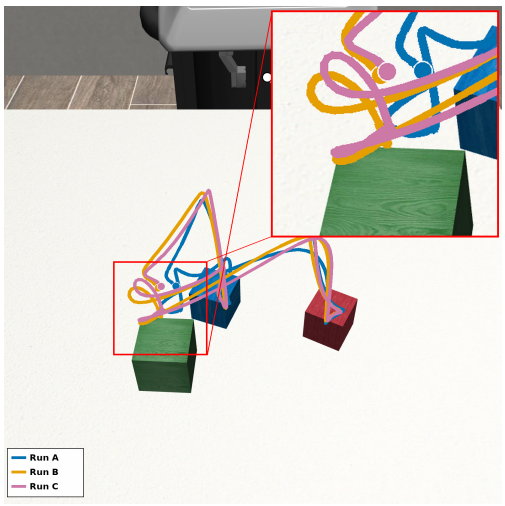


Figure 1: **Run-to-run variation.** Three runs under the same training setup start from the same simulator state and all succeed, yet follow different end-effector trajectories.

A widely used way to stabilize further training is to keep the updated model close to the original model through parameter or prediction constraints (Li et al., 2018; Lee et al., 2020; Jiang et al., 2022). These constraints compare each run only with the original model and do not relate different runs to one another. They can limit the magnitude of change in each run but do not directly align the direction of change across runs. This motivates relating the runs directly rather than constraining each one only relative to the original model.

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.

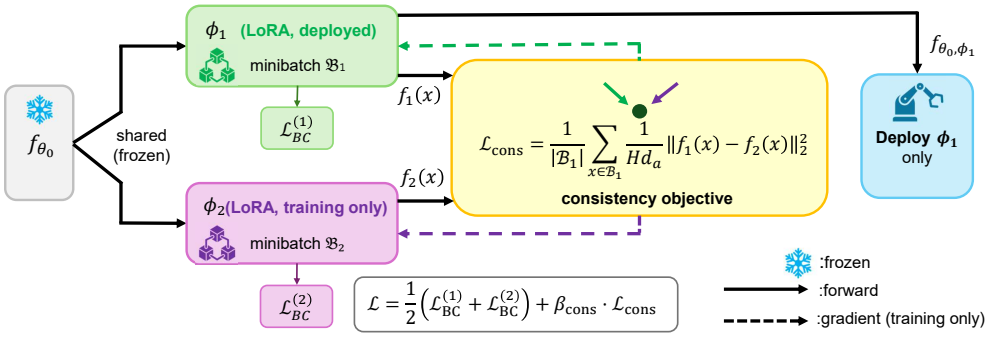


Figure 2: **Overview of CRC.** Two LoRA branches on the same fixed original policy use separate behavior-cloning minibatches. A consistency objective aligns their functional changes, and only branch 1 is deployed.

To relate runs directly, we propose Cross-Realization Consistency (CRC), which further trains two branches from the same original policy under different training randomness and aligns their functional changes. Each branch retains its own behavior-cloning objective, while a consistency term aligns their outputs on the same input. Rather than comparing each branch with the original policy, the consistency term compares the branches with each other and constrains their difference rather than the magnitude of each branch's change. Similar peer-matching ideas appear in mutual learning and codistillation (Zhang et al., 2018; Anil et al., 2018).

Branch alignment is motivated by what the difference between independently trained runs reveals. We decompose each run's functional change into a shared change—the mean functional change across runs—and a run-specific component. For independent runs, the shared change cancels when their functional changes are compared, leaving only the difference between their run-specific components; aligning the branches therefore targets run-specific variation without requiring either branch to stay close to the original policy. CRC branches interact through the consistency loss during training, so the independent-run decomposition above motivates the design rather than exactly describing the trained branches.

We evaluate CRC with a lightweight ACT-style policy (ACT-lite), based on action chunking and temporal ensembling (Zhao et al., 2023), on eight MimicGen manipulation tasks (Mandlekar et al., 2023), comparing runs on the same inputs and from the same initial states. CRC reduces functional difference and improves directional alignment on all eight tasks while maintaining average task success. Run pairs trained with CRC also show smaller differences in executed actions and end-effector trajectories. Additional controls show that the dual-branch structure alone has little effect, shared-minibatch alignment already provides a substantial gain, and a larger batch does not explain the full effect. At similar functional difference, neither post-hoc shrinkage nor training-time L2 regularization reproduces CRC's directional alignment.

2 Related Work

Policy Refinement. Robot imitation policies are frequently refined through further training (Ghosh et al., 2024; Kim et al., 2025; Zhang et al., 2025). Many methods update a trained policy with newly collected data, including corrections, interventions, and deployment experience (Ross et al., 2011; Kelly et al., 2019; Mandlekar et al., 2020; Hoque et al., 2022; Liu et al., 2023). These methods use further training mainly to improve task success as new data arrive. Our setting instead keeps the data, objective, and training recipe fixed across runs, isolating run-to-run variation from training randomness.

Variation Across Training Runs. The same training setup can yield different outcomes across runs (Reimers & Gurevych, 2017; Henderson et al., 2018; Bouthillier et al., 2021). Even with similar performance, models can make different predictions (Bahri & Jiang, 2021; Marx et al., 2020; D'Amour et al., 2022), and such variation has also been studied in fine-tuning and robot learning (Dodge et al., 2020; Mosbach et al., 2021; Zhang et al., 2025). Several approaches stabilize further training by keeping parameters or predictions close to the original model (Li et al., 2018; Lee et al., 2020; Jiang et al., 2022). Other approaches directly reduce prediction differences across retrained models (Anil et al., 2018; Bhojanapalli et al., 2021). CRC instead focuses on variation in the functional changes produced by further training from a shared original policy.

Aligning Multiple Models. Aligning model outputs during training is widely used in distillation and consistency learning (Hinton et al., 2015; Laine & Aila, 2017; Tarvainen & Valpola, 2017; Liang et al., 2021). Mutual learning and codistillation train peer models to match each other's predictions (Zhang et al., 2018; Anil et al., 2018), while codistillation has also been used to reduce prediction churn across retrained models (Bhojanapalli et al., 2021). CRC does not introduce peer alignment itself. Both branches are further trained from the same original policy, so matching their outputs is equivalent to matching their functional changes from that policy. Section 3.3 uses the corresponding independent-run decomposition as the design rationale. The branches also use separate behavior-cloning minibatch streams, whereas the deep mutual learning formulation of Zhang et al. (2018) trains both models on the same minibatches.

3 Cross-Realization Consistency

Cross-Realization Consistency (CRC) further trains two branches from the same original policy and aligns their functional changes during training (Figure 2).

3.1 Problem Setup

Let $\mathcal{D} = \{(x_i, a_i)\}_{i=1}^{N}$ be a behavior-cloning dataset, where x_i is an observation context and $a_i \in \mathbb{R}^{Hd_a}$ is a target action chunk with horizon H and action dimension d_a. We start from a previously learned original policy f_{θ_0} and further train it with a procedure C whose data, objective, architecture, and training recipe are fixed. The result of one run is

$$\theta_s = C(\theta_0, \mathcal{D}; s),$$

where s denotes the training randomness of the run, such as the initialization of trainable parameters and the minibatch order. Each run thus corresponds to one realization s of this randomness.

We describe the effect of further training by the functional change, defined as the change in policy output from the original policy for the same input:

$$\Delta f_s(x) = f_{\theta_s}(x) - f_{\theta_0}(x). \tag{1}$$

Training randomness varies across runs, allowing the same setup to produce different functional changes, $\Delta f_{s_1}(x) \neq \Delta f_{s_2}(x)$. We decompose the functional change as

$$\Delta f_s(x) = \mu(x) + \varepsilon_s(x), \qquad \mu(x) = \mathbb{E}_s[\Delta f_s(x)], \tag{2}$$

where the shared change $\mu(x)$ is the mean functional change across runs, and $\varepsilon_s(x) = \Delta f_s(x) - \mu(x)$ is the run-specific component. Across runs, these components represent the run-specific functional variation that we aim to reduce. This functional variation is distinct from task-success variation: runs with similar task success can still produce different functional changes.

3.2 Aligning Functional Changes

In standard further training, each run is trained only with its own behavior-cloning objective, so no term connects the functional changes of different runs. CRC instead further trains two branches from the same original policy under different training randomness and aligns their functional changes (Figure 2). Each branch $k \in \{1, 2\}$ is parameterized by ϕ_k on top of the fixed original policy, $f_k = f_{\theta_0, \phi_k}$, and continues to learn the task with its own behavior-cloning objective:

$$\mathcal{L}_{\mathrm{BC}}^{(k)} = \mathbb{E}_{(x,a)\sim\mathcal{D}}\left[\frac{1}{Hd_a}\,\|f_k(x) - a\|_2^2\right]. \tag{3}$$

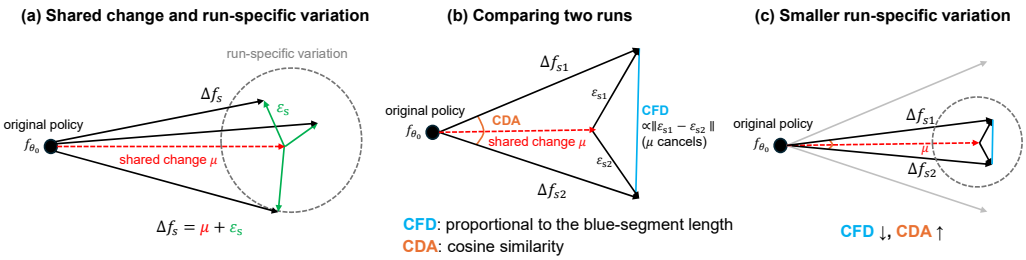


Figure 3: **Functional changes of runs.** Arrows show the change from the original policy for a single input. (a) The change of each run is the sum of the shared change μ and its run-specific component ε_s. (b) For independent runs, μ cancels in their difference; CFD measures the size of this difference, and CDA the cosine of the angle between the two changes. (c) Smaller run-specific components lower CFD and, for a fixed shared change, tend to raise CDA.

