DISCODEMO: DISCOVERING EFFICIENT AND DIVERSE ROBOT DEMONSTRATIONS FOR IMITATION LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Behavior cloning for robotic manipulation is highly sensitive to the coverage of its training data, yet collecting diverse successful demonstrations remains costly. We study reinforcement learning (RL) not as a deployable controller, but as a data generator for downstream visuomotor imitation learning. Our key premise is that multiple successful solution modes cover a broader range of successful trajectories than repeatedly collecting a single narrow solution. We introduce DISCODEMO, which trains a privileged latent-conditioned RL generator with a parallel reverse curriculum and a success-gated diversity objective to discover diverse solutions while preserving task success. The trained generator produces multiple successful closed-loop behaviors, which are safety-filtered and exported as a static dataset for downstream imitation learning. Across four manipulation tasks, DISCODEMO produces broader successful-state coverage and improves downstream visuomotor policies in both simulation and the real world. Averaged across tasks, downstream success increases to 62% from 40% for the strongest baseline in simulation, and to 78% from 39% on the real robot, respectively. These results suggest that successful solution diversity improves imitation learning for a pretrained flow-matching VLA and remains complementary to corrective data collection, highlighting RL as an effective mechanism for constructing broader imitation-learning datasets.

1 INTRODUCTION

Modern visuomotor policies are commonly trained from robot trajectories using behavior cloning (BC) (Pomerleau, 1988). Scaling this paradigm, however, is constrained by the cost of physical data collection. Robot trajectories are typically collected through human teleoperation, requiring access to physical robot hardware and operator time for every episode. Consequently, even large robot datasets contain only on the order of a million episodes (O'Neill et al., 2024; Khazatsky et al., 2024), far fewer than the billions of image–text pairs (Schuhmann et al., 2022) or trillions of tokens (Grattafiori et al., 2024) used to train foundation models in vision and language.

Simulation offers a scalable alternative. Robot trajectories can be generated cheaply and in parallel without requiring human teleoperation for every episode, and advances in sim-to-real transfer have made simulated experience increasingly useful for training real-world policies (Torne et al., 2024; Bauza et al., 2024). Scaling simulated data, however, raises two key questions: *how should successful trajectories be generated, and what distribution of successful behavior should they cover?* Together, these choices determine the supervision available to downstream BC.

Existing simulation pipelines generate successful trajectories in several ways. Motion-planning methods construct feasible motions between task-relevant poses (Chen et al., 2025a;b; Tian et al., 2026), but rely on task-specific planning structure. MimicGen-style methods reduce this burden by retargeting successful demonstration segments to new scene configurations (Mandlekar et al., 2023; Garrett et al., 2024; Jiang et al., 2025), but remain constrained by the behaviors contained in the source demonstrations and by where those behaviors can be successfully retargeted. These limitations are particularly relevant in contact-rich manipulation, where geometric transformation alone may not preserve successful interactions as the scene changes. Reinforcement learning (RL) provides a complementary approach by learning closed-loop behavior directly from task feedback, and recent methods bootstrap RL from demonstrations or pretrained behavior priors to make successful data generation practical under sparse rewards (Tao et al., 2024; Xiao et al., 2026; Xu et al., 2026).

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.

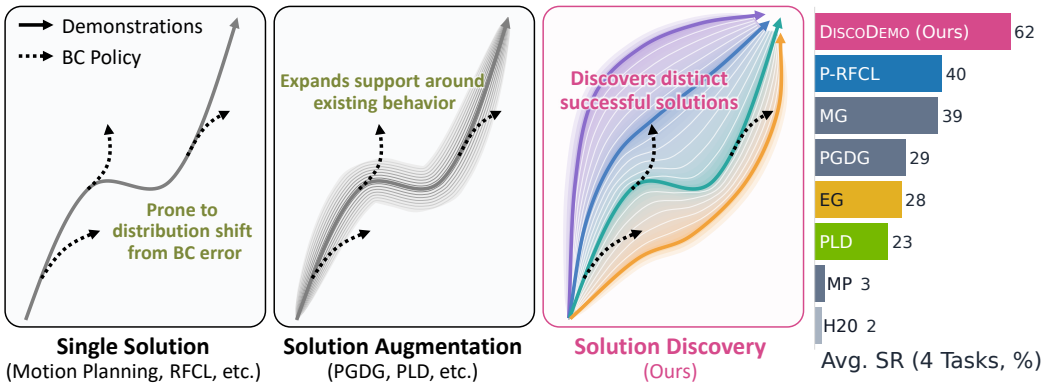


Figure 1: **Solution discovery broadens coverage.** Single-solution data provide narrow coverage, and augmentation mainly expands around existing trajectories. DISCODEMO, our method instead discovers distinct successful solutions that broaden state-space coverage.

However, an RL policy that reliably solves the task does not necessarily generate a training distribution well suited to downstream BC. As illustrated in Fig. 1, task reward alone can concentrate an RL policy around a narrow set of successful solutions, so collecting more rollouts need not substantially expand the states represented in the dataset (Kumar et al., 2020). This limited coverage can make BC brittle when prediction errors move the learned policy beyond states supported by its training demonstrations (Ross et al., 2011). Perturbation- and recovery-based methods broaden supervision around an existing demonstration or policy (Dai et al., 2026; Xiao et al., 2026), but their additional experience remains anchored to the supplied behavior. This motivates a different question: can the generator itself discover multiple successful ways of solving the task?

We address this question with DISCODEMO, which trains an RL generator to combine task competence with solution diversity. Starting from a few successful demonstrations, we use a parallel reverse curriculum to bootstrap task-solving behavior under sparse rewards, using demonstration states for curriculum resets rather than treating demonstration actions as the target solution. At the same time, we condition the generator on a latent skill and optimize a success-gated diversity objective that encourages different skills to realize distinct state-space behaviors while preserving task completion (Kumar et al., 2020; Park et al., 2024). The resulting generator discovers multiple successful solution modes whose union provides broader successful-state coverage for downstream BC.

We evaluate DISCODEMO on four real-world manipulation tasks reconstructed in simulation, spanning pick-and-place, stacking, and precision-sensitive peg insertion. For each data source, we train the same $\pi_{0.5}$ vision-language-action policy (Intelligence et al., 2025), pretrained on DROID (Khazatsky et al., 2024), and compare against human demonstrations and simulation data-generation methods based on planning (Fu et al., 2026), augmentation (Mandlekar et al., 2023; Dai et al., 2026), and RL (Xiao et al., 2026; Xu et al., 2026; Tao et al., 2024). DISCODEMO produces the strongest downstream policy across all four tasks in both simulation and the real world. Averaged across tasks, it reaches 62% success in simulation compared with 40% for the strongest baseline, and 78% compared with 39% on the real robot. The largest gains over the strongest baseline are 23 percentage points on FMB-Round in simulation and 45 on PnP-Banana in the real world.

2 RELATED WORK

Simulation Data Generation for Behavior Cloning. A large body of work scales robot data by reusing or transforming successful behavior. Motion-planning pipelines construct feasible trajectories from task structure (Chen et al., 2025a;b; Tian et al., 2026), while MimicGen-style methods transfer demonstrations across new scene configurations (Mandlekar et al., 2023; Garrett et al., 2024; Jiang et al., 2025). DAgger (Ross et al., 2011), DART (Laskey et al., 2017), and more recent intervention- and perturbation-based methods (Hoque et al., 2024; Dai et al., 2026) further expand coverage around learner errors and recovery states. These approaches broaden scene, state, and recovery coverage from existing demonstrations or behaviors. Our work follows a complementary line of RL-based data generation, training the generator itself to discover multiple successful modes.

RL as a Data Generator. RL provides another route to simulation data generation by learning the action-generating policy from privileged state and task rewards. RialTo (Torne et al., 2024) uses

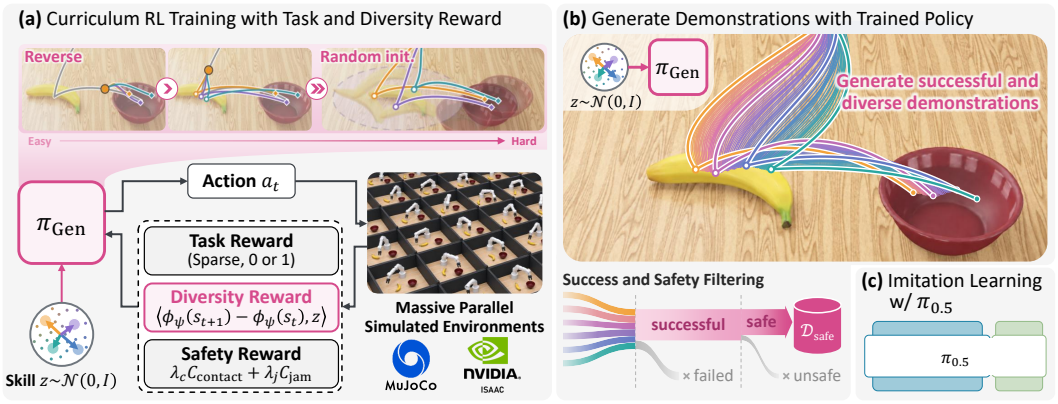


Figure 2: **Overview of DISCODEMO.** (a) We train a privileged generator with a parallel reverse curriculum, sparse task reward, success-gated diversity, and safety shaping. (b) The trained generator produces successful and diverse demonstrations, filtered for physically undesirable interactions. (c) The retained observation–action data train a downstream visuomotor policy by imitation learning.

simulation RL for visuomotor learning, whereas RLDG (Xu et al., 2025) uses real-world RL rollouts to fine-tune generalist policies. RFCL (Tao et al., 2024) and the contemporaneous DemoStart (Bauza et al., 2024) use closely related demonstration-driven curricula to bootstrap sparse-reward RL from successful states; DemoStart additionally distills the resulting privileged policy into a vision-based student. PLD (Xiao et al., 2026) and ExpertGen (Xu et al., 2026) connect RL data generation with pretrained downstream policies or behavior priors. DLR (Yang et al., 2025) instead discovers discrete behavioral patterns from successful demonstrations and reinforces them to construct diverse data for VLA pretraining. Our work similarly studies the distribution produced by an RL generator, but discovers diverse successful solutions online with a latent-conditioned, success-gated objective.

Diverse RL for Solution Discovery. Unsupervised skill discovery learns latent-conditioned policies with diverse behaviors without task-specific rewards (Eysenbach et al., 2018; Sharma et al., 2019; Laskin et al., 2022; Park et al., 2022; Kim et al., 2023; Park et al., 2024). METRA (Park et al., 2024) encourages latent skills to induce distinct state-space behaviors through a metric-aware representation, while task-aware methods such as SMERL (Kumar et al., 2020) seek diversity without sacrificing task performance. Our goal is not to learn a diverse repertoire for deployment, but to combine such diversity with trajectory-level success gating to broaden the range of successful trajectories available to downstream behavior cloning.

3 METHOD

As illustrated in Fig. 2, we first train a latent-conditioned privileged policy as a demonstration generator, using curriculum RL with a task-aware diversity objective (Section 3.2). We then freeze the generator and use it to produce diverse successful demonstrations, while filtering physically undesirable behaviors (Section 3.3). Finally, we use these filtered demonstrations to train a downstream visuomotor policy by behavior cloning, without privileged information (Section 3.4).

3.1 PROBLEM SETUP

We assume access to a simulator with privileged state information and resets to a specified state $s \in \mathcal{S}$. The downstream visuomotor policy instead observes only $o = h(s)$, consisting of visual observations and robot proprioception. For each task, we also assume a small set of human demonstrations $\mathcal{D}_{\text{human}} = \{\tau_i\}_{i=1}^{N}$, where $\tau_i = \{(s_t, a_t)\}_{t=0}^{H_i-1}$. We use these demonstrations as offline RL experience and their states as curriculum reset candidates.

We model the underlying task as a finite-horizon Markov decision process (MDP), $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \rho_0, \gamma\}$, with horizon H. At each time step t, the data-generation policy $\pi_{\text{Gen}}(a_t|s_t; \theta_{\text{Gen}})$ selects a continuous action $a_t \in \mathcal{A}$, and the environment transitions to $s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)$. The initial state is sampled as $s_0 \sim \rho_0$, and π_{Gen} is trained with a sparse task reward $r(s_t, a_t)$ to maximize $\mathbb{E}_{\pi_{\text{Gen}}}[\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t)]$. We denote the downstream visuomotor policy by $\pi_{\text{IL}}(a_t|o_t; \theta_{\text{IL}})$, which is trained only from the exported observation–action dataset.

3.2 Training a Task-Competent and Diverse Generator

Efficient Learning with a Reverse Curriculum. With sparse task rewards, a policy starting from the original initial-state distribution may rarely encounter a success signal. Following demonstration-based curricula (Tao et al., 2024; Bauza et al., 2024), we initially reset episodes to states near the end of successful demonstrations, so the policy only needs to complete the final part of the task. As its success rate improves, we progressively move the reset boundary earlier in the demonstration, requiring the policy to solve increasingly longer portions of the task.