In addition, CRC introduces a consistency objective that reduces the difference between the functional changes of the two branches, $\Delta f_k(x) = f_k(x) - f_{\theta_0}(x)$. Because both branches share the same original policy,

$$\Delta f_1(x) - \Delta f_2(x) = \big[f_1(x) - f_{\theta_0}(x)\big] - \big[f_2(x) - f_{\theta_0}(x)\big] = f_1(x) - f_2(x),$$

so the consistency objective can be computed from the outputs of the two branches without a separate forward pass through the original policy:

$$\mathcal{L}_{\text{cons}} = \mathbb{E}_{x\sim\mathcal{D}}\left[\frac{1}{Hd_a}\,\|f_1(x) - f_2(x)\|_2^2\right]. \tag{4}$$

The overall objective is

$$\mathcal{L} = \frac{1}{2}\left(\mathcal{L}_{\text{BC}}^{(1)} + \mathcal{L}_{\text{BC}}^{(2)}\right) + \beta_{\text{cons}}\,\mathcal{L}_{\text{cons}}, \tag{5}$$

where $\beta_{\text{cons}} > 0$ controls its strength. This term is optimized jointly with both behavior-cloning objectives, with gradients applied to both branches. Section 3.3 explains why this objective targets run-specific functional variation, and Section 3.4 describes how the two branches are constructed and deployed.

3.3 Why Two Branches

To see why Eq. 4 targets run-specific functional variation, we examine what a single run and two runs reveal.

In a single run, Δf_s combines the shared change μ with the run-specific component ε_s, so they cannot be separated from that run alone. A constraint on the size of Δf_s acts on the whole functional change rather than isolating ε_s.

For independently trained runs s_1 and s_2 under the same procedure, taking the difference between their functional changes cancels the shared change. Independence of the two runs, together with the zero-mean property of $\varepsilon_s(x)$, gives

$$\mathbb{E}_{s_1,s_2}\left[\|\Delta f_{s_1}(x) - \Delta f_{s_2}(x)\|_2^2\right] = 2\,\mathbb{E}_s\left[\|\varepsilon_s(x)\|_2^2\right], \tag{6}$$

so the difference between two independent runs depends only on their run-specific functional variation.

$\mathcal{L}_{\text{cons}}$ directly penalizes the difference between the two branches rather than either branch's distance from the original policy. The branches influence each other during training and are not independent, so Eq. 6 is a design rationale rather than an exact property of CRC.

Table 1: **Taskwise functional variation and task success on eight MimicGen tasks with ACT-lite.** CFD measures the magnitude of run-to-run functional variation, while CDA measures directional alignment of functional changes. Success is reported as a task-competence check. Functional metrics are averaged over evaluation inputs and all pairs of three training runs (with a five-run extension in Appendix D.3); success is averaged over the three runs. Std. denotes standard further training, Drop. denotes dropout consistency, and Mutual denotes coefficient-matched mutual learning. Best CFD and CDA values within each task are bolded. CRC improves both metrics over every compared method on all eight tasks (each two-sided exact Wilcoxon signed-rank test over tasks, $p = 0.0078$; Wilcoxon, 1945).

	CFD ↓				CDA ↑				Success (%) ↑			
Task	Std.	Drop.	Mutual	CRC	Std.	Drop.	Mutual	CRC	Std.	Drop.	Mutual	CRC
Kitchen	0.02696	0.02538	0.02271	**0.02069**	0.9100	0.9157	0.9290	**0.9384**	80.50	82.17	83.83	84.00
Stack Three	0.03485	0.03151	0.02842	**0.02701**	0.8728	0.9002	0.9131	**0.9209**	48.67	50.67	46.83	51.17
Hammer Cleanup	0.02429	0.02357	0.02133	**0.01929**	0.9121	0.9157	0.9280	**0.9397**	99.67	99.17	99.00	98.83
Stack	0.03252	0.03062	0.02725	**0.02589**	0.8823	0.8884	0.9043	**0.9141**	95.33	95.00	94.83	95.50
Threading	0.02402	0.02245	0.01985	**0.01786**	0.9082	0.9171	0.9310	**0.9439**	95.67	98.00	95.50	98.50
Three-Piece Assembly	0.03113	0.02995	0.02695	**0.02516**	0.9010	0.9044	0.9199	**0.9282**	6.00	5.17	5.50	8.17
Square	0.02969	0.02796	0.02567	**0.02357**	0.9093	0.9185	0.9281	**0.9381**	68.33	67.83	62.17	68.50
Mug Cleanup	0.02747	0.02631	0.02351	**0.02219**	0.9115	0.9204	0.9326	**0.9403**	64.67	68.00	66.67	67.17
Average	0.02886	0.02722	0.02446	**0.02271**	0.9009	0.9100	0.9232	**0.9329**	69.85	70.75	69.29	71.48

Under identical forward passes and matched loss weights, L_{cons} gives the same per-branch gradients as a coefficient-matched squared-error analogue of two-way mutual learning (Zhang et al., 2018; Appendix B.1). The reported training procedures nevertheless differ: Mutual uses a shared minibatch and two forward passes, whereas CRC uses separate behavior-cloning minibatches and an additional cross-branch evaluation.

Eq. 6 also indicates how to measure run-to-run variation. We compare the policies of different runs on a common set of inputs $\mathcal{X}$ (for CRC, the deployed branch of each run) using two quantities. Cross-run Functional Difference (CFD) measures the magnitude of the difference between the functional changes of two runs, and Cross-run Directional Alignment (CDA) measures how closely the two changes point in the same direction:

$$\begin{aligned} \text{CFD} &= \underset{(s,s'),\,x}{\text{mean}} \sqrt{\frac{1}{Hd_a} \left\|\Delta f_s(x) - \Delta f_{s'}(x)\right\|_2^2}, \\ \text{CDA} &= \underset{(s,s'),\,x}{\text{mean}} \frac{\langle \Delta f_s(x),\, \Delta f_{s'}(x) \rangle}{\left\|\Delta f_s(x)\right\|_2 \, \left\|\Delta f_{s'}(x)\right\|_2}, \end{aligned} \tag{7}$$

where the mean is taken over pairs of runs (s, s') and inputs $x \in \mathcal{X}$ (Figure 3(b); details in Appendix C). Uniformly shrinking the functional changes lowers CFD but does not change their directions because CDA is based on cosine similarity. A higher CDA instead reflects closer directional alignment, which is expected when the run-specific components are small relative to the shared change; Section 4.3 examines this distinction with post-hoc shrinkage and training-time L2 regularization.

3.4 Training and Deployment

Each branch is a LoRA adapter (Hu et al., 2022) on the fixed original policy. At each step, the two branches compute their behavior-cloning losses on independently sampled minibatches $\mathcal{B}_1$ and $\mathcal{B}_2$, while the consistency loss compares both branches on the inputs from $\mathcal{B}_1$. CRC therefore uses three forward passes per update. In the CPU benchmark reported in Table A.2, CRC takes $2.94\times$ the training-loop time of Std. After training, only branch 1 is deployed with its LoRA weights merged into the original policy, leaving the inference architecture unchanged. Appendix A provides the full training settings and timing protocol.

4 Experiments

The experiments address whether CRC reduces run-to-run functional variation, whether the gain also appears during closed-loop execution, and what explains the gain. For each of eight MimicGen

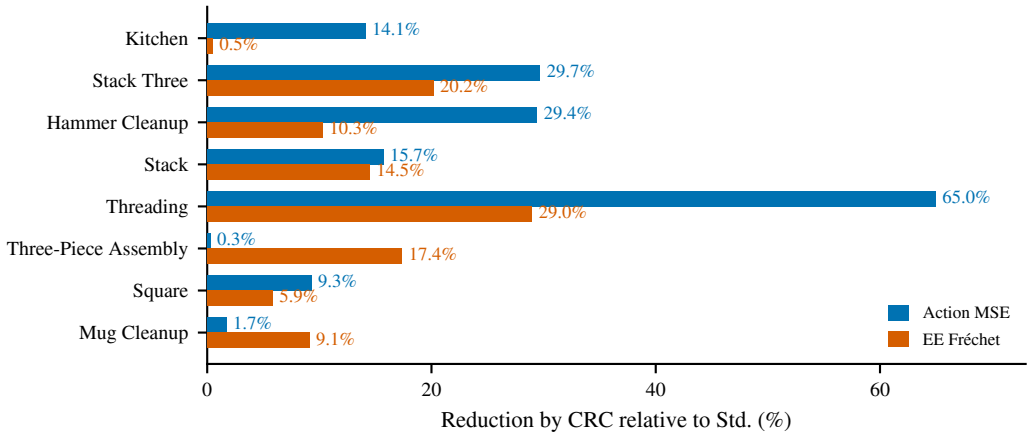


Figure 4: **Taskwise reduction in closed-loop variation.** For each task, the bars show how much CRC reduces the difference among three runs relative to Std.

tasks (Mandlekar et al., 2023), we train one ACT-lite policy and use its final checkpoint as the original policy for further training. Each method uses three runs on the same demonstrations used to train the original policy. CRC uses $\beta_{\text{cons}} = 0.5$, carried over from an earlier experimental configuration and kept fixed across all tasks and runs without re-tuning. Closed-loop comparisons use identical simulator reset conditions; full settings are provided in Appendices A and C.

4.1 Run-to-Run Functional Variation

First, we compare policies obtained from different training runs on a fixed input set $\mathcal{X}$ drawn from the validation split (Appendix C.1). Evaluation uses independently trained runs, while CRC aligns only the two branches within each run. The baselines include standard further training (Std.), dropout consistency (Drop.; Liang et al., 2021), and coefficient-matched mutual learning (Mutual; Zhang et al., 2018); Drop. aligns the outputs of two dropout passes of one model, Mutual matches two models on the same minibatches, and CRC uses branches trained on separate behavior-cloning minibatch streams (Appendix B). Across all eight tasks, CRC reduces CFD and improves CDA (Table 1), indicating smaller functional differences and stronger directional alignment across runs. Task success serves as a performance check; CRC maintains average success. For Std. and CRC, this 8/8 pattern persists with five runs (CFD: 0.02898→0.02288; CDA: 0.9007→0.9321; Appendix D.3); separately, the CFD reduction remains about 21% under local input perturbations (Appendix D.4).

4.2 Closed-Loop Variation

Section 4.1 compares policies on the same inputs, but during execution each policy receives states caused by its own earlier actions, allowing small differences to compound over time (Ross et al., 2011). To capture this effect, we roll out each run from the same initial states and measure differences in executed actions (action MSE) and end-effector trajectories (EE Fréchet; Eiter & Mannila, 1994). Even under Std., runs trained with the same setup execute different actions and follow different trajectories (Table 2). Across all eight tasks, CRC reduces both measures relative to Std. (Figure 4; Appendix D.2). Drop. and Mutual also reduce these differences to similar levels, with Mutual slightly lower than CRC in action MSE. Runs further trained with CRC thus show smaller differences both on fixed inputs and during execution.

Table 2: **Run-to-run closed-loop variation.** Averaged over eight tasks and three runs; notation as in Table 1.

	Action MSE ↓	EE Fréchet ↓
Std.	0.5117	0.1076
Drop.	0.4496	0.0954
Mutual	**0.4405**	0.0952
CRC	0.4482	**0.0929**

Table 3: **Controls for the functional gains of CRC.** Structural controls test the dual-branch structure, shared minibatches, and increased sample exposure. Magnitude/proximity controls compare CRC with post-hoc shrinkage and training-time L2 regularization at similar CFD. $\bar{\Delta}$ denotes the average size of the functional change (Eq. C.1). All reported control coefficients are shared across tasks; no control uses task-specific coefficients. Values are averaged over eight tasks using three training runs per task.

Method	CFD ↓	CDA ↑	$\bar{\Delta}$	Action MSE ↓	EE Fréchet ↓	Success (%) ↑
Std.	0.02886	0.9009	0.07127	0.5117	0.1076	69.85
Structural controls						
$\beta_{\text{cons}} = 0$	0.02865	0.9022	0.07120	0.4891	0.0979	70.10
Shared batch	0.02457	0.9225	0.06860	0.4656	0.0976	71.27
Batch-512	0.02738	0.9152	0.07312	0.4464	0.0985	69.35
Magnitude/proximity controls						
Shrinkage (matched, $\lambda = 0.8$)	0.02292	0.9017	0.05687	0.4563	0.0966	68.44
L2 (matched, $c = 0.01$)	0.02340	0.9217	0.06520	0.4613	0.0978	68.96
CRC (proposed)	**0.02271**	**0.9329**	0.06827	0.4482	0.0929	71.48

4.3 Ablation and Control Analysis

Five controls examine whether CRC's functional gain can be attributed to the dual-branch structure, shared minibatches, increased sample exposure, smaller functional changes, or proximity to the original policy (Table 3).

Dual-branch structure. To analyze whether using two branches alone explains the gain, we remove the consistency term while keeping the dual-branch training structure ($\beta_{\text{cons}} = 0$). Its CFD and CDA (0.02865 and 0.9022) remain close to Std. (0.02886 and 0.9009), showing that the dual-branch structure alone has little effect on run-to-run functional variation.

Shared minibatches. A shared-batch variant keeps the branch-alignment objective but feeds the same behavior-cloning minibatch to both branches. This control already improves CFD and CDA to 0.02457 and 0.9225, while CRC further improves both metrics on all eight tasks, reaching 0.02271 and 0.9329 (Appendix E.1). Thus, branch alignment itself reduces functional variation, with separate behavior-cloning minibatch streams providing an additional functional gain.

Increased sample exposure. Separate behavior-cloning minibatches expose CRC to more training-example draws per update, so a batch-size control tests whether increased sample exposure alone accounts for the gain. Increasing the Std. batch size to 512 without cross-branch consistency yields CFD 0.02738 and CDA 0.9152, improving over Std. but remaining behind CRC. Shared batch also outperforms Batch-512 on both metrics across all eight tasks (Appendix E.2), so the functional gain cannot be explained by increased sample exposure alone.

Change magnitude. A post-hoc shrinkage control examines whether smaller functional changes alone account for CRC's gain by scaling completed Std. updates toward the original policy. At $\lambda = 0.8$, shrinkage nearly matches CRC in CFD (0.02292 vs. 0.02271) but not in CDA (0.9017 vs. 0.9329). It also reduces the average functional-change size by 20% relative to Std., compared with 4% for CRC; $\bar{\Delta}$ is reported to characterize how much the policy changes, not as a performance criterion. This matched CFD with much lower CDA shows that reducing the size of the difference between runs is not equivalent to aligning their functional-change directions.

Training-time proximity. We finally compare against L2 regularization toward the original policy. For a matched comparison, we use the tested L2 strength whose average CFD across the eight tasks is closest to that of CRC; the full sweep, including stronger settings ($c \geq 0.03$) with lower average CFD and higher average CDA than CRC, is reported in Appendix E.4. At similar CFD (0.02340 vs. 0.02271), L2 gives lower CDA (0.9217 vs. 0.9329) while reducing the average functional-change size by 9% relative to Std., compared with 4% for CRC. Training-time proximity can therefore

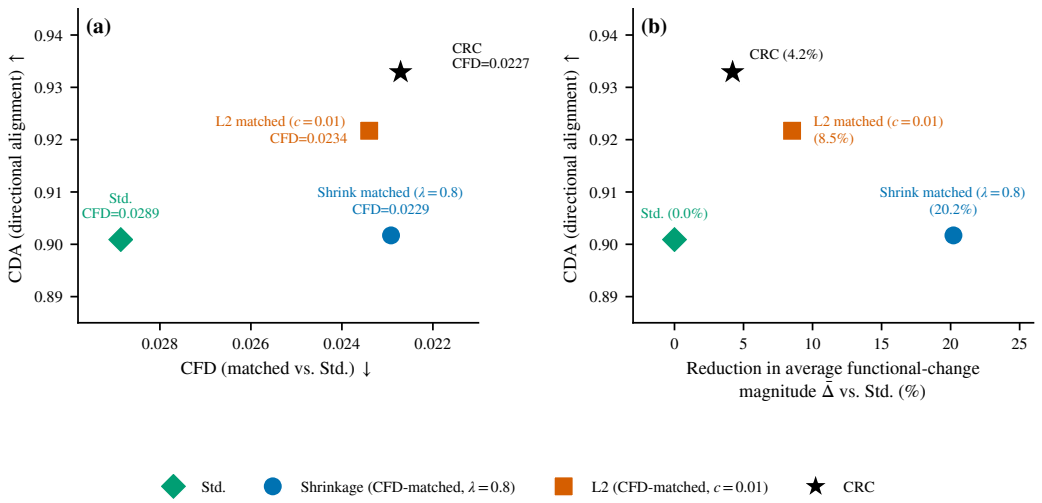


Figure 5: **Matched controls separate functional-difference magnitude from directional alignment.** (a) At similar CFD, post-hoc shrinkage and training-time L2 regularization yield lower CDA than CRC. (b) At the matched-CFD operating points, CRC achieves higher CDA with a smaller reduction in average functional-change magnitude than shrinkage and L2. Points are computed from three training runs per task and averaged over the eight tasks. The full shrinkage and L2 sweeps are reported in Appendices E.3 and E.4, respectively.

reduce run-to-run variation, but the selected L2 setting yields lower directional alignment and a smaller average functional-change magnitude than CRC at similar CFD.