We train π_{Gen} with FlashSAC (Kim et al., 2026a) using a massively parallel reverse curriculum. For each demonstration τ_i, we assign a group of parallel environments and maintain an independent curriculum boundary c_i, initialized near the end of the trajectory. Episodes within each group are reset to states sampled from the demonstration suffix beginning at c_i. When the success rate $\hat{p}_i$ over episodes started at demonstration i's boundary reaches a threshold p, we move that boundary backward by Δ_c:

$$c_i \leftarrow \begin{cases} \max(0, c_i - \Delta_c), & \hat{p}_i \geq p, \\ c_i, & \text{otherwise.} \end{cases} \tag{1}$$

Once half of the demonstrations are solved at $c_i = 0$, all environments switch to the original task distribution ρ_0. Thus, multiple demonstration-specific curricula advance concurrently from batched rollouts. Details are provided in Section B.

Discovering Multiple Successful Solutions. The curriculum helps the generator learn to solve the task, but task competence alone does not ensure that its demonstrations are useful for training a downstream visuomotor policy. As illustrated in Fig. 1, under sparse rewards, once a reliable solution is discovered, the policy has little incentive to explore alternative solutions (Kumar et al., 2020), causing repeated rollouts to concentrate around a narrow state–action distribution.

To encourage multiple successful solutions, we augment the task reward with a diversity reward based on METRA (Park et al., 2024). We extend the generator to a latent-conditioned policy $\pi_{\text{Gen}}(a_t|s_t, z; \theta)$, where a latent skill $z \sim \mathcal{N}(0, I)$ is sampled at the start of each episode and held fixed throughout the rollout. Different values of z encourage the generator to realize distinct successful behaviors across rollouts. We jointly learn a state representation $\phi_\psi(s) \in \mathbb{R}^d$ from privileged simulator states, and reward state-space displacements that align with the sampled latent:

$$r_{\text{div}}(s_t, s_{t+1}, z) = \langle \phi_\psi(s_{t+1}) - \phi_\psi(s_t), z \rangle \,. \tag{2}$$

Unconstrained diversity can also encourage distinct failure modes rather than useful task solutions. Inspired by the structured-diversity principle of SMERL (Kumar et al., 2020), we therefore gate the diversity reward by trajectory success. Let $y(\tau) \in \{0, 1\}$ indicate whether trajectory τ completes the task. We optimize the combined reward

$$\tilde{r}_t = r_{\text{task}}(s_t, a_t) + \alpha \, y(\tau) \, r_{\text{div}}(s_t, s_{t+1}, z), \tag{3}$$

where α controls the strength of the diversity objective. Thus, the generator is rewarded for diversity only among trajectories that successfully complete the task. Without the gate ($y(\tau) \equiv 1$), the diversity reward dominates the sparse task reward and the generator fails to learn the task.

Safety Shaping. Even successful trajectories can involve undesirable physical interactions, such as unintended scene contact or excessive force during insertion. We therefore define a safety penalty

$$r_{\text{safe}}(s_t, a_t) = \lambda_{\text{c}} \, C_{\text{contact}}(s_t) + \lambda_{\text{j}} \, C_{\text{jam}}(s_t), \tag{4}$$

where C_{contact} penalizes undesirable robot–scene contact outside the intended finger–object interaction, and C_{jam}, used for insertion tasks, penalizes excessive force between the held object and the insertion surface. The generator is trained with the final reward $\tilde{r}_t - r_{\text{safe}}(s_t, a_t)$, which softly biases the successful and diverse solutions toward physically plausible interactions.

3.3 Generating Successful and Diverse Demonstrations

Latent-Conditioned Rollouts. With the generator trained as described above, we freeze π_{Gen} and collect trajectories from initial states sampled from the original task distribution ρ_0. For each rollout i, we independently sample a skill $z^{(i)} \sim \mathcal{N}(0, I)$ and hold it fixed until the episode ends. The rollout yields a trajectory $\tau^{(i)} = \{(s_t^{(i)}, a_t^{(i)})\}_{t=0}^{T_i - 1}$ of length T_i. By varying z across rollouts, we collect different task-solving behaviors from the same trained generator.

Safety Filtering and Dataset Export. Despite the safety penalty used during training (4), generated trajectories may still contain undesirable physical interactions. We therefore discard failed trajectories and filter successful demonstrations using the same safety signals employed during training. Specifically, we retain

$$\mathcal{T}_{\text{keep}} = \{\tau \mid y(\tau) = 1,\ \mathcal{F}_{\text{task}}(\tau) = 1\}\,, \tag{5}$$

where $\mathcal{F}_{\text{task}}$ checks contact magnitude and duration, plus insertion force for insertion tasks (Section B.4). For each retained trajectory, we render the downstream observation $o_t = h(s_t)$ and pair it with the commanded joint-position target after action processing and the gripper command. The resulting observation–action pairs form the static dataset $\mathcal{D}$ used for downstream imitation learning.

3.4 Downstream Imitation Learning

Given the generated dataset $\mathcal{D}$, we train a downstream visuomotor policy π_{IL} using standard imitation learning. Unlike the privileged generator π_{Gen}, π_{IL} takes visual observations and robot proprioception as input, without access to the privileged simulator state or latent skill. We optimize

$$\theta^{\star}_{\text{IL}} = \arg\min_{\theta_{\text{IL}}} \mathbb{E}_{(o,a)\sim\mathcal{D}} \left[\ell_{\text{IL}}\left(\pi_{\text{IL}}(\cdot|o;\theta_{\text{IL}}), a\right)\right], \tag{6}$$

where ℓ_{IL} denotes the imitation-learning objective of the downstream policy. π_{IL} is trained solely on the fixed dataset, without further interaction with the environment or the generator.

4 Experiments

We evaluate whether generating successful, diverse demonstrations improves downstream visuomotor imitation learning. We describe the experimental setup, then investigate the following questions:

RQ1: Does our generated data improve downstream imitation in simulation and on a real robot?

RQ2: How do the generated datasets differ in solution diversity and coverage, and how are these properties related to downstream performance?

RQ3: How does the diversity objective trade off task competence, solution diversity, and downstream imitation performance?

RQ4: How efficiently can diverse demonstrations be learned and generated, and are their benefits complementary to on-policy correction?

4.1 Experimental Setup

Tasks. We evaluate on four Franka Research 3 manipulation tasks reconstructed in Isaac Sim (NVIDIA). PnP-Banana places a banana into a bowl, StackCube stacks a red cube on a blue cube, and FMB-Round and FMB-SqCircle require inserting distinct peg geometries from the Functional Manipulation Benchmark (FMB) (Luo et al., 2024) into a fixture board. All tasks randomize the initial object poses, with episode horizons of 30 s for PnP-Banana and StackCube and 20 s for the two insertion tasks (details in Section A).

Data Sources and Baselines. We use the original 20 human demonstrations as H20, while all generated-data methods provide 3,000 successful demonstrations. We compare H20 with motion planning (MP) (Fu et al., 2026), MimicGen (MG) (Mandlekar et al., 2023), PGDG (Dai et al., 2026), PLD (Xiao et al., 2026), ExpertGen (EG) (Xu et al., 2026), P-RFCL, and DiscoDemo. For MP, we build a CaP-X-style scripted motion planner (Fu et al., 2026). MG and PGDG transform and perturb human demonstrations, respectively, whereas PLD, EG, P-RFCL, and DiscoDemo generate experience with RL. P-RFCL (DiscoDemo@α=0) is our parallelized RFCL (Tao et al., 2024) without diversity; PGDG reimplements its perturbation stage, while PLD and EG are reimplemented by state-based with FlashSAC (Section C). DiscoDemo uses a common diversity weight of $\alpha = 0.3$ for the main downstream comparison.

Downstream Training and Evaluation. For each generated-data method, we construct a dataset of 3,000 successful demonstrations from left-shoulder and wrist cameras, and fine-tune the same DROID-pretrained $\pi_{0.5}$ (Intelligence et al., 2025; Khazatsky et al., 2024) visuomotor policy with joint-space action control using an identical training recipe. We evaluate three seeds over 50 episodes each in simulation and the preselected first seed over 20 trials per task on the real robot, deployed zero-shot. We report success rate (SR) and mean completion time over successful episodes (Time). Further details are provided in Section B.

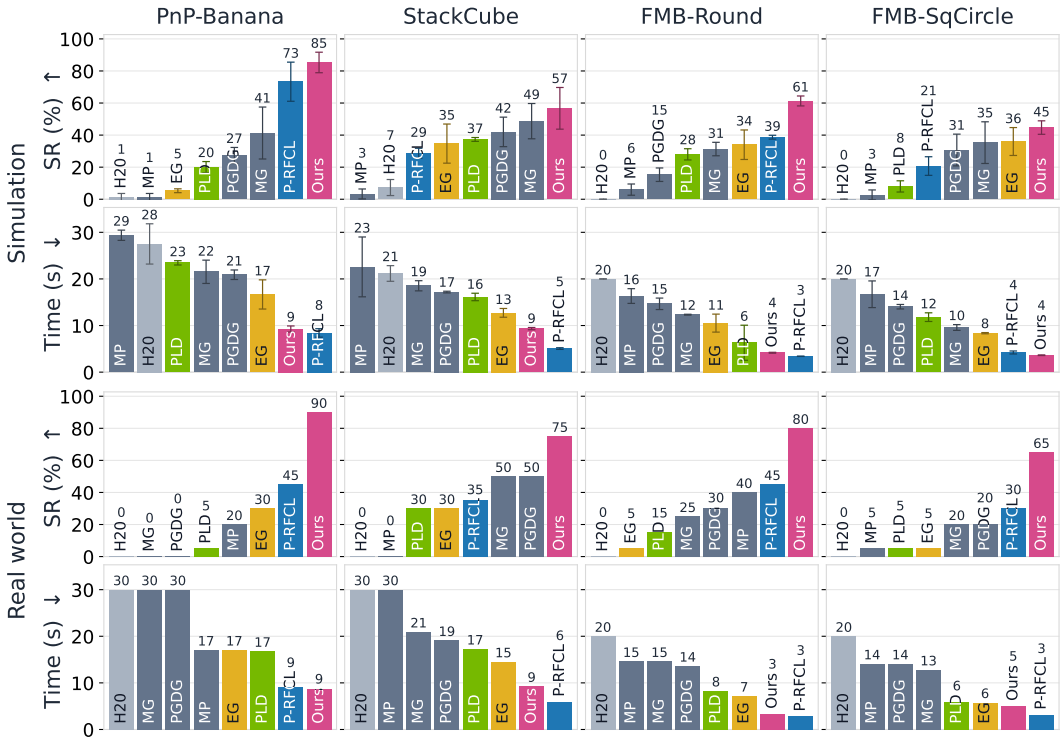


Figure 3: **Downstream imitation performance in simulation and the real world.** DISCODEMO achieves the highest success rate across all tasks with competitive completion times. Simulation: mean ± s.d. over three fine-tuning seeds.

4.2 DOWNSTREAM IMITATION PERFORMANCE

Fig. 3 compares downstream policies trained on each data source using the same visuomotor learner (Khazatsky et al., 2024; Intelligence et al., 2025). Across both simulation and real-world evaluation, the results reveal distinct trade-offs in the types of demonstrations produced by different data-generation strategies. H20 provides too little coverage from only a small set of human demonstrations, resulting in poor downstream success across the tasks. MP (Fu et al., 2026) substantially increases the amount of successful data, but remains relatively weak in our setting. This challenge may be amplified by the joint-space action formulation of our DROID-pretrained $\pi_{0.5}$ downstream policy, which makes nominal planned trajectories more sensitive to imitation errors.

MG (Mandlekar et al., 2023) and PGDG (Dai et al., 2026) are considerably stronger in terms of success rate. By transforming or perturbing human demonstrations, they preserve useful human behavior while expanding the range of training states. Their successful rollouts, however, are often substantially longer, suggesting that these methods also inherit suboptimal motions from the source demonstrations. For example, on PnP-Banana, the learned policy can take an unnecessarily circuitous path around the banana before grasping it. We provide qualitative rollout comparisons illustrating these behaviors in Section F.

P-RFCL exhibits the opposite behavior. Because it uses the same RL generator as DISCODEMO without the diversity objective, it often discovers a direct solution and, when successful, completes the task in the shortest time. However, concentrating on such a narrow solution mode leaves the downstream policy with limited support once its rollout deviates from that nominal behavior, making recovery difficult and reducing overall success. PLD (Xiao et al., 2026) and EG (Xu et al., 2026) similarly demonstrate the benefit of RL-generated data for learning efficient motions, producing substantially shorter successful episodes than the human-augmentation baselines, but their downstream success rates remain limited.