Together, these controls show that CRC's functional gain is not explained by branch duplication or increased sample exposure alone. Shared-batch alignment already provides a substantial gain, with separate behavior-cloning minibatch streams giving an additional improvement. At similar CFD, the selected shrinkage and L2 settings yield lower CDA and smaller average functional-change magnitudes than CRC (Figure 5).

5 Conclusion

We studied run-to-run functional variation when a learned policy is further trained with the data, objective, and training recipe fixed. CRC reduces this variation by aligning two branches further trained from the same original policy. Across eight MimicGen manipulation tasks using ACT-lite, CRC reduces the magnitude of functional differences and improves directional alignment while maintaining average task success, with smaller differences also observed in executed actions and end-effector trajectories. Controls show that dual-branch structure and more samples per step do not fully explain the result; post-hoc shrinkage and CFD-matched L2 regularization reduce functional difference but do not reproduce CRC's directional alignment, while shared-batch consistency already gives a substantial gain and separate behavior-cloning minibatch streams add a further improvement. Our main evidence is limited to eight simulated MimicGen D0 tasks with low-dimensional observations and ACT-lite, with further training using the same training split as the original policy; broader policy architectures, visual inputs, new-data settings, and real-robot settings remain open. Overall, run-to-run functional variation can be measured and reduced through training design rather than inferred from one run's task success alone.

REPRODUCIBILITY STATEMENT

Appendix A specifies the model, the training settings, one training step of CRC (Algorithm 1), and the training cost; Appendix B gives the objectives and settings of the baselines; and Appendix C describes the data split, the evaluation protocol, and the computation of all metrics. All methods use the same fixed original policy for each task and fixed sets of evaluation inputs and initial states. The main comparisons use run seeds 24, 29, and 30; the five-run analysis for Std. and CRC additionally uses seeds 37 and 38 (Appendix D.3). A targeted rollout reproducibility check is described in Appendix C.1.

AI USE STATEMENT

Generative AI tools, including ChatGPT and Claude, were used in a supporting role for refining hypotheses and conceptual framing, providing feedback on experimental design and controls, assisting with result interpretation, code development and debugging, literature search and organization, translation, and manuscript drafting and editing. All methodological choices, experimental configurations, reported results, citations, and scientific claims were reviewed and validated by the authors. The authors take full responsibility for the design, execution, analysis, and final content of this work.

REFERENCES

Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E. Dahl, and Geoffrey E. Hinton. Large scale distributed neural network training through online distillation. In *International Conference on Learning Representations*, 2018.

Dara Bahri and Heinrich Jiang. Locally adaptive label smoothing improves predictive churn. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 532–542. PMLR, 2021.

Srinadh Bhojanapalli, Kimberly Wilber, Andreas Veit, Ankit Singh Rawat, Seungyeon Kim, Aditya Menon, and Sanjiv Kumar. On the reproducibility of neural network predictions. *arXiv preprint arXiv:2102.03349*, 2021.

Xavier Bouthillier, Pierre Delaunay, Mirko Bronzi, Assya Trofimov, Brennan Nichyporuk, Justin Szeto, Nazanin Mohammadi Sepahvand, Edward Raff, Kanika Madan, Vikram Voleti, Samira Ebrahimi Kahou, Vincent Michalski, Tal Arbel, Chris Pal, Gaël Varoquaux, and Pascal Vincent. Accounting for variance in machine learning benchmarks. *Proceedings of Machine Learning and Systems*, 3:747–769, 2021.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin C. M. Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi: 10.15607/RSS.2023.XIX.026.

Felipe Codevilla, Eder Santana, Antonio M. López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9329–9338, 2019.

Alexander D'Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D. Hoffman, Farhad Hormozdiari, Neil Houlsby, Shaobo Hou, Ghassen Jerfel, Alan Karthikesalingam, Mario Lucic, Yian Ma, Cory McLean, Diana Mincu, Akinori Mitani, Andrea Montanari, Zachary Nado, Vivek Natarajan, Christopher Nielson, Thomas F. Osborne, Rajiv Raman, Kim Ramasamy, Rory Sayres, Jessica Schrouff, Martin Seneviratne, Shannon Sequeira, Harini Suresh, Victor Veitch, Max Vladymyrov, Xuezhi Wang, Kellie Webster, Steve Yadlowsky, Taedong Yun, Xiaohua Zhai, and D. Sculley. Underspecification presents challenges for credibility in modern machine learning. *Journal of Machine Learning Research*, 23(226):1–61, 2022.

Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah A. Smith. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*, 2020.

Thomas Eiter and Heikki Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Christian Doppler Laboratory for Expert Systems, TU Vienna, 1994.

Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024. doi: 10.15607/RSS.2024.XX.090.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, pp. 3207–3214, 2018. doi: 10.1609/aaai.v32i1.11694.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Ryan Hoque, Ashwin Balakrishna, Ellen Novoseller, Albert Wilcox, Daniel S. Brown, and Ken Goldberg. ThriftyDAgger: Budget-aware novelty and risk gating for interactive imitation learning. In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 598–608. PMLR, 2022.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Heinrich Jiang, Harikrishna Narasimhan, Dara Bahri, Andrew Cotter, and Afshin Rostamizadeh. Churn reduction via distillation. In *International Conference on Learning Representations*, 2022.

Michael Kelly, Chelsea R. Sidrane, Katherine R. Driggs-Campbell, and Mykel J. Kochenderfer. HG-DAgger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8077–8083. IEEE, 2019. doi: 10.1109/ICRA.2019.8793698.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 2025.

Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. In *International Conference on Learning Representations*, 2017.

Cheolhyoung Lee, Kyunghyun Cho, and Wanmo Kang. Mixout: Effective regularization to finetune large-scale pretrained language models. In *International Conference on Learning Representations*, 2020.

Xuhong Li, Yves Grandvalet, and Franck Davoine. Explicit inductive bias for transfer learning with convolutional networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2825–2834. PMLR, 2018.

Xiaobo Liang, Lijun Wu, Juntao Li, Yue Wang, Qi Meng, Tao Qin, Wei Chen, Min Zhang, and Tie-Yan Liu. R-Drop: Regularized dropout for neural networks. In *Advances in Neural Information Processing Systems*, volume 34, pp. 10890–10905, 2021.

Huihan Liu, Soroush Nasiriany, Lance Zhang, Zhiyao Bao, and Yuke Zhu. Robot learning on the job: Human-in-the-loop autonomy and learning during deployment. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi: 10.15607/RSS.2023.XIX.005.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Yuke Zhu, Li Fei-Fei, and Silvio Savarese. Human-in-the-loop imitation learning using remote teleoperation. *arXiv preprint arXiv:2012.06733*, 2020.

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 *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 1678–1690. PMLR, 2022.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. MimicGen: A data generation system for scalable robot learning using human demonstrations. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 1820–1864. PMLR, 2023.

Charles Marx, Flavio Calmon, and Berk Ustun. Predictive multiplicity in classification. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 6765–6774. PMLR, 2020.

Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. On the stability of fine-tuning BERT: Misconceptions, explanations, and strong baselines. In *International Conference on Learning Representations*, 2021.

Nils Reimers and Iryna Gurevych. Reporting score distributions makes a difference: Performance study of LSTM-networks for sequence tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 338–348. Association for Computational Linguistics, 2017. doi: 10.18653/v1/D17-1035.

Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 627–635. PMLR, 2011.

Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, and Ludwig Schmidt. Robust fine-tuning of zero-shot models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7959–7971, 2022.

Wenbo Zhang, Yang Li, Yanyuan Qiao, Siyuan Huang, Jiajun Liu, Feras Dayoub, Xiao Ma, and Lingqiao Liu. Effective tuning strategies for generalist robot manipulation policies. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, 2025. doi: 10.1109/ICRA55743.2025.11127492.

Ying Zhang, Tao Xiang, Timothy M. Hospedales, and Huchuan Lu. Deep mutual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4320–4328, 2018.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi: 10.15607/RSS.2023.XIX.016.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Kevin Lin, Abhiram Maddukuri, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

Appendix A gives implementation details and training cost, Appendix B the baselines, and Appendix C the data, evaluation protocol, and metrics. Appendix D reports detailed results, and Appendix E the controls.

A Implementation Details

All methods share, for each task, the same original policy, data split, number of training steps, and evaluation protocol. Table A.1 lists the settings.

A.1 Policy, Original Policy, and LoRA

We use a lightweight encoder-only Transformer policy, referred to as ACT-lite, on low-dimensional MimicGen observations (Mandlekar et al., 2023). It adopts action chunking and temporal ensembling from ACT (Zhao et al., 2023), without its encoder–decoder or conditional variational formulation. No latent variable or KL loss is used. A three-step observation history is projected to a model dimension of 128 and processed by two Transformer encoder layers (Vaswani et al., 2017) with four attention heads, feed-forward dimension 512, GELU activations, and dropout 0.1. The final encoded timestep passes through LayerNorm and a linear head, producing a flat 35-dimensional vector that is reshaped into an action chunk with horizon $H = 5$ and action dimension $d_a = 7$. Behavior cloning minimizes mean-squared error on the normalized 35-dimensional action-chunk target.