In contrast, DISCODEMO combines the two desirable properties. Its diverse RL generator retains the short, goal-directed behaviors characteristic of RL-generated demonstrations while covering multiple successful solution modes that provide broader support for downstream execution and recovery. Consequently, the same pattern holds in both simulation and on the real robot: DISCODEMO

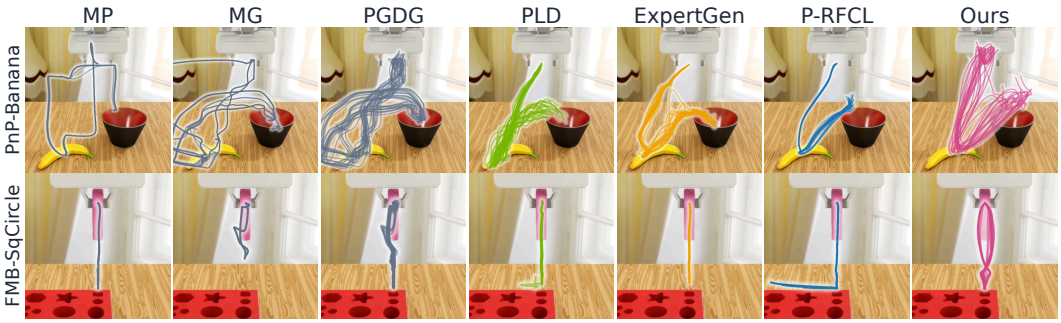


Figure 4: **Qualitative Results of Generated Demonstrations.** Each line traces the tool center point (TCP) of one successful demonstration from the same initial placement.

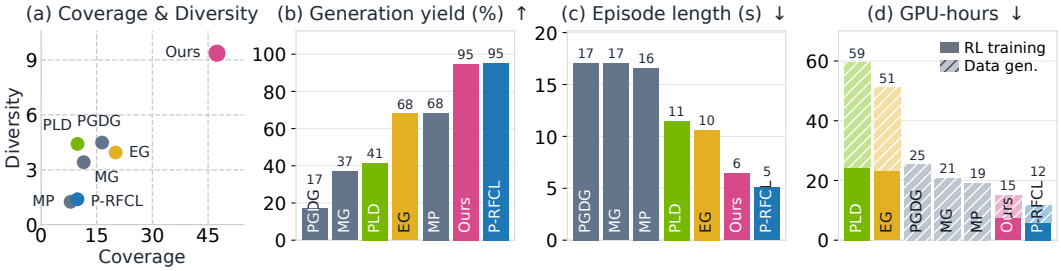


Figure 5: **Dataset analysis.** On average across tasks, DISCODEMO produces the most diverse and widest-coverage datasets, with high yield, short episodes, and low end-to-end GPU cost.

achieves the strongest success rates while maintaining completion times close to the most efficient RL-based baselines.

4.3 ANALYSIS OF GENERATED DEMONSTRATIONS

Qualitative Solution Diversity. To separate solution diversity from variation in initial conditions, we generate successful demonstrations from the same initial placement. As shown in Fig. 4, MP and P-RFCL repeatedly follow relatively narrow solution modes, whereas MG and PGDG mainly introduce variations around the source human behaviors. PLD and ExpertGen exhibit greater variation through their pretrained behavior priors, but their solutions remain influenced by behaviors represented in those priors. In contrast, DISCODEMO produces distinct successful trajectories across all four tasks (Fig. 4; StackCube and FMB-Round in Section F). This suggests that the diversity objective expands the set of successful solution strategies, rather than only perturbing existing trajectories.

Quantitative Diversity and Coverage. We quantify these differences in Fig. 5 (a) using the trajectory representation of PGDG (Dai et al., 2026). Each successful trajectory is represented by its object and end-effector motion, and similarity between trajectories is computed with an RBF kernel. We summarize each set using the Vendi score (Friedman & Dieng, 2023), which can be interpreted as the effective number of distinct trajectories (Section E.1). *Diversity* applies this metric to successful rollouts from one fixed initial scene, isolating variation in how the same task instance is solved. *Coverage* instead applies it to the full 3,000-demonstration dataset.

DISCODEMO achieves the highest diversity and coverage on average. Dataset-level coverage is more closely associated with downstream success than fixed-scene diversity in our analysis (Section E). Thus, visually distinct trajectories are not themselves the goal; multiple successful solutions are useful when they broaden the behaviors represented across the task distribution.

Generation Yield and Trajectory Efficiency. The remaining panels characterize whether this broader support can be produced reliably and efficiently. *Generation yield* (Fig. 5 (b)) measures how often a data-generation attempt produces a successful trajectory before export filtering. Yield alone does not determine dataset quality: scripted or transformation-based methods may simply reject many unsuccessful candidates, while for learned generators low yield can also indicate regions where the teacher has not acquired reliable task-solving behavior (Mandlekar et al., 2023). Nevertheless, low yield restricts where successful supervision can be collected, even when the final dataset size is matched. Despite optimizing for multiple solution modes, DISCODEMO maintains high generation yield, indicating that its diversity does not simply arise from unconstrained exploration or

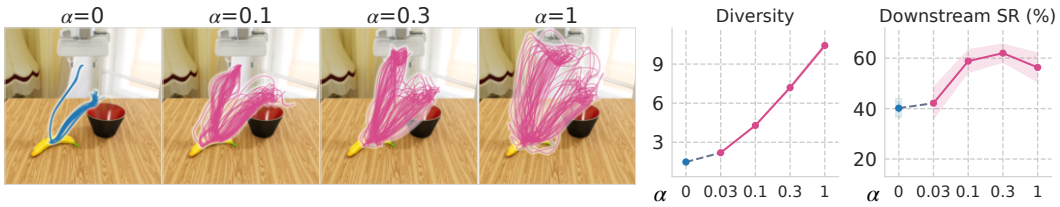


Figure 6: **Effect of diversity strength α.** Banana trajectory overlays and four-task averages diversity and SR. Diversity increases with α, while downstream success peaks at an intermediate value.

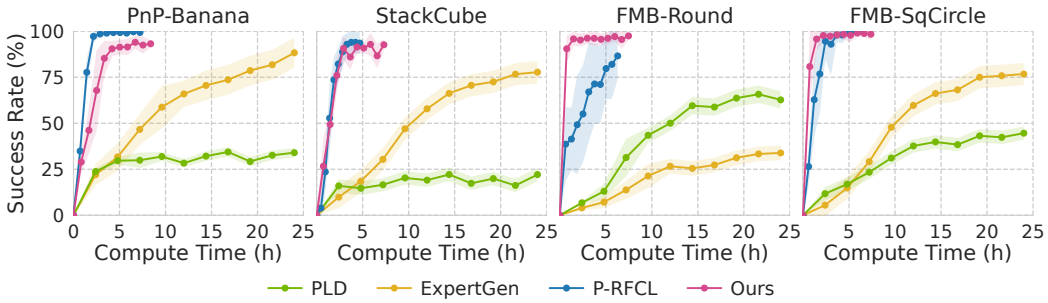


Figure 7: **Generator training efficiency.** P-RFCL and DISCODEMO complete training faster than PLD and ExpertGen. Times are estimated for the former and recorded for the latter (Section F.4).

additional failure modes. *Episode length* (Fig. 5 (c)) provides a complementary view of trajectory quality by measuring how directly successful rollouts reach the goal. RL-based generators generally produce shorter, more goal-directed trajectories than methods that transform human demonstrations. DISCODEMO preserves this property while increasing solution coverage, whereas P-RFCL shows why short trajectories alone are insufficient: it often produces efficient successes around a much narrower solution distribution.

End-to-End Data-Generation Cost. The GPU-hour panel (Fig. 5 (d)) measures the end-to-end compute required to obtain the final 3,000 demonstrations, including generator training when applicable and subsequent rollout and rendering. For a consistent comparison, all costs are normalized to a single process on one GPU. The hatched data-generation portion, however, consists of independent rollout and rendering jobs and can be parallelized across multiple processes in practice. Because P-RFCL and DISCODEMO use lightweight state-based policies that can be trained and rolled out in massively parallel simulation, the additional computation used for solution discovery remains modest relative to methods that repeatedly invoke large vision-based behavior priors.

Effect of Diversity Strength. We vary the diversity weight α to examine whether increasing behavioral diversity directly improves downstream imitation. As α increases, the generator produces increasingly distinct solution modes, confirming that the objective controls solution diversity. However, averaged over tasks, downstream success peaks at an intermediate value: when the diversity objective becomes too strong, further behavioral variation begins to come at the expense of reliable task completion, consistent with Luo et al. (2026). These results indicate that the desired property is not maximum diversity itself, but broad coverage by behaviors that remain successful. Per task, the peak ranges from $\alpha = 0.1$ to $\alpha = 1$ (FMB-Round), as shown in Section F.2.

4.4 GENERATOR TRAINING EFFICIENCY AND SAFETY CURATION

Generator Training Efficiency. Fig. 7 compares training time, since per-step computation differs across generators. PLD and ExpertGen rely on pretrained vision-based behavior priors and repeatedly invoke those models during RL, making each interaction comparatively expensive. P-RFCL and DISCODEMO instead train lightweight privileged-state policies with massively parallel simulation and therefore complete their training budgets substantially faster. The diversity objective adds representation learning and additional optimization, but does not introduce a systematic slowdown across tasks. In particular, on the insertion tasks, the broader exploration induced by the diversity objective can expose successful strategies early in training despite this additional computation. Thus, the main efficiency advantage comes from the parallel state-based generator, while solution discovery adds only a task-dependent optimization overhead.

Table 1: Effect of safety shaping and filtering (5). Illegal contact (IC) and container displacement (CD) measured on the exported demonstrations.

Generator		PnP-Banana		FMB-SqCircle	
Shaping	Filter	IC (N·s) ↓	CD (cm) ↓	IC (N·s) ↓	CD (cm) ↓
✗	✗	21.3	1.37	73.0	10.48
✓	✗	15.1	1.32	22.9	13.13
✗	✓	8.1	1.06	3.3	**2.20**
✓	✓	**6.8**	**1.04**	**1.2**	5.89

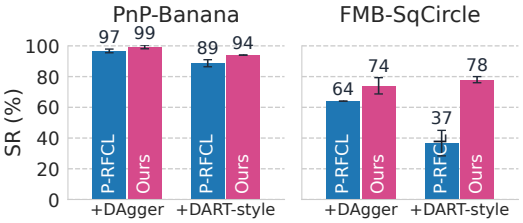


Figure 8: Downstream success rate after corrective data collection.

Safety Shaping and Filtering. We examine how physically undesirable successful behaviors are removed before export using illegal contact (IC) and container displacement (CD) in Tab. 1. IC measures unintended robot–scene contact, while CD measures motion of the bowl or fixture board during execution. Safety shaping and filtering act at different stages: shaping softly biases the RL generator away from such interactions, whereas filtering rejects trajectories that violate the safety criteria after generation. The ablation shows that shaping alone cannot eliminate all undesirable successes, since task completion can still outweigh a soft penalty. Filtering produces the larger reduction in unsafe interactions, and combining both yields the lowest illegal contact on both tasks while reducing container displacement relative to the unconstrained generator; on FMB-SqCircle, the combination concedes some container displacement to filtering alone, as shaping concentrates on illegal contact. By improving candidates before filtering, shaping also raises pass rates among successful rollouts from 31% to 50% on PnP-Banana and from 24% to 46% on FMB-SqCircle.

4.5 Complementarity with Corrective Data Collection

DAgger (Ross et al., 2011) and DART (Laskey et al., 2017) also address behavior-cloning distribution shift by adding supervision beyond the demonstrations. DAgger queries the teacher on learner-visited states, whereas DART perturbs teacher actions to expose recovery behavior (Section C). In contrast, DiscoDemo broadens the static dataset before downstream training by discovering multiple successful solution modes. The approaches thus target different coverage: DiscoDemo expands successful behaviors, while corrective collection adds supervision around off-nominal states.

As shown in Section 4.5, both DAgger and DART substantially improve P-RFCL, confirming the value of corrective supervision. Applying the same corrections to DiscoDemo yields further gains, showing that solution diversity does not eliminate learner-specific distribution shift. At the same time, DiscoDemo remains stronger under the same correction, indicating that broader successful-behavior coverage remains useful after corrective data are added. Unlike DAgger, DART requires no iterative learner rollouts. Overall, DiscoDemo improves the reusable static dataset without downstream interaction and remains complementary to corrective collection.

5 Conclusion

We presented DiscoDemo, which uses RL not as the controller, but as a generator of successful and diverse demonstrations for downstream visuomotor imitation learning. By combining a parallel reverse curriculum with latent-conditioned solution discovery, DiscoDemo expands the successful behaviors represented in the training data while retaining efficient state-based generation. Across four tabletop tasks, these datasets improve the downstream performance of a pretrained flow-matching VLA in simulation and on the real robot. Our analyses further show that broader dataset coverage is associated with stronger downstream performance, while excessive diversity can compromise task competence. The gains remain complementary to corrective data collection such as DAgger and DART, with safety shaping and filtering providing an additional mechanism for curating generated trajectories. Together, these results suggest that RL can be a practical tool for constructing broader imitation-learning datasets rather than only learning policies for direct deployment.

Limitations. DiscoDemo relies on accurate simulation, privileged state and reset access, and a well-defined success signal, limiting applicability to tasks with hard-to-model dynamics or ambiguous objectives. Our evaluation is restricted to four tabletop tasks with simple scenes and a modest sim-to-real gap; more complex settings may require real-world adaptation. The safety criteria are task-specific, and the diversity objective encourages distinct successful behaviors rather than directly optimizing downstream utility. Future work could explore more general safety specifications and learner-aware objectives that target useful coverage more directly.

Reproducibility Statement

We describe the simulated tasks, their initial-state distributions and success criteria, and the demonstrations in Section A; the generator's training procedure, reverse curriculum, diversity objective, safety terms, and all hyperparameters in Section B; our implementation of every baseline in Section C; the downstream fine-tuning recipe and evaluation protocol in Section D; and the definitions of all analysis metrics in Section E. Simulated downstream results average three fine-tuning seeds with 50 evaluation episodes each, generator training curves average five RL seeds, and real-robot results use 20 trials per task and method. The code, generated datasets, and trained checkpoints will be released.

AI Use Statement

In this work, we used generative AI tools to polish the writing, draft parts of the manuscript, and search for related work. We also used them as coding assistants to implement the data-generation pipelines, baselines, and analysis and plotting scripts. All AI-assisted text and code were reviewed by the authors, who take responsibility for the final content of this work.

References

Maria Bauza, Jose Enrique Chen, Valentin Dalibard, Nimrod Gileadi, Roland Hafner, Murilo F. Martins, Joss Moore, Rugile Pevceviciute, Antoine Laurens, Dushyant Rao, Martina Zambelli, Martin Riedmiller, Jon Scholz, Konstantinos Bousmalis, Francesco Nori, and Nicolas Heess. DemoStart: Demonstration-led auto-curriculum applied to sim-to-real with multi-fingered robots. *arXiv preprint arXiv:2409.06613*, 2024. (Cited on pages 1, 3, and 4.)

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025a. (Cited on pages 1 and 2.)

Xinyi Chen, Yilun Chen, Yanwei Fu, Ning Gao, Jiaya Jia, Weiyang Jin, Hao Li, Yao Mu, Jiangmiao Pang, Yu Qiao, et al. Internvla-m1: A spatially guided vision-language-action framework for generalist robot policy. *arXiv preprint arXiv:2510.13778*, 2025b. (Cited on pages 1 and 2.)

Cunxi Dai, Haoran Chang, Aditya Nisal, Rahul Kumar, Guofei Chen, Tao Chen, Yuzhe Qin, and Guanya Shi. Pgdg: Physically grounded data generation for robust bimanual policy learning from a single demonstration. *arXiv preprint arXiv:2605.21710*, 2026. (Cited on pages 2, 5, 6, 7, 18, 19, and 23.)

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018. (Cited on page 3.)

Dan Friedman and Adji Bousso Dieng. The Vendi score: A diversity evaluation metric for machine learning. *TMLR*, 2023. (Cited on page 7.)

Max Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Fei-Fei Li, Guanya Shi, et al. Cap-x: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026. (Cited on pages 2, 5, 6, 18, and 23.)

Caelan Reed Garrett, Ajay Mandlekar, Bowen Wen, and Dieter Fox. SkillMimicGen: Automated Demonstration Generation for Efficient Skill Learning and Deployment. In *CoRL*, 2024. (Cited on pages 1 and 2.)

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. (Cited on page 1.)

Ryan Hoque, Ajay Mandlekar, Caelan Reed Garrett, Ken Goldberg, and Dieter Fox. IntervenGen: Interventional data generation for robust and data-efficient robot imitation learning. In *IROS*, 2024. (Cited on page 2.)

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *CoRL*, 2025. (Cited on pages 2, 5, 6, and 19.)

Zhenyu Jiang, Yuqi Xie, Kevin Lin, Zhenjia Xu, Weikang Wan, Ajay Mandlekar, Linxi Jim Fan, and Yuke Zhu. Dexmimicgen: Automated data generation for bimanual dexterous manipulation via imitation learning. In *ICRA*, 2025. (Cited on pages 1 and 2.)

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024. (Cited on pages 1, 2, 5, 6, and 19.)

Donghu Kim, Youngdo Lee, Minho Park, Kinam Kim, I Nahendra, Takuma Seno, Sehee Min, Daniel Palenicek, Florian Vogt, Danica Kragic, et al. Flashsac: Fast and stable off-policy reinforcement learning for high-dimensional robot control. In *RSS*, 2026a. (Cited on pages 4 and 15.)

Hyunseung Kim, Byung Kun Lee, Hojoon Lee, Dongyoon Hwang, Sejik Park, Kyushik Min, and Jaegul Choo. Learning to discover skills through guidance. *NeurIPS*, 2023. (Cited on page 3.)

Kinam Kim, Namiko Saito, Heecheol Kim, Katsushi Ikeuchi, Jaegul Choo, and Yasuyuki Matsushita. Object-centric residual RL for zero-shot sim-to-real VLA enhancement. *arXiv preprint arXiv:2606.18953*, 2026b. (Cited on page 18.)

Saurabh Kumar, Aviral Kumar, Sergey Levine, and Chelsea Finn. One solution is not all you need: Few-shot extrapolation via structured maxent rl. *NeurIPS*, 2020. (Cited on pages 2, 3, and 4.)

Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. Dart: Noise injection for robust imitation learning. In *CoRL*, 2017. (Cited on pages 2, 9, and 19.)

Michael Laskin, Hao Liu, Xue Bin Peng, Denis Yarats, Aravind Rajeswaran, and Pieter Abbeel. Cic: Contrastive intrinsic control for unsupervised skill discovery. *arXiv preprint arXiv:2202.00161*, 2022. (Cited on page 3.)

Jianlan Luo, Charles Xu, Fangchen Liu, Liam Tan, Zipeng Lin, Jeffrey Wu, Pieter Abbeel, and Sergey Levine. FMB: A functional manipulation benchmark for generalizable robotic learning. *IJRR*, 2024. (Cited on page 5.)

Qian Luo, Ruizhe Liu, Pei Zhou, Xunzhe Zhou, and Yanchao Yang. Geometric entropy: When trajectory diversity helps and hurts in imitation learning. *arXiv preprint arXiv:2606.20871*, 2026. (Cited on page 8.)

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 *CoRL*, 2023. (Cited on pages 1, 2, 5, 6, 7, 18, and 23.)

NVIDIA. Isaac Sim. URL https://github.com/isaac-sim/IsaacSim. (Cited on pages 5 and 14.)

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *ICRA*, 2024. (Cited on page 1.)

Seohong Park, Jongwook Choi, Jaekyeom Kim, Honglak Lee, and Gunhee Kim. Lipschitz-constrained unsupervised skill discovery. In *International Conference on Learning Representations*, 2022. (Cited on page 3.)

Seohong Park, Oleh Rybkin, and Sergey Levine. Metra: Scalable unsupervised rl with metric-aware abstraction. In *ICLR*, 2024. (Cited on pages 2, 3, 4, and 16.)

Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *NeurIPS*, 1988. (Cited on page 1.)

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011. (Cited on pages 2, 9, and 19.)

Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. Laion-5b: An open large-scale dataset for training next generation image-text models. *NeurIPS*, 2022. (Cited on page 1.)

Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*, 2019. (Cited on page 3.)

Stone Tao, Arth Shukla, Tse-kai Chan, and Hao Su. Reverse forward curriculum learning for extreme sample and demonstration efficiency in reinforcement learning. In *ICLR*, 2024. (Cited on pages 1, 2, 3, 4, 5, and 17.)

Yang Tian, Yuyin Yang, Yiman Xie, Zetao Cai, Xu Shi, Ning Gao, Hangxu Liu, Xuekun Jiang, Zherui Qiu, Feng Yuan, et al. Interndata-a1: Pioneering high-fidelity synthetic data for pre-training generalist policy. In *CVPR*, 2026. (Cited on pages 1 and 2.)

Marcel Torne, Anthony Simeonov, Zechu Li, April Chan, Tao Chen, Abhishek Gupta, and Pulkit Agrawal. Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation. In *RSS*, 2024. (Cited on pages 1 and 2.)

Wenli Xiao, Haotian Lin, Andy Peng, Haoru Xue, Tairan He, Yuqi Xie, et al. Self-improving vision-language-action models with data generation via residual rl. In *ICLR*, 2026. (Cited on pages 1, 2, 3, 5, 6, 18, and 23.)

Charles Xu, Qiyang Li, Jianlan Luo, and Sergey Levine. RLDG: Robotic generalist policy distillation via reinforcement learning. In *RSS*, 2025. (Cited on page 3.)

Zifan Xu, Ran Gong, Maria Vittoria Minniti, Kausik Sivakumar, Ahmet Salih Gundogdu, Eric Rosen, Riedana Yan, Tushar Kusnur, Zixing Wang, Di Deng, Peter Stone, Xiaohan Zhang, and Karl Schmeckpeper. ExpertGen: Scalable sim-to-real expert policy learning from imperfect behavior priors. *arXiv preprint arXiv:2603.15956*, 2026. (Cited on pages 1, 2, 3, 5, 6, 18, and 23.)

Rushuai Yang, Zhiyuan Feng, Tianxiang Zhang, Kaixin Wang, Chuheng Zhang, Li Zhao, Xiu Su, Yi Chen, and Jiang Bian. Discover, learn, and reinforce: Scaling vision-language-action pretraining with diverse rl-generated trajectories. *arXiv preprint arXiv:2511.19528*, 2025. (Cited on page 3.)

APPENDIX

In this appendix, we provide the implementation details needed to reproduce our results, together with additional results that do not fit in the main paper. We first describe the tasks, their initial-state distributions and success conditions, and how the demonstrations were collected (Section A). We then detail the generator, including its hyperparameters, the parallel reverse curriculum, the diversity objective with success gating, the safety terms, and its differences from the original RFCL (Section B), followed by our implementation of every baseline (Section C) and the downstream fine-tuning recipe and evaluation protocol (Section D). Next, we define the metrics used to analyze the generated datasets, namely diversity, coverage, generation yield, and the safety metrics (Section E). Finally, we report the full simulation and real-robot results, the diversity trade-off on every task, how training and generation cost are measured, and qualitative rollouts of the downstream policies (Section F). Below is the table of contents for the appendix.

A Tasks and Experimental Setup **14**

B Generator Implementation **15**

B.1 Training Setup 15

B.2 Parallel Reverse Curriculum 16

B.3 Diversity Objective 16

B.4 Safety Shaping and Filtering 16

B.5 RFCL and Our Parallel Reimplementation 17

C Baselines **18**

D Downstream Policy Training and Evaluation **19**

E Analysis Metrics **19**

E.1 Diversity and Coverage Metrics 19

E.2 Generation Yield and Qualitative Trajectories 20

E.3 Safety Metrics 20

F Additional Results **22**

F.1 Main results — full tables 22

F.2 Diversity trade-off on every task 22

F.3 Qualitative trajectories on StackCube and FMB-Round 22

F.4 Efficiency Measurement 22

F.5 Qualitative Rollouts 23

A TASKS AND EXPERIMENTAL SETUP

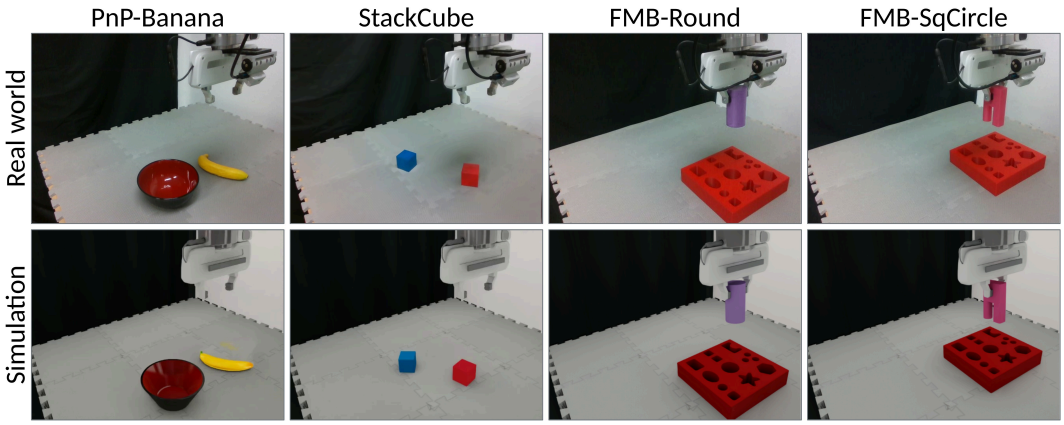


Figure 9: **Experimental Setup.** The four tasks on the real FR3 (top) and in the simulated counterpart used for data generation (bottom), seen from the same exocentric camera.