For each task, the original policy f_{θ_0} is trained for 3000 steps with AdamW (Loshchilov & Hutter, 2019) (learning rate 10^{-3}, weight decay 10^{-5}, batch size 512), and its final checkpoint is used without early stopping or checkpoint selection. Observation and action normalization statistics are computed from the training split and reused unchanged in further training and evaluation.

Further training uses rank-4 LoRA (Hu et al., 2022) updates $W = W_0 + \frac{1}{r}BA$ with $r = 4$, Kaiming-uniform initialization for A, and zero initialization for B. LoRA is applied to the self-attention input and output projections and to the two feed-forward layers of each Transformer layer, giving eight adapted matrices and 16,384 trainable parameters per adapter. All other parameters of the original policy are frozen.

A.2 CRC Training

For a run with seed s, branch 1 uses seed s and branch 2 uses seed $s + 500$; LoRA initialization and minibatch sampling are seeded separately for the two branches. At every step, the minibatches $\mathcal{B}_1$ and $\mathcal{B}_2$ are sampled independently, with replacement, from the same training data. Dropout masks are drawn anew for every forward pass from the shared global random number generator, so the two branches differ in LoRA initialization and minibatch sampling rather than in separately seeded dropout.

The objective is $\mathcal{L} = \frac{1}{2}\big(\mathcal{L}_{\mathrm{BC}}^{(1)} + \mathcal{L}_{\mathrm{BC}}^{(2)}\big) + \beta_{\mathrm{cons}}\mathcal{L}_{\mathrm{cons}}$ with $\beta_{\mathrm{cons}} = 0.5$. Each behavior-cloning loss uses its own branch's minibatch, and $\mathcal{L}_{\mathrm{cons}}$ compares the two branches on the inputs of $\mathcal{B}_1$ without stop-gradient, so the consistency term updates both branches. Training runs for 1500 steps with batch size 256 per branch and AdamW (learning rate 10^{-3}, weight decay 10^{-5}), without learning-rate schedules, warmup, gradient clipping, mixed precision, early stopping, or checkpoint selection. Algorithm 1 summarizes one training step.

The branch-1 output on $\mathcal{B}_1$ is reused for behavior cloning and consistency; branch 2 is evaluated separately on $\mathcal{B}_2$ and $\mathcal{B}_1$, giving three forward passes per update.

Consistency coefficient. $\beta_{\mathrm{cons}} = 0.5$ was carried over from an earlier experimental configuration and kept fixed across all tasks and runs; it was not re-tuned for CRC. With the factor $\frac{1}{2}$ on the behavior-cloning losses, this value gives each branch equal gradient coefficients for matching its demonstrations and for matching the other branch.

Algorithm 1 One training step of CRC

Require: frozen original policy f_{θ_0}; LoRA branches ϕ_1, ϕ_2; training data $\mathcal{D}$; β_{cons}

1: Sample $\mathcal{B}_1 = \{(x_i, a_i)\}_{i=1}^n$ and $\mathcal{B}_2 = \{(x'_i, a'_i)\}_{i=1}^n$ independently from $\mathcal{D}$
2: $\mathcal{L}_{\text{BC}}^{(1)} \leftarrow \frac{1}{n}\sum_i \frac{1}{Hd_a}\|f_1(x_i) - a_i\|_2^2$ ▷ f_k: f_{θ_0} with branch ϕ_k
3: $\mathcal{L}_{\text{BC}}^{(2)} \leftarrow \frac{1}{n}\sum_i \frac{1}{Hd_a}\|f_2(x'_i) - a'_i\|_2^2$
4: $\mathcal{L}_{\text{cons}} \leftarrow \frac{1}{n}\sum_i \frac{1}{Hd_a}\|f_1(x_i) - f_2(x_i)\|_2^2$ ▷ inputs of $\mathcal{B}_1$, no stop-gradient
5: $\mathcal{L} \leftarrow \frac{1}{2}\big(\mathcal{L}_{\text{BC}}^{(1)} + \mathcal{L}_{\text{BC}}^{(2)}\big) + \beta_{\text{cons}}\mathcal{L}_{\text{cons}}$
6: Update ϕ_1 and ϕ_2 with one AdamW step on $\mathcal{L}$
After training, merge ϕ_1 into f_{θ_0} and deploy the result.

A.3 Deployment

Branch 1 was designated for deployment before the reported experiments, rather than selected through a post-hoc comparison of the two branches. After training, only branch 1 is deployed: its LoRA update is merged into the frozen weights, and the result is evaluated as an ordinary dense policy with dropout disabled, so inference cost equals that of the original policy. A new action chunk is predicted at every environment step, all overlapping chunk predictions for the current step are averaged with uniform weights (temporal ensembling; Zhao et al., 2023), and the resulting action is de-normalized using the fixed training-split statistics and clipped to $[-1, 1]$ before execution.

Table A.1: **Training and evaluation settings.**

Setting	Value
Observation history	3
Action horizon / dimension	5 / 7
Transformer layers / heads	2 / 4
Model / feed-forward dimension	128 / 512
Dropout	0.1
Original policy: steps / batch size	3000 / 512
Further training: steps / batch size	1500 / 256 per branch
LoRA rank / parameters per adapter	4 / 16,384
Optimizer / learning rate / weight decay	AdamW / 10^{-3} / 10^{-5}
Consistency coefficient β_{cons}	0.5
Inputs of the consistency term	$\mathcal{B}_1$
Run seeds	24, 29, 30 (37, 38 added for five runs)
Branch-2 seed	branch-1 seed + 500
Deployment	branch 1

A.4 Training Cost

Table A.2 compares training cost. All methods deploy a single policy with merged LoRA weights and the same inference architecture.

Wall-clock times were measured on Threading (seed 24) using an Intel Core i7-8700K CPU, with all runs pinned to the same four distinct physical cores. All methods used four PyTorch intra-op threads, one inter-op thread, and identical OpenMP/MKL/OpenBLAS thread settings of four. OpenMP and OpenBLAS thread-pool settings were verified at runtime using `threadpoolctl`. Each method was measured four times in fresh processes, with execution order counterbalanced across four blocks. These are timing repetitions with a fixed training seed, not independent training seeds. Process monitoring at 10-second intervals detected no other Python jobs during the benchmark.

Timing used `time.perf_counter()` and covered 1,500 training steps, from immediately before the first minibatch sampling to immediately after the final optimizer update. Initial dataset and checkpoint loading, model/optimizer initialization, logging, and final LoRA merging/checkpoint saving were excluded. All methods used rank-4 LoRA and a batch size of 256 per branch.

Table A.2: **Training cost per run.** Wall-clock times are mean ± sample standard deviation over four timing repetitions. Trainable parameters count all adapters optimized during training. Time ratios are computed from mean wall-clock times.

	Std.	Drop.	Mutual	CRC
Trainable parameters	16,384	16,384	32,768	32,768
Forward passes per update	1	2	2	3
Wall-clock time (s)	37.98 ± 0.11	74.45 ± 0.43	75.44 ± 0.24	111.57 ± 0.98
Time relative to Std.	$1.00\times$	$1.96\times$	$1.99\times$	$2.94\times$

CRC reuses the branch-1 output on $\mathcal{B}_1$ for behavior cloning and consistency, while branch 2 is evaluated separately on $\mathcal{B}_2$ and $\mathcal{B}_1$, giving three forward passes per update. In this configuration, CRC takes approximately $2.94\times$ the training-loop time of Std. and $1.5\times$ that of Drop. or Mutual. These timing ratios are specific to the measured hardware and configuration.

B BASELINES

All baselines use the same original policy, LoRA rank, training data, number of steps, batch size per branch or dropout pass, optimizer, and run seeds as CRC, and their consistency terms use the same coefficient $\beta = 0.5$. Table B.1 summarizes how they differ. Drop. and Mutual adapt the distribution-matching objectives of R-Drop and deep mutual learning to continuous action regression by using squared-error consistency.

Drop. Drop. (following Liang et al., 2021) trains a single LoRA adapter and passes each minibatch $\mathcal{B}$ through the policy twice with dropout active, so the two outputs $f^{(1)}(x)$ and $f^{(2)}(x)$ differ only in their dropout masks (rate 0.1 in all encoder layers). The objective is

$$\mathcal{L}_{\text{Drop}} = \tfrac{1}{2}\big(\mathcal{L}_{\text{BC}}(f^{(1)}) + \mathcal{L}_{\text{BC}}(f^{(2)})\big) + \beta \frac{1}{|\mathcal{B}|} \sum_{x \in \mathcal{B}} \frac{1}{H d_a} \big\| f^{(1)}(x) - f^{(2)}(x) \big\|_2^2, \tag{B.1}$$

with no stop-gradient. The single adapter is deployed.