Robot and simulation. All tasks use a Franka Research 3 arm with a parallel-jaw gripper, reconstructed in Isaac Sim (NVIDIA) from the real workcell (Fig. 9). Physics runs at 120 Hz and the policy acts at 20 Hz. The generator's action is 8-dimensional in $[-1, 1]$: seven relative joint-position increments, each scaled by a per-joint rate limit taken from the 99th percentile of joint velocities in the teleoperated demonstrations and passed through a jerk-limited command filter, and one binary gripper command. The privileged state is 45-dimensional: the pose of the manipulated object and of the container (bowl, blue cube, or fixture board) together with their positions relative to the tool center point, the seven joint positions, the gripper state, the end-effector pose, the gripper actuation state, and the previous action. The downstream policy instead observes a left-shoulder exocentric camera and a wrist camera, whose intrinsics, distortion, and extrinsics are calibrated on the real robot, together with the joint and gripper state.

Tasks. Tab. 2 lists the horizon, initial-state distribution ρ_0, and success condition of each task. An episode succeeds once the success condition in Tab. 2 is met. When training the generator, we additionally require the condition to hold for 20 consecutive control steps (1 s) before rewarding and terminating the episode; this training-time choice makes the generator settle into stable end states rather than pass through the goal, which yields more stable demonstrations.

Table 2: Task specifications. Positions are in the robot base frame.

Task	Horizon	Initial-state distribution ρ_0	Success condition
PnP-Banana	30 s	banana and bowl positions uniform in $x \in [0.41, 0.64]$ m, $y \in [-0.15, 0.15]$ m, at least 20 cm apart; banana yaw uniform in $\pm 180^\circ$	banana center inside the bowl, in contact with the bowl, and released by the gripper
StackCube	30 s	red and blue cube positions in the same region, at least 15 cm apart; red-cube yaw uniform in $\pm 45^\circ$	red cube on the blue cube: height offset within 1 cm of 5 cm, horizontal offset ≤ 2.5 cm, speed ≤ 2 cm/s
FMB-Round	20 s	peg grasped at the home configuration; fixture-board position uniform in ± 3 cm and yaw in $\pm 15^\circ$	peg fully inserted (within 2 mm), lateral error $\leq$ 5 mm, axis misalignment $\leq 10^\circ$
FMB-SqCircle	20 s	as FMB-Round	as FMB-Round

Demonstrations. For each task we collect 20 demonstrations by teleoperating the real robot with a Meta Quest VR controller, recorded at 20 Hz. The object poses of each episode are estimated from the exocentric RGB-D view, and the recorded joint trajectory is replayed in simulation from

the corresponding initial scene; successful replays form the demonstration set used for the reverse curriculum and for H20, MimicGen, and PGDG. For the insertion tasks, where the peg starts in the gripper, the gripper command is held closed.

Real-robot evaluation. Each policy is evaluated for 20 trials per task on the real robot, reproducing the same set of initial object placements for every method. Transfer is zero-shot: the policies are trained only on simulated data, without real-robot fine-tuning. For every method we deploy the policy of the first training seed, fixed in advance of the real-robot evaluation. SR and Time are defined as in simulation: SR is the fraction of the 20 trials that succeed, and Time is the mean completion time over the successful trials.

B Generator Implementation

B.1 Training Setup

We train P-RFCL and DiscoDemo with FlashSAC (Kim et al., 2026a) for 50M environment steps on a single B200 GPU, using 2,048 parallel environments; PLD and ExpertGen are trained for up to 24 hours each (Section C). The actor and critic follow FlashSAC's residual MLP architecture with weight normalization; the critic is a pair of categorical (distributional) Q-functions whose target is the minimum of the two. Privileged observations are mapped to $[-1, 1]$ with fixed physical bounds—joint limits and the task workspace—rather than statistics of the demonstrations, so that no dimension saturates when the policy leaves the demonstrated region. Rewards are scaled by a running estimate of the return scale. Every batch draws half of its transitions from the demonstrations and half from online experience, throughout training. Tab. 3 lists the hyperparameters.

Table 3: Generator hyperparameters, shared by P-RFCL and DiscoDemo and by all tasks.

Hyperparameter	Value
Parallel environments	2,048
Environment steps	50M
Actor / critic	2 residual blocks, width 128 / 256
Critic support	101 atoms on $[-5, 5]$, two critics
Optimizer	Adam, lr $3\times10^{-4} \rightarrow 1.5\times10^{-4}$ (cosine)
Batch size	2,048 (50% demonstrations)
Gradient updates per vectorized step	64
Actor update period	2
n-step return, discount γ	8, 0.99
Target update rate τ	0.01
Initial temperature	0.01
Target entropy	$\frac{\lvert\mathcal{A}\rvert}{2}\ln(2\pi e\,\sigma^2)$, $\sigma = 0.15$
Replay capacity	100M transitions
Precision	mixed (fp16), `torch.compile`
Curriculum stride Δc	16 demonstration steps
Advance threshold p / window	0.9 / 10 episodes
Reset-offset distribution	geometric, $q = 0.5$
Forward transition	half of the demonstrations solved
Latent skill z	3-dim, $\mathcal{N}(0, I)$
Representation ϕ_ψ	MLP 256–256, ReLU, output dim. 3
ϕ_ψ / dual learning rate	10^{-4}
Dual variable λ (init.) / slack ϵ	1.0 / 10^{-3}
Contact / jamming weight λ_c / λ_j	10^{-4} / 10^{-6}

B.2 Parallel Reverse Curriculum

Let $\tau_1, \ldots, \tau_N$ be the demonstrations ($N = 20$) with lengths T_i, and let c_i be the curriculum frontier of τ_i, initialized near its end. The M parallel environments are divided as evenly as possible among the demonstrations, so that each demonstration τ_i is served by a group of $\lfloor M/N \rfloor$ or $\lceil M/N \rceil$ environments (102 or 103 for $M = 2{,}048$ and $N = 20$). Whenever an environment of group i resets during the reverse phase, it draws a start step

$$\Pr(t) \;\propto\; (1-q)^{\,t-c_i}, \qquad t \in \{c_i, \ldots, T_i - 1\}, \quad q = 0.5, \tag{7}$$

and resets the simulator to the demonstration state s_t^i, so that start steps concentrate at the frontier. The curriculum of τ_i is advanced using the episodes of its group started exactly at its frontier: once at least 10 such episodes have finished and the success rate $\hat{p}_i$ of the most recent 10 reaches $p = 0.9$, we set $c_i \leftarrow \max(0, c_i - \Delta c)$. A demonstration is solved once this test also passes at $c_i = 0$, and when half of the demonstrations are solved all environments switch to the task distribution ρ_0 (the forward phase). Reverse-phase episodes are truncated after a fixed margin plus the remaining demonstration length $T_i - t$, and, as in the forward phase, count as successful only after the success condition has held for 1 s.

B.3 Diversity Objective

The latent skill $z \in \mathbb{R}^3$ is drawn from $\mathcal{N}(0, I)$ at every episode start and appended to the observations of the actor and the critics. The representation ϕ_ψ is an MLP with two hidden layers of 256 ReLU units and output dimension 3, applied to the normalized privileged state without the previous action and the gripper actuation state. Following METRA (Park et al., 2024), ϕ_ψ maximizes $\mathbb{E}[(\phi_\psi(s') - \phi_\psi(s))^\top z]$ subject to $\|\phi_\psi(s') - \phi_\psi(s)\|^2 \le 1$, which we enforce with a dual variable λ on the relaxed constraint $\min(\epsilon,\, 1 - \|\phi_\psi(s') - \phi_\psi(s)\|^2)$; ϕ_ψ is trained on every sampled transition, including demonstrations. Because the critic uses n-step returns, the diversity reward of a sampled transition is computed at update time over the same n steps, $\langle \phi_\psi(s_{t+n}) - \phi_\psi(s_t), z \rangle$, which is the sum of the per-step rewards in (3).

Success gating. The indicator $y(\tau)$ is known only when an episode ends, so it is applied retroactively. Each episode receives a unique identifier that is stored with its transitions; when the episode terminates or is truncated, its outcome is written to a table indexed by that identifier, and each sampled transition looks up its gate at update time. Transitions of episodes that have not finished yet, and demonstration transitions, receive no diversity reward.

Reward. The generator is trained on

$$r_t \;=\; r_{\text{task}}(s_t, a_t) \;-\; r_{\text{safe}}(s_t, a_t) \;+\; \alpha\, y(\tau)\, r_{\text{div}}(s_t, s_{t+1}, z), \tag{8}$$

where r_{task} is $+1$ at the step at which the success condition has held for 1 s, which also terminates the episode, and 0 otherwise. The reward scaling above is applied to $r_{\text{task}} - r_{\text{safe}}$, and the diversity term is added after it without further scaling. The diversity objective is active in both curriculum phases. For the main downstream comparison, we use $\alpha = 0.3$ on every task; P-RFCL uses the same generator with $\alpha = 0$.

B.4 Safety Shaping and Filtering

The two terms of (4) are

$$C_{\text{contact}}(s_t) = \min\Big(\sum_{(b,o) \notin \mathcal{C}_{\text{allow}}} \|F_{b,o}\|_2,\; c_{\text{c}} \Big), \tag{9}$$

$$C_{\text{jam}}(s_t) = \sum_{d \in \{\perp, \|\}} \big[\min(F^d_{\text{peg}},\, c_{\text{j}}) - \delta_d \big]_+^2, \tag{10}$$

where $[x]_+ = \max(x, 0)$, so C_{jam} is a squared excess force (in N^2) that grows quadratically with how far the peg pushes past each deadband. C_{contact} sums the contact forces $F_{b,o}$ between robot links b and scene bodies o over all pairs outside the allowed set $\mathcal{C}_{\text{allow}}$, which contains only the finger–object contacts that constitute a grasp. C_{jam} splits the force of the peg on the board into

its axial component $F^{\perp}_{\text{peg}}$ and lateral component $F^{\parallel}_{\text{peg}}$ with separate deadbands $\delta_{\perp} < \delta_{\parallel}$: a correct insertion is guided by lateral contact with the hole wall, whereas a jammed peg pushes axially, so penalizing $\|F\|$ as a whole would favor the failure. Both forces are saturated at $c_{\text{c}} = c_{\text{j}} = 100\,\text{N}$ so that rare force spikes do not dominate the reward. We use $\lambda_{\text{c}} = 10^{-4}$ on every task and $\lambda_{\text{j}} = 10^{-6}$ with $\delta_{\perp} = 5\,\text{N}$ and $\delta_{\parallel} = 15\,\text{N}$ on the insertion tasks. For the export filter we use $\epsilon_{\text{j}} = (15\,\text{N})^2$, i.e., we discard trajectories whose deadband-adjusted force vector has an ℓ_2 norm above $15\,\text{N}$.

For contact, a trajectory is discarded if C_{contact} exceeds $\epsilon_{\text{c}} = 50\,\text{N}$ at any step or stays above $20\,\text{N}$ for $150\,\text{ms}$ (three consecutive control steps). The same export filter is applied to every dataset in the main comparison: C_{contact} on every task and C_{jam} on the two insertion tasks. The export filter uses only these two signals; container displacement is not a filter criterion and is reported only as an evaluation metric (Section E.3).

B.5 RFCL and Our Parallel Reimplementation

Original RFCL. RFCL (Tao et al., 2024) targets sample- and demonstration-efficient RL: from a handful of demonstrations, it learns a single state-based policy that solves a sparse-reward task, and it is evaluated by the success rate reached per environment interaction. Its reverse curriculum pins one environment to each demonstration (eight environments in total), samples start steps from a short geometric window at the demonstration's frontier, and moves the frontier back by four steps once the pinned environment has succeeded in its last three episodes. A forward curriculum then samples initial states by prioritized level replay, and the policy is trained with SAC using two layer-normalized critics and a discount of 0.9.

RFCL was not designed for data generation. RFCL optimizes the success of one policy and has no mechanism that keeps several successful behaviors, which is what DISCODEMO adds with the diversity objective. Its curriculum is also built for a few environments: the frontier of a demonstration advances only when every recent episode of its pinned environment succeeds, a criterion suited to measuring sample efficiency that does not scale to thousands of parallel environments.

Our reimplementation. We keep RFCL's idea of resetting to demonstration states but replace most of its components to exploit massively parallel simulation (Tab. 4); the change that makes the curriculum parallelizable is the advance criterion, which pools the outcomes of all episodes started at a demonstration's frontier instead of requiring each pinned environment to succeed (Section B.2). Because of these changes, the $\alpha = 0$ results in Figures 6 and 12 should not be compared directly with the numbers reported by Tao et al. (2024).

Table 4: Our $\alpha = 0$ generator (P-RFCL in our experiments) compared with the original RFCL configuration.

	Original RFCL	Ours
Reverse-phase environments	8	2,048
Environment-to-demonstration	one pinned environment each	a group of $\approx M/N$ environments each
Advance criterion	3 consecutive successes	success rate ≥ 0.9 over 10 episodes
Curriculum stride	4	16
Forward phase	PLR over initial states	ρ_0 once half the demonstrations are solved
Success	at a single step	held for 1 s
Critic	MLP Q-function	categorical, residual
Discount / n-step	0.9 / 1	0.99 / 8
Batch size / updates per step	256 / 16	2,048 / 64
Target rate / initial temperature	0.005 / 1.0	0.01 / 0.01
Additional terms	—	safety shaping

Speedup from parallelization. We compare the two curricula on an earlier version of PnP-Banana with five demonstrations, running only the reverse phase on a single GPU with the same number of gradient updates per vectorized step. With RFCL's rule of one pinned environment per demonstration, the reverse phase takes 8.2 hours of wall-clock time, and giving the same rule 2,048 environ-

ments does not help, since its per-environment criterion must then hold for hundreds of environments at once and the frontier stalls. Our pooled criterion with 2,048 environments completes the reverse phase in 29 minutes, about 17× faster in wall-clock time. The gain is in wall-clock time rather than in samples: the parallel curriculum consumes more environment steps, which massively parallel simulation makes cheap.

C BASELINES

All generated-data baselines produce 3,000 successful demonstrations under the same initial-state distribution, success condition, horizon, and camera configuration as DISCODEMO.

H20. The 20 teleoperated demonstrations of Section A, used directly.

Motion planning (MP, CaP-X-style). Following CaP-X (Fu et al., 2026), a scripted waypoint program reads the privileged object poses, solves inverse kinematics for each waypoint, and connects the resulting joint targets with jerk-limited trajectories under the same joint rate limits as the generator. For PnP-Banana it grasps, lifts, and releases the banana above the bowl rim; for StackCube it adds a pre-grasp alignment waypoint and uses the cube's 90° symmetry to choose the grasp; for the insertion tasks it aligns the grasped peg over the hole, descends, and inserts, retrying with a small spiral search if the peg jams. The planner does not reason about collisions.

MimicGen (MG). Following MimicGen (Mandlekar et al., 2023), each demonstration is split at its first grasp into object-centric segments (grasp and place for PnP-Banana and StackCube, a single insertion segment for the insertion tasks). For a new scene, each segment is transformed into the frame of its reference object, $T'_{\text{ee}} = T'_{\text{obj}} T_{\text{obj}}^{-1} T_{\text{ee}}$, and converted to joint targets by inverse kinematics; the source of each segment is chosen among the three demonstrations whose object layout is closest to the new scene. Segments are joined by jerk-limited joint interpolation, and the gripper replays the recorded commands.

PGDG (perturbation-stage reimplementation). Since no official implementation of PGDG (Dai et al., 2026) is available, we reimplement its perturbation stage on top of the MimicGen demonstrations. For each source demonstration, the arm's joint-target sequence is perturbed by a residual B-spline with 16 control points whose coefficients are Gaussian with initial scale 0.03; we sample 16 candidates per iteration for 2 iterations, refit the perturbation distribution to the successful candidates that stay within a tube around the source (quantiles 0.1 and 0.8 of the object-feature deviation), and keep up to three successful perturbations per source. Because several perturbed variants are kept per source, the retained dataset covers fewer distinct initial scenes than a MimicGen dataset of the same size.

PLD (FlashSAC variant). Following PLD (Xiao et al., 2026), a $\pi_{0.5}$ policy fine-tuned on the demonstrations serves as a frozen base policy, and a residual policy outputs a correction to its arm command, $a = \text{clip}(a_{\text{base}} + 0.4\, a_{\text{res}})$. The original PLD learns the residual from images, which did not learn within our budget; following object-centric residual RL (Kim et al., 2026b), we therefore give the residual policy the privileged state while the base policy remains vision-based, and train it with FlashSAC. The residual is trained for up to 24 hours (2.4–2.6M environment steps); demonstrations are collected from rollouts of the combined policy.

ExpertGen (EG, FlashSAC variant). Following ExpertGen (Xu et al., 2026), an RL policy steers the initial noise of the same frozen $\pi_{0.5}$ prior, making one 32-dimensional decision per 15-step action chunk with magnitude bounded by 3. We train it with FlashSAC in place of FastTD3 under the sparse task reward, for up to 24 hours (3.2–4.0M environment steps), and collect demonstrations from the steered policy.

P-RFCL (DISCODEMO@α=0). The generator of Section B with $\alpha = 0$, trained and rolled out identically to DISCODEMO; it keeps RFCL's reset-to-demonstration curriculum but parallelizes it (Section B.5).

DAgger. For Section 4.5, we use the frozen P-RFCL or DISCODEMO generator as the teacher in DAgger (Ross et al., 2011). The learner starts from a policy fine-tuned on that generator's demonstrations and controls the robot during collection. At each visited state, the teacher supplies an action label, converted to a joint-position target through its command filter. We retain both successful and unsuccessful trajectories, add the labeled observations to the accumulated dataset, and fine-tune the learner before the next collection round.

DART. For DART (Laskey et al., 2017), the teacher controls the robot with perturbed actions. We add fixed isotropic Gaussian noise with standard deviation $\sigma \in \{0.1, 0.3\}$ to the seven normalized arm commands, clip them to $[-1, 1]$, and independently flip the gripper command with probability 0.02. Each observation is labeled with the noise-free teacher target, computed from the joint and command-filter state before executing the perturbed action. The resulting demonstrations are filtered using the task-specific export criteria and used for downstream fine-tuning.

D DOWNSTREAM POLICY TRAINING AND EVALUATION

Dataset export. Each retained trajectory is rendered by resetting the simulator to its stored states and capturing the two cameras at 480×640 resolution and 20 Hz. For DISCODEMO, a new latent skill is drawn for every episode and held fixed throughout the rollout. The observation stores the seven joint positions and the gripper state; the action stores the commanded joint-position target after action processing and a binary gripper command.

Policy training. We fine-tune the same DROID-pretrained $\pi_{0.5}$ (Intelligence et al., 2025; Khazatsky et al., 2024) on every dataset with the recipe in Tab. 5. Images are resized with padding to 224×224 without augmentation, joint actions are expressed relative to the joint state at the start of each chunk while the gripper stays absolute, and states and actions are normalized by their 1st and 99th percentiles. Each task uses one fixed language instruction, e.g., "stack the red cube on the blue cube". For each dataset we train three policies with different seeds.

Table 5: Downstream $\pi_{0.5}$ fine-tuning recipe, identical for every data source.

Hyperparameter	Value
Initialization	DROID-pretrained $\pi_{0.5}$ (joint-position actions)
Trained parameters	all (vision encoder, language model, action expert)
Precision	bfloat16
Optimizer	AdamW, $\beta = (0.9, 0.95)$, weight decay 0.01, grad. clip 1.0
Learning rate	1k warmup, cosine $2.5 \times 10^{-5} \rightarrow 2.5 \times 10^{-6}$
Batch size / steps	16 / 20k
Cameras	left-shoulder exocentric, wrist; 224×224
Action chunk	15 steps, all executed before replanning
Flow-matching steps	10

Simulation evaluation. Each policy is evaluated from its final checkpoint for 50 episodes drawn from the task distribution ρ_0, the same distribution used to generate its training data. An episode is counted as successful if the success condition is met at any step. *SR* is the fraction of successful episodes; *Time* is the mean time to the first success over successful episodes, with a policy that never succeeds charged the full horizon H; and *Succ./h* is the number of successes per hour of operation, $3600 \cdot \mathrm{SR}/(\mathrm{SR} \cdot \mathrm{Time} + (1 - \mathrm{SR}) \cdot H)$, which charges every failure the full horizon. Each metric is computed per training seed and reported as mean $\pm$ standard deviation over the three seeds.

E ANALYSIS METRICS

E.1 DIVERSITY AND COVERAGE METRICS

We follow the trajectory representation of PGDG (Dai et al., 2026) with two changes: we drop the action channel and replace the log-determinant with the Vendi score.

Features. At each step we use $\psi(s_t) = [\mathbf{p}_{\text{obj}}, \mathbf{q}_{\text{obj}}, \mathbf{p}_{\text{ee}}, \mathbf{q}_{\text{ee}}] \in \mathbb{R}^{14}$, the single-arm version of PGDG's $[T_{\text{obj}}, T_{\text{ee}}]$; quaternions are sign-canonicalized ($w \geq 0$). PGDG additionally appends the action a_t. We omit it because action signals differ in character across generators—replayed or planned trajectories are nearly constant, whereas RL policies are not—so the action channel would dominate the comparison without reflecting different solutions. When the end-effector pose is not stored, we recover it by forward kinematics (max. position error 2.2×10^{-5} m against logged poses).

Embedding and kernel. Features are z-scored with statistics pooled over all sources of a task. Each trajectory is truncated at its first success, linearly resampled to 64 steps, and transformed per dimension with DCT-II; we drop the DC term and keep the next 8 coefficients (PGDG Eq. 31), giving a 112-dimensional embedding. Similarity is $k(\xi_i, \xi_j) = \exp(-\gamma \|\xi_i - \xi_j\|^2)$ (PGDG Eq. 32), with γ set per task to the inverse median pairwise squared distance over all sources, as PGDG does not report its bandwidth.

Vendi score. For a set of m trajectories with kernel matrix K, $\text{VS} = \exp(-\sum_i \lambda_i \log \lambda_i)$, where λ_i are the eigenvalues of K/m; it ranges from 1 (identical trajectories) to m. PGDG's objective $\log\det(K + \epsilon I)$ is suited to subset selection but not to evaluation: near-duplicate trajectories drive eigenvalues to zero, so the value is set by $\log \epsilon$ and the number of such eigenvalues rather than by diversity. Because VS grows with m, we fix m per metric and average over random subsets.

Inputs. *Diversity*: for each task, one fixed initial scene; per source, 24 successful rollouts, m=20, averaged over 200 subsets. *Coverage*: per source, 1,000 trajectories sampled from the 3,000-demonstration training dataset (PLD and ExpertGen: from their generated pools), m=200, averaged over 100 subsets. Both are averaged over the four tasks. Fig. 10 reports every panel of Fig. 5 per task; its coverage–diversity row uses logarithmic axes because the baselines cluster within a narrow range on individual tasks.

The diversity axis of Figures 6 and 12 applies the same diversity procedure to the rollouts of each generator in the α sweep, from the same fixed scene per task.

Coverage, diversity, and downstream success. Fig. 11 plots, for each generator, the coverage (top) and diversity (bottom) of its dataset against the success rate of the $\pi_{0.5}$ policy trained on it (Fig. 3). Each panel reports the Spearman (S) and Pearson (P) correlation over the seven generators; Pearson coefficients and the dashed least-squares fits use the $\log_{10}$ of the metric. Coverage of the full training dataset tracks downstream success more closely than diversity from a single fixed initial state: its rank correlation is higher or equal on every task (S 0.86 vs. 0.68 on PnP-Banana, 0.79 vs. 0.64 on StackCube, 0.50 vs. 0.50 on FMB-Round, 0.43 vs. 0.18 on FMB-SqCircle, and 0.50 vs. 0.39 on the task average). These associations support dataset-level coverage as a useful indicator of downstream performance, rather than fixed-scene diversity alone; with seven generators per task, however, individual coefficients remain noisy.

E.2 Generation Yield and Qualitative Trajectories

Generation yield is the fraction of rollouts from ρ_0 that succeed, before any export filter. For PGDG it is the product of the MimicGen yield and the success rate of its perturbed candidates; for PLD and ExpertGen it is the success rate at the end of RL training. The trajectory overlays in Figures 4, 6 and 13 use the same fixed scene per task as the diversity metric: for each method we collect 24 successful rollouts and project the tool center point into the calibrated exocentric camera.