Mutual. Mutual (following Zhang et al., 2018) trains two separately initialized LoRA adapters (seeds s and $s + 500$) on the same minibatch $\mathcal{B}$ at every step, and each adapter matches the other's output through a stop-gradient term:

$$\mathcal{L}_{\text{Mutual}} = \tfrac{1}{2}\big(\mathcal{L}_{\text{BC}}^{(1)} + \mathcal{L}_{\text{BC}}^{(2)}\big) + \beta\big(D(f_1, \text{sg}[f_2]) + D(f_2, \text{sg}[f_1])\big), \tag{B.2}$$

where $D(f, g) = \frac{1}{|\mathcal{B}|} \sum_{x \in \mathcal{B}} \frac{1}{H d_a} \|f(x) - g(x)\|_2^2$ and sg denotes stop-gradient. Each adapter is evaluated once per update, and its output is reused for both behavior cloning and mutual consistency. The first adapter is deployed. The factor $\frac{1}{2}$ on the behavior-cloning terms matches the loss weighting of CRC and Drop. We verified with automatic differentiation that the ratio between the consistency and behavior-cloning gradient coefficients received by each branch is 1 for both Mutual and CRC.

Table B.1: **How the baselines and CRC differ.** All other settings are shared.

	Drop.	Mutual	CRC
LoRA adapters trained	1	2	2
Minibatches per step	1	1 (shared)	2 (independent)
Outputs aligned	two dropout passes	two adapters	two branches
Inputs of the consistency term	the minibatch	the shared minibatch	$\mathcal{B}_1$
Stop-gradient	none	on the other adapter	none
Forward passes per update	2	2	3
Deployed model	the adapter	adapter 1	branch 1

B.1 Relation between Mutual Learning and CRC

For disjoint branch parameters, identical forward passes, and matched loss coefficients, symmetric squared-error consistency and the sum of two directional stop-gradient terms give the same per-branch gradients. Writing $D(u, v) = \|u - v\|_2^2/(Hd_a)$, we have

$$\nabla_{\phi_1}\big[\beta D(u_1, u_2)\big] = \nabla_{\phi_1}\big[\beta D(u_1, \mathrm{sg}[u_2]) + \beta D(u_2, \mathrm{sg}[u_1])\big], \tag{B.3}$$

and analogously for ϕ_2. This is an equivalence of gradients for matched forward passes, not an equivalence of the reported stochastic training procedures.

Mutual uses a shared minibatch and one forward pass per adapter, reusing both outputs for behavior cloning and consistency. CRC uses separate behavior-cloning minibatches and an additional branch-2 forward pass on $\mathcal{B}_1$. The reported Shared batch control sets $\mathcal{B}_2 = \mathcal{B}_1$ but retains this additional forward pass: its branch-2 outputs for behavior cloning and consistency are computed separately and can use different dropout masks. Shared batch therefore uses three forward passes per update, whereas Mutual uses two. Because dropout draws from a shared global random number generator, the extra forward pass also changes random-number consumption. The similar functional results of Shared batch and Mutual (CFD/CDA: 0.02457/0.9225 and 0.02446/0.9232, respectively) do not establish identical optimization trajectories.

C Data, Evaluation, and Metrics

C.1 Data and Evaluation Protocol

All experiments use the MimicGen low-dimensional split with 700 training, 100 validation, and 200 held-out demonstrations per task using split seed 0 (Table C.1). Training the original policy and further training use only the training split. Observation windows repeat the first observation near the start of an episode, and target chunks repeat the final action near its end.

Observations and normalization. Inputs concatenate a fixed ordering of the low-dimensional observation keys `object`, `robot0_eef_pos`, `robot0_eef_quat`, `robot0_eef_vel_ang`, `robot0_eef_vel_lin`, `robot0_gripper_qpos`, `robot0_gripper_qvel`, `robot0_joint_pos`, `robot0_joint_pos_cos`, `robot0_joint_pos_sin`, and `robot0_joint_vel`. No image, depth, or trajectory-progress input is used. Observations and action targets are z-score normalized using training-split statistics; standard deviations below 10^{-6} are replaced by 1.

Tasks. We use the D0 variants of eight MimicGen tasks (Mandlekar et al., 2023), simulated in robosuite (Zhu et al., 2020), each with a maximum episode length of 1000 steps. The goals are as follows. *Kitchen*: turn on the stove, place the pot on the stove, place bread in the pot, move the pot to the serving region, and turn off the stove. *Stack Three*: stack three cubes in order. *Hammer Cleanup*: put a hammer in a drawer and close the drawer. *Stack*: stack one cube on another. *Threading*: insert a needle through the ring of a tripod. *Three-Piece Assembly*: assemble two pieces onto a base in order. *Square*: place a square nut around its peg. *Mug Cleanup*: put a mug upright in a drawer and close the drawer.

Table C.1: **Data usage.**

Stage	Training split	Validation split	Held-out demonstrations
Training the original policy	gradient updates	monitoring only	not used
Further training	gradient updates	not used	not used
Functional evaluation	not used	fixed input set $\mathcal{X}$	not used
Closed-loop evaluation	not used	not used	not replayed

Runs. Each method is trained with three runs (seeds 24, 29, 30); the five-run extension for Std. and CRC adds seeds 37 and 38. For CRC, each run contributes its deployed branch, and the other branch is not counted as a run. All runs of all methods start from the same original policy of their task.

Inputs for the functional metrics. For each task, a fixed set $\mathcal{X}$ of $N = 2048$ observation contexts is selected deterministically from the validation demonstrations using linearly spaced indices and shared by all methods and runs. Outputs are computed with dropout disabled in normalized action-output space, before de-normalization, temporal averaging, and clipping, using the full 35-dimensional action chunk.

Closed-loop evaluation. Closed-loop evaluation resets the simulator and rolls out each policy's own actions; held-out demonstrations are not replayed. Within each task and evaluation metric, fixed initial-state manifests are shared by all methods and runs. Success rates use 200 initial states per run on every task. Action MSE and EE Fréchet use 200 initial states on Hammer Cleanup, Stack, Threading, and Square, and 50 on Kitchen, Stack Three, Three-Piece Assembly, and Mug Cleanup. For the four tasks evaluated on 50 initial states, the trajectory-metric manifest is generated separately and is disjoint from the 200-state success manifest. On Square, using the seed-24 checkpoints of Std. and CRC and the first 20 evaluation initial states, repeated rollouts with the trajectory-evaluation code produced exactly identical action and end-effector position arrays within a process, across separate processes, and relative to the stored evaluation trajectories. This check is limited to these checkpoints and initial states.

C.2 Functional Metrics

CFD and CDA (Eq. 7) average over all unordered pairs of runs and all inputs in $\mathcal{X}$, with $Hd_a = 35$ in the normalization. For CFD, the root-mean-square difference over the 35 output dimensions is computed separately for each input and run pair, before averaging. CFD, CDA, and $\bar{\Delta}$ are all computed in normalized action-output space. For each run pair, inputs at which either functional-change vector has norm below 10^{-8} are excluded from both the CDA sum and its denominator. No inputs were excluded in the reported eight-task evaluation. Because all runs share the original policy, $\Delta f_s(x) - \Delta f_{s'}(x) = f_{\theta_s}(x) - f_{\theta_{s'}}(x)$, so CFD compares the outputs of the runs directly.

Size of the change. The average size of the functional change, used in Section 4.3 and Appendix E.3, is

$$\bar{\Delta} = \frac{1}{SN} \sum_{s=1}^{S} \sum_{x \in \mathcal{X}} \sqrt{\frac{1}{Hd_a} \|\Delta f_s(x)\|_2^2}. \tag{C.1}$$

It measures how far the runs move from the original policy and is not a consistency metric.

C.3 Closed-Loop Metrics

For runs r and r' and an initial state z, let $a_r(z,t) \in \mathbb{R}^{d_a}$ be the action executed at step t, after temporal averaging, de-normalization, and clipping, and let $T_r(z)$ be the episode length. Action MSE compares the executed actions at matching time steps up to the shorter episode,

$$\mathrm{MSE}_{r,r'}(z) = \frac{1}{T} \sum_{t=1}^{T} \big\| a_r(z,t) - a_{r'}(z,t) \big\|_2^2, \qquad T = \min\big(T_r(z), T_{r'}(z)\big), \tag{C.2}$$

without normalizing by the action dimension. EE Fréchet is the discrete Fréchet distance (Eiter & Mannila, 1994) between the recorded end-effector position trajectories $\big(p_r(z,t)\big)_{t=1}^{T_r(z)} \subset \mathbb{R}^3$ of the two runs, with Euclidean distance between points and without truncation, padding, or resampling. End-effector positions are measured in meters in the MuJoCo world frame. Both successful and failed episodes are included. Episodes terminate upon success, an environment termination signal, the evaluation horizon, or a non-finite-value safeguard. Both metrics are averaged over initial states and pairs of runs within each task.

C.4 Aggregation and Statistics

All metrics are computed per task and then averaged over the eight tasks with equal weight (Average). Comparisons across tasks use the task as the unit: when all eight tasks favor one method, a two-sided exact Wilcoxon signed-rank test (Wilcoxon, 1945) over tasks gives $p = 2^{-7} \approx 0.0078$.