E.3 Safety Metrics

We report illegal contact (IC) and container displacement (CD) on the exported demonstrations. Let $\mathcal{C}_{\text{allow}}$ denote the intended finger–object contacts. For a trajectory τ, IC is defined as

$$\text{IC}(\tau) = \sum_t \sum_{(b,o) \notin \mathcal{C}_{\text{allow}}} \|F_{b,o}(t)\|_2 \, \Delta t, \qquad \Delta t = 0.05 \text{ s}, \tag{11}$$

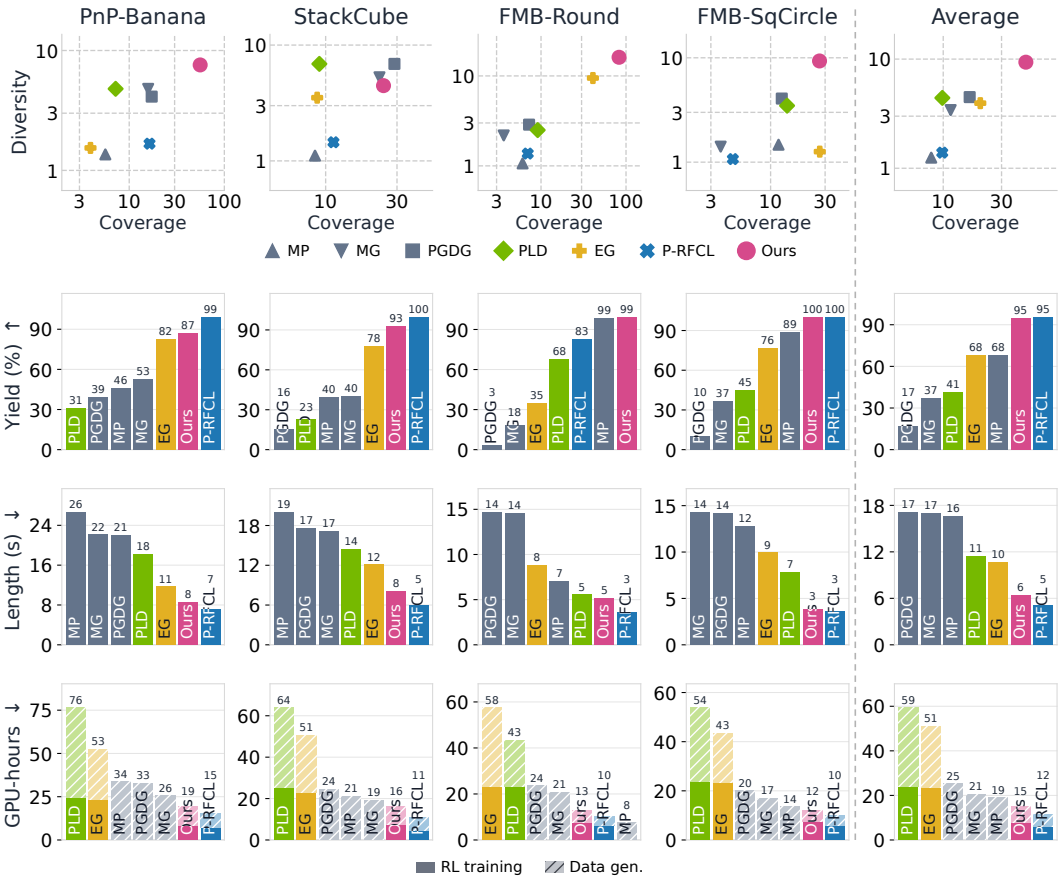


Figure 10: **Per-task dataset analysis.** DISCODEMO attains the highest diversity and the highest or tied-highest coverage on three of the four tasks; on StackCube, PGDG exceeds it on both.

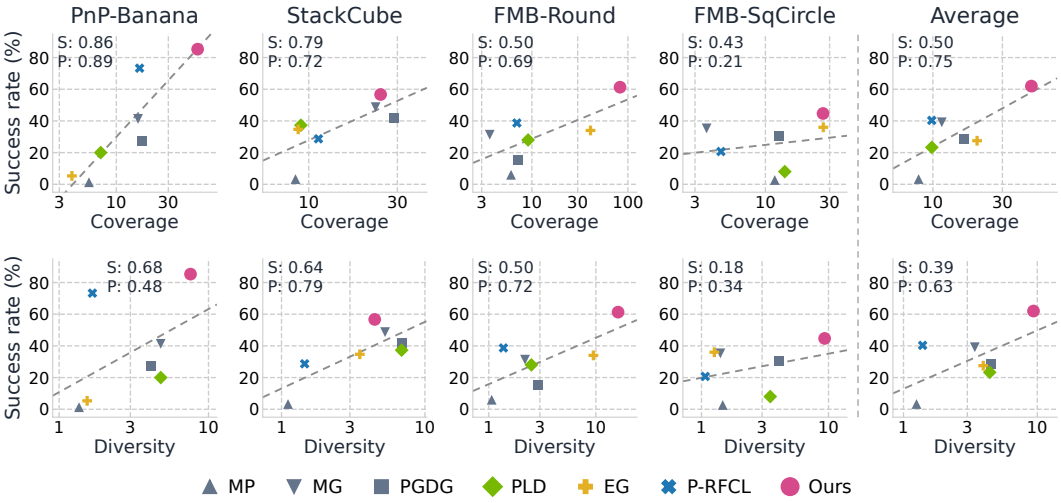


Figure 11: **Coverage and diversity versus downstream success.** Dataset coverage (top) tracks the success of the downstream policy more closely than fixed-scene diversity (bottom).

where $F_{b,o}(t)$ is the simulator contact force between robot body b and scene object o. The allowed contacts are finger–banana contact for PnP-Banana and finger–peg contact for FMB-SqCircle; all other monitored robot–scene pairs are treated as illegal. Unlike the training contact penalty, IC does not clip large contact forces and is used only as an evaluation metric.

Container displacement is defined as

$$\mathrm{CD}(\tau) = \max_t \|\mathbf{p}^{xy}_{\text{container}}(t) - \mathbf{p}^{xy}_{\text{container}}(0)\|_2 \,, \tag{12}$$

where the container is the bowl for PnP-Banana and the fixture board for FMB-SqCircle. We report the dataset mean of both metrics, with CD converted to centimeters.

Filter pass rates. Tab. 6 lists how many successful rollouts of each generator in Tab. 1 pass the export filter of (5). Each pool is collected with deterministic actions from a single checkpoint at a comparable training budget (29–33M environment steps). Unfiltered rows of Tab. 1 use a fixed random subset of 3,000 successes, and filtered rows use the first 3,000 passing successes in the same order. Without shaping, only 2,701 FMB-SqCircle successes pass, so that row of Tab. 1 is computed on these 2,701 trajectories.

Table 6: Export-filter pass rates among successful rollouts. The last column is the number of successes needed for 3,000 filtered demonstrations.

Task	Shaping	Successes	Passed	Pass rate (%)	Needed
PnP-Banana	✗	11,428	3,577	31.3	9,585
PnP-Banana	✓	10,730	5,358	49.9	6,008
FMB-SqCircle	✗	11,371	2,701	23.8	12,630
FMB-SqCircle	✓	11,632	5,395	46.4	6,468

F Additional Results

F.1 Main results — full tables

For completeness, Tables 7 and 8 report the real-robot and simulation results shown in Fig. 3.

F.2 Diversity trade-off on every task

Fig. 6 combines PnP-Banana trajectories with four-task averages; Fig. 12 reports generator success, diversity, and downstream success per task and on average. The top row adds the generator's training curve at each α, and the generator success below is where each curve ends; both average five seeds per cell and shade $\pm$ one standard deviation.

F.3 Qualitative trajectories on StackCube and FMB-Round

Fig. 4 shows PnP-Banana and FMB-SqCircle; the remaining two tasks, drawn the same way, are below.

F.4 Efficiency Measurement

Generator training. The wall-clock training time of P-RFCL and DiscoDemo in Fig. 7 is estimated from measurements on one B200 GPU with in-training evaluation, checkpointing, and video recording disabled: we time 1.5M environment steps of each curriculum phase and extrapolate to 50M steps using each run's own phase-switch step, so that queueing and preemption do not enter the comparison. PLD and ExpertGen report the wall-clock time of their own training runs on the same GPU type. Every curve in Fig. 7 averages five training seeds, with the shaded band showing ± 1 standard deviation.

End-to-end cost. The GPU-hours in Fig. 5 (d) are normalized to a single process on one GPU, so that job packing and preemption do not enter. Generator training counts the full 50M-step run without in-training evaluation, and data generation counts the rollout and rendering of 3,000 retained demonstrations; $\pi_{0.5}$ fine-tuning is excluded because it is identical for every source. The per-task split into the two stages is the bottom row of Fig. 10.

Table 7: Real-world experiments.

	PnP-Banana			StackCube			FMB-Round			FMB-SqCircle		
Data source	SR (%) ↑	Time (s) ↓	Succ./h ↑	SR (%) ↑	Time (s) ↓	Succ./h ↑	SR (%) ↑	Time (s) ↓	Succ./h ↑	SR (%) ↑	Time (s) ↓	Succ./h ↑
H20	0.0	30.0	0	0.0	30.0	0	0.0	20.0	0	0.0	20.0	0
MP (2026)	20.0	17.1	26	0.0	30.0	0	40.0	14.6	81	5.0	14.1	9
MG (2023)	0.0	30.0	0	50.0	20.9	71	25.0	14.6	48	20.0	12.8	39
PGDG (2026)	0.0	30.0	0	50.0	19.1	73	30.0	13.6	60	20.0	14.1	38
PLD (2026)	5.0	16.8	6	30.0	17.3	41	15.0	8.3	30	5.0	5.8	9
ExpertGen (2026)	30.0	17.1	41	30.0	14.5	43	5.0	7.2	9	5.0	5.6	9
P-RFCL	45.0	9.1	79	35.0	**6.0**	58	45.0	**2.8**	132	30.0	**3.2**	72
Ours	**90.0**	**8.7**	**300**	**75.0**	9.3	**187**	**80.0**	3.4	**429**	**65.0**	5.1	**227**

Table 8: Simulation experiments.

	PnP-Banana			StackCube			FMB-Round			FMB-SqCircle		
Data source	SR (%) ↑	Time (s) ↓	Succ./h ↑	SR (%) ↑	Time (s) ↓	Succ./h ↑	SR (%) ↑	Time (s) ↓	Succ./h ↑	SR (%) ↑	Time (s) ↓	Succ./h ↑
H20	1.3 ±2.3	27.5 ±4.3	2 ±3	7.3 ±5.0	21.2 ±1.7	9 ±6	0.0 ±0.0	20.0	0 ±0	0.0 ±0.0	20.0	0 ±0
MP (2026)	1.3 ±2.3	29.4 ±1.1	2 ±3	3.3 ±3.1	22.6 ±6.4	4 ±4	6.0 ±3.5	16.3 ±1.6	11 ±6	2.7 ±3.1	16.7 ±2.9	5 ±6
MG (2023)	41.3 ±16.2	21.5 ±2.5	58 ±26	48.7 ±11.0	18.5 ±1.1	73 ±20	31.3 ±4.2	12.3 ±0.1	64 ±10	35.3 ±13.0	9.6 ±0.6	81 ±37
PGDG (2026)	27.3 ±5.0	20.9 ±1.0	36 ±7	42.0 ±9.2	17.2 ±0.2	62 ±16	15.3 ±4.2	14.7 ±1.2	29 ±8	30.7 ±9.9	14.0 ±0.5	61 ±22
PLD (2026)	20.0 ±3.5	23.4 ±0.5	25 ±4	37.3 ±1.2	16.1 ±0.8	54 ±3	28.0 ±3.5	6.3 ±3.8	63 ±12	8.0 ±3.5	11.8 ±0.9	15 ±7
ExpertGen (2026)	5.3 ±1.2	16.7 ±3.1	7 ±1	34.7 ±12.2	12.7 ±0.9	53 ±23	34.0 ±9.2	10.5 ±1.9	75 ±26	36.0 ±8.7	8.4 ±0.1	83 ±25
P-RFCL	73.3 ±12.2	**8.4** ±0.9	192 ±60	28.7 ±3.1	**5.1** ±0.2	45 ±6	38.7 ±1.2	**3.4** ±0.0	103 ±5	20.7 ±5.8	4.2 ±0.3	45 ±14
Ours	**85.3** ±6.4	9.3 ±0.7	**253** ±44	**56.7** ±13.0	9.3 ±0.2	**116** ±45	**61.3** ±3.1	4.2 ±0.1	**215** ±20	**44.7** ±4.2	**3.6** ±0.1	**127** ±18

F.5 Qualitative Rollouts

We show how each method solves every task over time, first for the data generators and then for the downstream policies trained on their data. Each figure has one row per data source and one column per time step, from 0 to 28 s for PnP-Banana and StackCube and to 17.5 s for the insertion tasks; the interval is shared by all rows and by both kinds of figures, so a row ends earlier when its method completes the task sooner. When a rollout completes, the next cell shows its final frame together with the time at which it completed; a rollout that does not complete within the shown horizon is marked as a failure in the last column. Full videos are provided in the supplementary material.

Data generators. Figures 14 to 17 roll out each generator from the same fixed initial scene per task as Fig. 4. Each row shows the successful rollout whose length is closest to the average episode length of that generator's dataset (Fig. 5); on StackCube, MP and PLD produced no successful rollout from this scene, so a failed one is shown instead. H20 has no generator and is therefore omitted.

Downstream policies. Figures 18 to 21 show the $\pi_{0.5}$ policies fine-tuned on each dataset, all rolled out in simulation from the same initial state. We evaluate every policy on ten initial states shared by all methods and show, for each task, the one on which the most methods succeed. A row ends when the policy first satisfies the success condition. PLD and ExpertGen are omitted from these figures.

Real robot. Figures 22 to 25 show the same downstream policies on the real robot, including PLD and ExpertGen. For each task we reproduce three initial scenes for every method, and show the one used to compare completion speed in the supplementary video; time starts at the first policy command. A successful run ends at the time it completed, and a run that ends without success is marked as a failure in the next column.