Because the metrics are related, we do not treat such tests as independent pieces of evidence. Functional metrics compare policies on identical inputs, whereas closed-loop metrics compare rollouts that start from the same initial state but visit different states over time; neither replaces task success, which is reported separately.

D Detailed Results

D.1 Summary

Table D.1 collects the averages of the main baselines and selected controls. Among the configurations listed in this table, CRC has the lowest CFD and EE Fréchet and the highest CDA, while Mutual and Batch-512 have slightly lower action MSE. Their average success rates are within two percentage points of Std. The full training-time L2 sweep is reported in Appendix E.4.

Table D.1: **Summary of main baselines and selected controls.** Task-level metrics use three training runs and are averaged over the eight tasks.

	CFD ↓	CDA ↑	Success (%) ↑	Action MSE ↓	EE Fréchet ↓
Std.	0.02886	0.9009	69.85	0.5117	0.1076
Drop.	0.02722	0.9100	70.75	0.4496	0.0954
Mutual	0.02446	0.9232	69.29	**0.4405**	0.0952
CRC	**0.02271**	**0.9329**	**71.48**	0.4482	**0.0929**
$\beta_{\text{cons}} = 0$	0.02865	0.9022	70.10	0.4891	0.0979
Shared batch	0.02457	0.9225	71.27	0.4656	0.0976
Batch-512	0.02738	0.9152	69.35	0.4464	0.0985
Shrinkage (matched, $\lambda = 0.8$)	0.02292	0.9017	68.44	0.4563	0.0966
L2 (matched, $c = 0.01$)	0.02340	0.9217	68.96	0.4613	0.0978

D.2 Taskwise Closed-Loop Results

Table D.2 reports the closed-loop metrics per task. CRC reduces both metrics relative to Std. on all eight tasks. Mutual reduces both on seven tasks, and Drop. reduces action MSE on six tasks and EE Fréchet on seven. The lowest value varies by task: CRC has the lowest EE Fréchet on five tasks and the lowest action MSE on four, Mutual the lowest action MSE on three, and Drop. both lowest values on Kitchen. Among these four methods, Mutual has the lowest average action MSE and CRC the lowest average EE Fréchet.

Table D.2: **Taskwise closed-loop variation.** Averaged over initial states and all pairs of three training runs; initial-state counts follow Appendix C. Best values within each task and metric are bolded.

	Action MSE ↓				EE Fréchet ↓			
Task	Std.	Drop.	Mutual	CRC	Std.	Drop.	Mutual	CRC
Kitchen	0.8161	**0.4657**	0.5894	0.7009	0.0988	**0.0628**	0.0745	0.0983
Stack Three	0.6262	0.4864	**0.4353**	0.4404	0.1923	0.1572	**0.1512**	0.1535
Hammer Cleanup	0.4043	0.3849	0.3017	**0.2853**	0.0218	0.0218	0.0197	**0.0196**
Stack	0.1582	0.1334	**0.1234**	0.1333	0.0507	0.0449	0.0483	**0.0434**
Threading	0.0262	0.0230	0.0216	**0.0092**	0.0297	0.0239	0.0292	**0.0211**
Three-Piece Assembly	0.9035	0.9370	**0.8429**	0.9008	0.2537	0.2443	0.2223	**0.2096**
Square	0.3121	0.3232	0.3026	**0.2830**	0.1047	0.1004	**0.0984**	0.0986
Mug Cleanup	0.8471	0.8433	0.9070	**0.8325**	0.1093	0.1076	0.1182	**0.0993**
Average	0.5117	0.4496	**0.4405**	0.4482	0.1076	0.0954	0.0952	**0.0929**

D.3 Five Runs

To check that the functional result does not depend on the particular three runs, we train two more runs of Std. and CRC (seeds 37 and 38) and recompute CFD and CDA from all five runs using all

ten unordered run pairs; the three- and five-run estimates are computed separately (Table D.3). With five runs, CRC improves both metrics over Std. on all eight tasks, and the averages stay close to their three-run values. The five-run success rates are 69.76% for Std. and 70.51% for CRC.

Table D.3: **Five runs.** CFD and CDA of Std. and CRC computed from five runs (seeds 24, 29, 30, 37, 38). The last row repeats the three-run averages.

	CFD ↓		CDA ↑	
Task	Std.	CRC	Std.	CRC
Kitchen	0.02704	**0.02078**	0.9101	**0.9384**
Stack Three	0.03477	**0.02710**	0.8781	**0.9231**
Hammer Cleanup	0.02389	**0.01932**	0.9138	**0.9396**
Stack	0.03338	**0.02611**	0.8768	**0.9124**
Threading	0.02368	**0.01818**	0.9082	**0.9400**
Three-Piece Assembly	0.03137	**0.02504**	0.9003	**0.9283**
Square	0.03006	**0.02435**	0.9068	**0.9338**
Mug Cleanup	0.02767	**0.02215**	0.9112	**0.9407**
Average (five runs)	0.02898	**0.02288**	0.9007	**0.9321**
Average (three runs)	0.02886	**0.02271**	0.9009	**0.9329**

D.4 Perturbed Inputs

Because CRC reduces the difference between branch outputs during training, one may ask whether its lower CFD holds only on the exact evaluation inputs. We add Gaussian noise with standard deviation $\sigma \in \{0.01, 0.05, 0.1\}$ to the normalized observations in $\mathcal{X}$ and recompute CFD without retraining (Table D.4). The reduction relative to Std. stays at about 21% and holds on all eight tasks at every noise level. This tests local robustness around the evaluation inputs, not generalization to unrelated states.

Table D.4: **Perturbed inputs.** CFD with Gaussian noise added to the normalized observations in $\mathcal{X}$, without retraining. The reduction is from Std. to CRC.

Noise	Std. CFD	CRC CFD	Reduction (%)	Tasks favoring CRC
None	0.02886	0.02271	21.33	8/8
$\sigma = 0.01$	0.02887	0.02271	21.33	8/8
$\sigma = 0.05$	0.02899	0.02281	21.32	8/8
$\sigma = 0.10$	0.02931	0.02306	21.34	8/8

E Controls

E.1 Two-Branch Controls

$\beta_{\text{cons}} = 0$ keeps the two branches, their separate initializations, and their independent minibatches, but removes the consistency term. Shared batch keeps the consistency term but feeds both branches the same minibatch at every step, removing the difference in minibatch sampling. It retains the separate branch-2 forward passes for behavior cloning and consistency, as described in Appendix B.1. Table E.1 reports taskwise results. $\beta_{\text{cons}} = 0$ leaves CFD and CDA close to Std. on average, with mixed taskwise differences. Shared batch improves both metrics over Std. on all eight tasks, and CRC improves them further on all eight tasks.

E.2 Samples per Step

CRC draws two independent minibatches of 256 per step, giving 512 sampled training examples in total, compared with 256 for Std.; sampling with replacement allows repeated examples. To separate this from the coupling, we train standard further training with a batch size of 512, keeping the original policy, LoRA rank, number of steps, optimizer, learning rate, and seeds unchanged.

Table E.1: **Taskwise results of the two-branch controls.** Shared denotes Shared batch. Functional metrics average over evaluation inputs and all pairs of three runs; success averages over the three runs, using 200 initial states per run. Best values within each task and metric are bolded.

	CFD ↓				CDA ↑				Success (%) ↑			
Task	Std.	β_{cons}=0	Shared	CRC	Std.	β_{cons}=0	Shared	CRC	Std.	β_{cons}=0	Shared	CRC
Kitchen	0.02696	0.02689	0.02296	**0.02069**	0.9100	0.9091	0.9263	**0.9384**	80.50	80.33	83.00	**84.00**
Stack Three	0.03485	0.03411	0.02860	**0.02701**	0.8728	0.8798	0.9114	**0.9209**	48.67	47.33	**51.67**	51.17
Hammer Cleanup	0.02429	0.02430	0.02135	**0.01929**	0.9121	0.9134	0.9273	**0.9397**	**99.67**	99.33	98.67	98.83
Stack	0.03252	0.03256	0.02768	**0.02589**	0.8823	0.8802	0.9032	**0.9141**	95.33	96.00	**96.67**	95.50
Threading	0.02402	0.02290	0.01955	**0.01786**	0.9082	0.9138	0.9331	**0.9439**	95.67	95.50	97.83	**98.50**
Three-Piece Assembly	0.03113	0.03175	0.02716	**0.02516**	0.9010	0.8972	0.9172	**0.9282**	6.00	6.00	6.50	**8.17**
Square	0.02969	0.02944	0.02563	**0.02357**	0.9093	0.9107	0.9284	**0.9381**	68.33	**70.17**	69.17	68.50
Mug Cleanup	0.02747	0.02724	0.02361	**0.02219**	0.9115	0.9135	0.9332	**0.9403**	64.67	66.17	66.67	**67.17**
Average	0.02886	0.02865	0.02457	**0.02271**	0.9009	0.9022	0.9225	**0.9329**	69.85	70.10	71.27	**71.48**

Table E.2 compares it with standard further training at batch size 256, Shared batch, and CRC. The larger batch improves CFD and CDA over Std. on all eight tasks but remains below CRC on all eight tasks; averaged over tasks, it recovers 24% of the CFD reduction and 45% of the CDA increase from Std. to CRC. Shared batch, which couples two branches on a single minibatch of 256 and thus sees no additional samples, outperforms the larger batch on all eight tasks in both metrics. In closed-loop evaluation, Batch-512 reaches an action MSE similar to that of CRC (0.4464 versus 0.4482) but a higher EE Fréchet (0.0985 versus 0.0929), and its success rate (69.35%) stays close to that of Std.

Table E.2: **Samples per step.** Std. (512) is standard further training with twice the batch size; Shared denotes Shared batch. Averaged over evaluation inputs and all pairs of three training runs. Best values within each task and metric are bolded.

	CFD ↓				CDA ↑			
Task	Std.	Std. (512)	Shared	CRC	Std.	Std. (512)	Shared	CRC
Kitchen	0.02696	0.02505	0.02296	**0.02069**	0.9100	0.9251	0.9263	**0.9384**
Stack Three	0.03485	0.03261	0.02860	**0.02701**	0.8728	0.8995	0.9114	**0.9209**
Hammer Cleanup	0.02429	0.02325	0.02135	**0.01929**	0.9121	0.9248	0.9273	**0.9397**
Stack	0.03252	0.03236	0.02768	**0.02589**	0.8823	0.8886	0.9032	**0.9141**
Threading	0.02402	0.02231	0.01955	**0.01786**	0.9082	0.9236	0.9331	**0.9439**
Three-Piece Assembly	0.03113	0.02931	0.02716	**0.02516**	0.9010	0.9129	0.9172	**0.9282**
Square	0.02969	0.02790	0.02563	**0.02357**	0.9093	0.9242	0.9284	**0.9381**
Mug Cleanup	0.02747	0.02623	0.02361	**0.02219**	0.9115	0.9231	0.9332	**0.9403**
Average	0.02886	0.02738	0.02457	**0.02271**	0.9009	0.9152	0.9225	**0.9329**

E.3 SHRINKAGE

Shrinkage scales the learned update of each completed Std. run after training,

$$W_\lambda = W_0 + \lambda\,(W_{\text{Std}} - W_0), \qquad \lambda \in \{1.0, 0.9, \ldots, 0.5\}, \tag{E.1}$$

which in the LoRA parameterization scales the update by λ and requires no retraining; this uses the same form of weight-space interpolation between original and fine-tuned weights as WiSE-FT (Wortsman et al., 2022). Because the policy is nonlinear, this does not scale the functional change exactly, but it directly controls the size of the update. A single λ is chosen for all tasks and runs as the value whose average CFD on $\mathcal{X}$ is closest to that of CRC, which gives $\lambda = 0.8$; the choice uses neither taskwise tuning nor closed-loop results.

Table E.3 shows that shrinkage lowers CFD in proportion to the size of the change while CDA barely moves. At $\lambda = 0.8$, where CFD approximately matches that of CRC (0.02292 versus 0.02271), the average size of the change (Eq. C.1) is 20% smaller than for Std. (0.05687 versus 0.07127), whereas CRC reduces it by only 4% (0.06827). Across the whole range, shrinkage recovers at most 4.3% of the CDA improvement from Std. to CRC, and the ratio between CFD and the size of the change stays between 0.401 and 0.404, compared with 0.332 for CRC. At $\lambda = 0.8$, task success is 68.44%, and action MSE and EE Fréchet are 0.4563 and 0.0966.

Table E.3: **Shrinkage.** Averages over the eight tasks on $\mathcal{X}$. $\bar{\Delta}$ is the average size of the functional change (Eq. C.1). The ratio is computed within each task before averaging across the eight tasks, rather than by dividing the two reported averages.

λ	CFD ↓	CDA ↑	$\bar{\Delta}$	CFD / $\bar{\Delta}$ ↓
1.0 (Std.)	0.02886	0.9009	0.07127	0.4041
0.9	0.02587	0.9014	0.06406	0.4029
0.8 (matched)	0.02292	0.9017	0.05687	0.4020
0.7	0.02000	0.9020	0.04971	0.4013
0.6	0.01710	0.9022	0.04256	0.4008
0.5	0.01423	0.9023	0.03543	0.4006
CRC	0.02271	0.9329	0.06827	0.3316

E.4 Training-Time Proximity Regularization

To test whether run-to-run variation can also be reduced by constraining further training toward the original policy, we add an L2 penalty to standard further training. For each LoRA-adapted matrix, the effective weight change is

$$\Delta W_k = \frac{1}{r} B_k A_k,$$

and the regularizer is

$$\mathcal{R}_{\mathrm{L2}} = \sum_{k \in \mathcal{K}} \|\Delta W_k\|_F^2 ,$$

where $\mathcal{K}$ contains the eight matrices adapted by LoRA. The training objective is

$$\mathcal{L}_{\mathrm{L2}} = \mathcal{L}_{\mathrm{BC}} + \mu \, \mathcal{R}_{\mathrm{L2}}.$$

Thus, the penalty acts on the effective LoRA weight update rather than on the LoRA factors separately or on policy outputs. No additional dimension normalization is applied to the summed penalty. This term is separate from AdamW weight decay, which remains 10^{-5} as in Std. All other training settings are unchanged from standard further training.

The full eight-task sweep uses $c \in \{0.003, 0.01, 0.03, 0.1, 0.3\}$, with $\mu = 0.0488563\, c$. The scale factor 0.0488563 was computed once as $\mathcal{L}_{\mathrm{BC}}^{\mathrm{Std}} / \mathcal{R}_{\mathrm{L2}}^{\mathrm{Std}}$ on the Square Std. checkpoint (seed 24), where $\mathcal{L}_{\mathrm{BC}}^{\mathrm{Std}}$ is the behavior-cloning loss over the full training set and $\mathcal{R}_{\mathrm{L2}}^{\mathrm{Std}}$ is the L2 regularizer defined above. The resulting scale factor is fixed across all tasks and runs.

Table E.4: **Training-time L2 regularization.** Averages over the eight tasks on $\mathcal{X}$. $\bar{\Delta}$ is the average size of the functional change (Eq. C.1). The ratio is computed within each task before averaging across the eight tasks, rather than by dividing the two reported averages.

c	CFD ↓	CDA ↑	$\bar{\Delta}$	CFD / $\bar{\Delta}$ ↓
0 (Std.)	0.02886	0.9009	0.07127	0.4041
0.003	0.02633	0.9108	0.06865	0.3824
0.01 (matched)	0.02340	0.9217	0.06520	0.3577
0.03	0.01965	0.9344	0.05986	0.3272
0.1	0.01503	0.9477	0.05121	0.2927
0.3	0.01091	0.9573	0.04100	0.2653
CRC	0.02271	0.9329	0.06827	0.3316

For the matched comparison in the main paper, we select the tested coefficient whose eight-task average CFD is closest to CRC. This gives $c = 0.01$, with CFD 0.02340, CDA 0.9217, and average

functional-change magnitude $\bar{\Delta} = 0.06520$, compared with 0.02271, 0.9329, and 0.06827 for CRC. The corresponding Action MSE, EE Fréchet, and success are 0.4613, 0.0978, and 68.96%, respectively. At similar CFD, training-time proximity therefore produces substantially higher directional alignment than Std. and post-hoc shrinkage, but remains below CRC in CDA while reducing the average functional-change magnitude more strongly.

A stronger setting, $c = 0.03$, gives CFD 0.01965, CDA 0.9344, and $\bar{\Delta} = 0.05986$. Its Action MSE, EE Fréchet, and success are 0.4125, 0.0916, and 68.94%. This operating point attains lower average CFD and higher average CDA than CRC, while suppressing the average functional change more strongly. Taskwise, its CDA exceeds CRC on six of the eight tasks rather than all eight. Stronger L2 settings therefore provide an important boundary condition: sufficiently strong proximity regularization can further reduce run-to-run functional variation, but do so together with a larger reduction in the magnitude of further-training changes.

This control differs from the shrinkage analysis in Appendix E.3. Shrinkage rescales a completed Std. update after training, whereas the L2 penalty modifies the training objective itself. Consequently, the near-constant directional alignment observed under post-hoc shrinkage does not imply that training-time proximity regularization cannot improve CDA. The main paper uses the tested L2 point with CFD closest to CRC as the matched diagnostic comparison, while the full sweep reports the behavior at stronger and weaker proximity strengths.

E.5 Closed-Loop Results of the Controls

Table 3 includes the closed-loop metrics of $\beta_{\text{cons}} = 0$, Shared batch, Batch-512, shrinkage, and L2. Action MSE and EE Fréchet measure only the size of the differences between runs; they can decrease when functional changes become smaller, as observed with shrinkage, and they also depend on how rollouts diverge over time, so small differences between the controls are hard to interpret. All five controls reduce both metrics relative to Std. They reduce them less than CRC, except that Batch-512 reaches a similar action MSE (0.4464 versus 0.4482) while its EE Fréchet remains higher (0.0985 versus 0.0929). Whether the changes of different runs are smaller or better aligned is distinguished by CDA at the function level (Section 4.3), where $\beta_{\text{cons}} = 0$ and shrinkage stay at the level of Std. and the other controls remain below CRC.