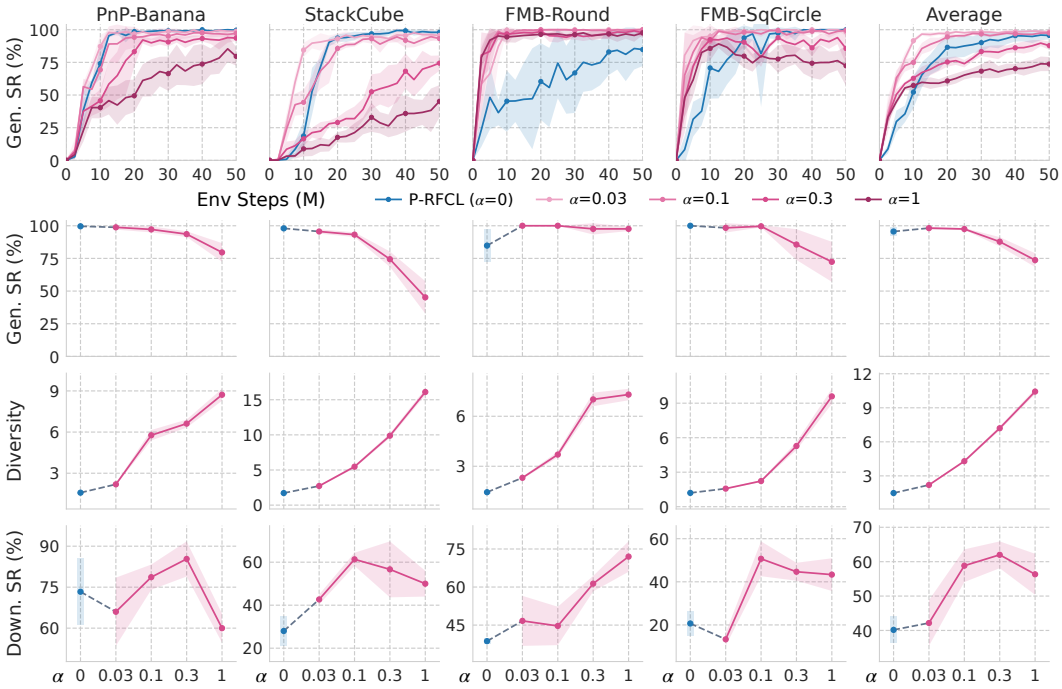


Figure 12: The shape of the trade-off is common to the four tasks, but the α at which generator success starts to give way is not.

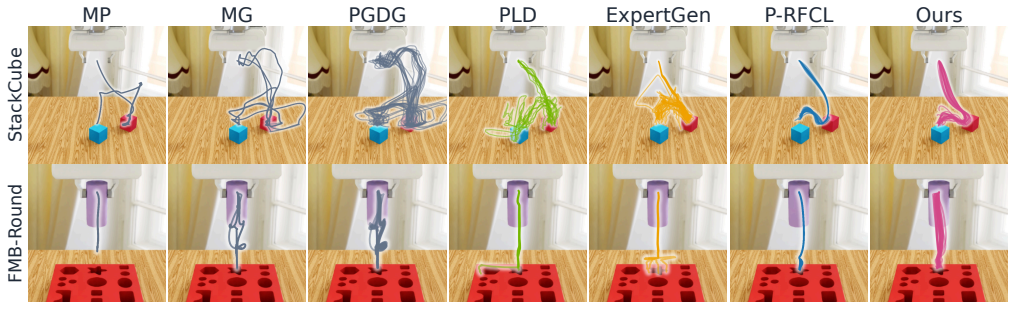


Figure 13: Generated demonstrations on StackCube and FMB-Round, each line one successful demonstration from the same initial placement.

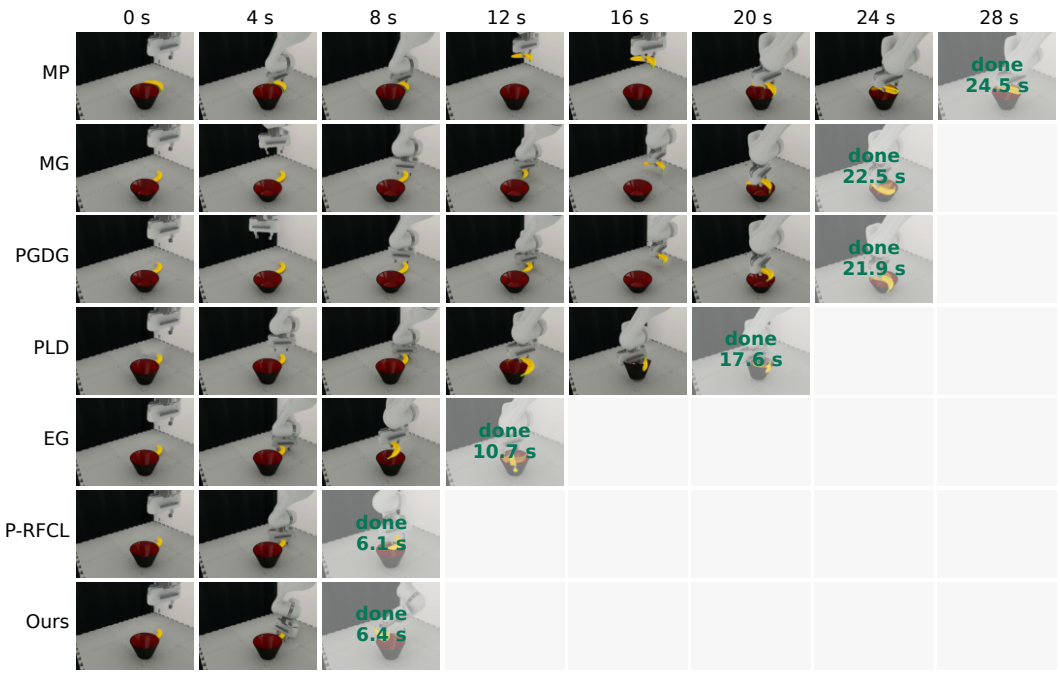


Figure 14: **Data generators on PnP-Banana.** Rollouts from the shared initial scene.

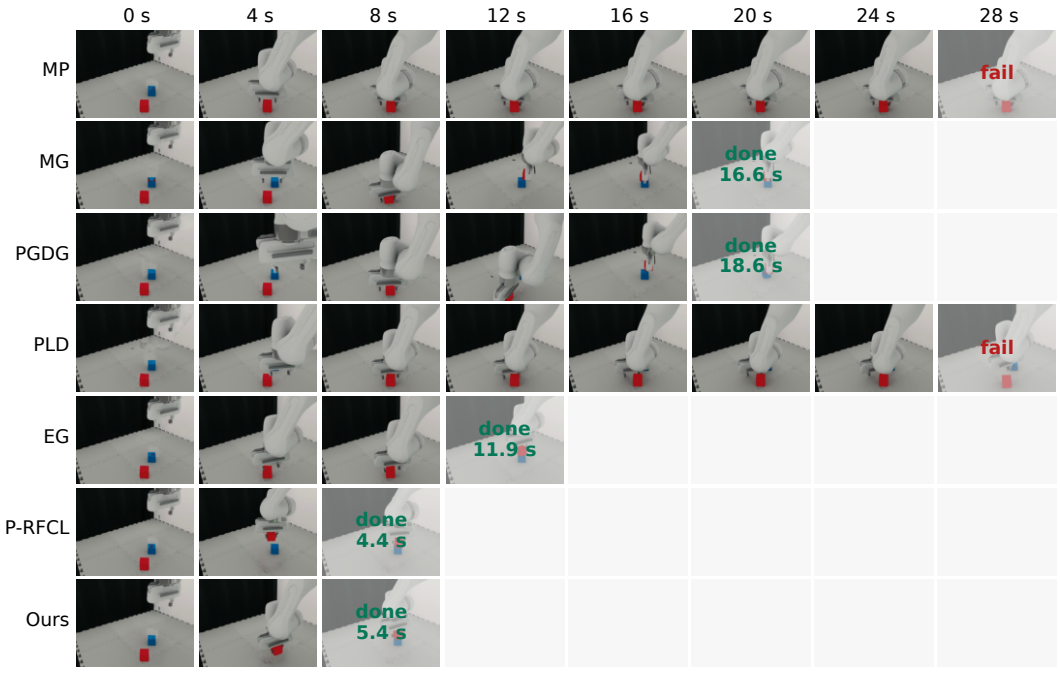


Figure 15: **Data generators on StackCube.** Rollouts from the shared initial scene.

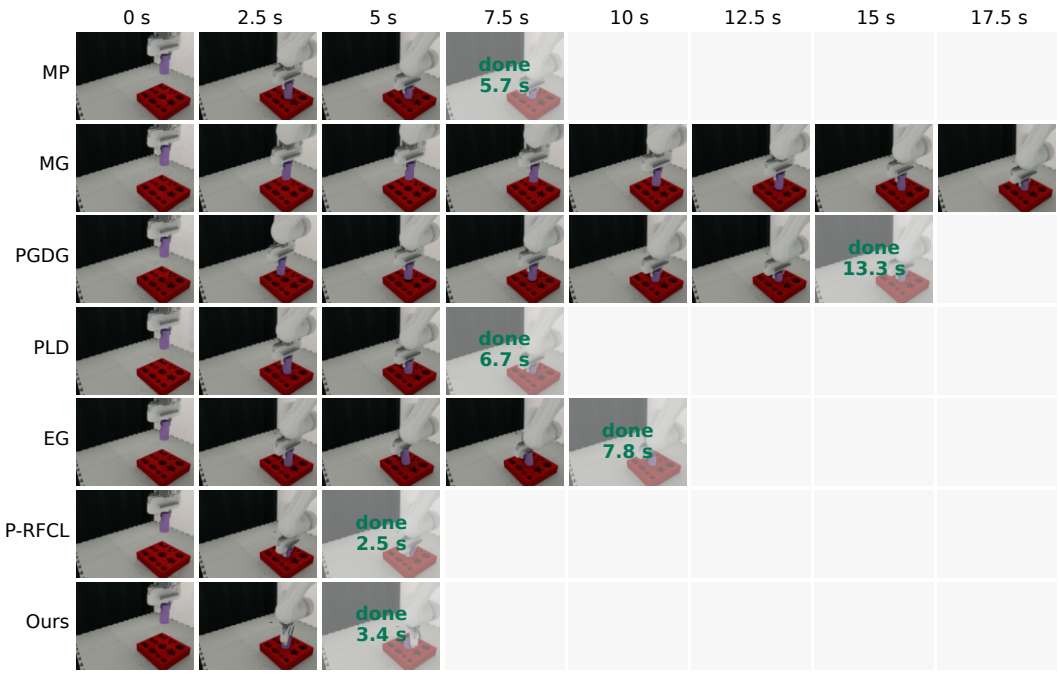


Figure 16: **Data generators on FMB-Round.** Rollouts from the shared initial scene.

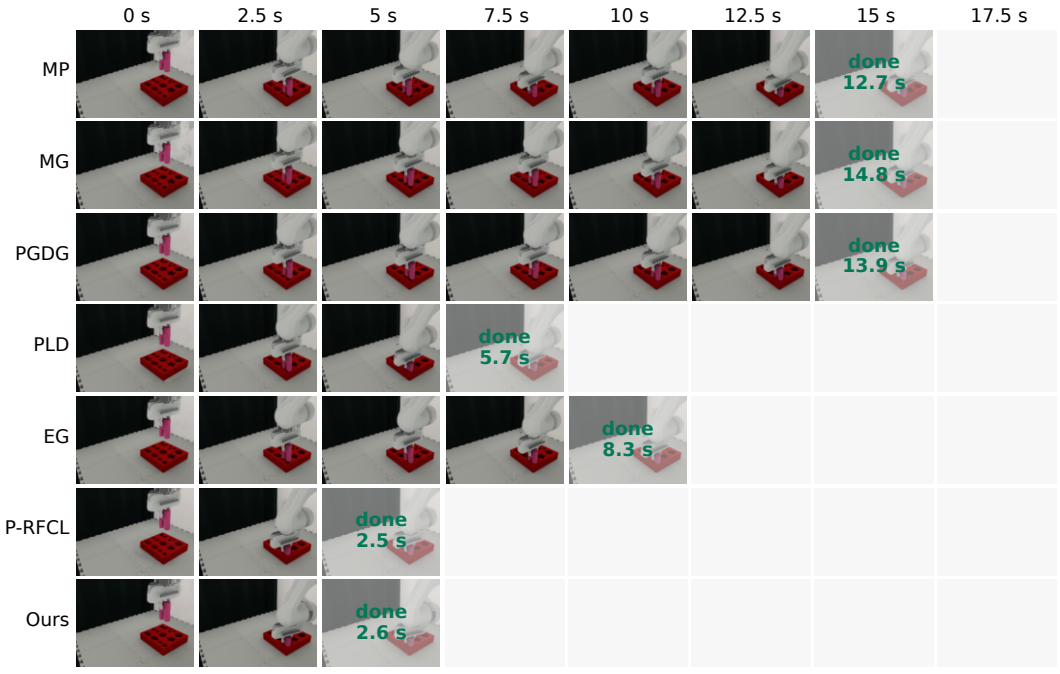


Figure 17: **Data generators on FMB-SqCircle.** Rollouts from the shared initial scene.

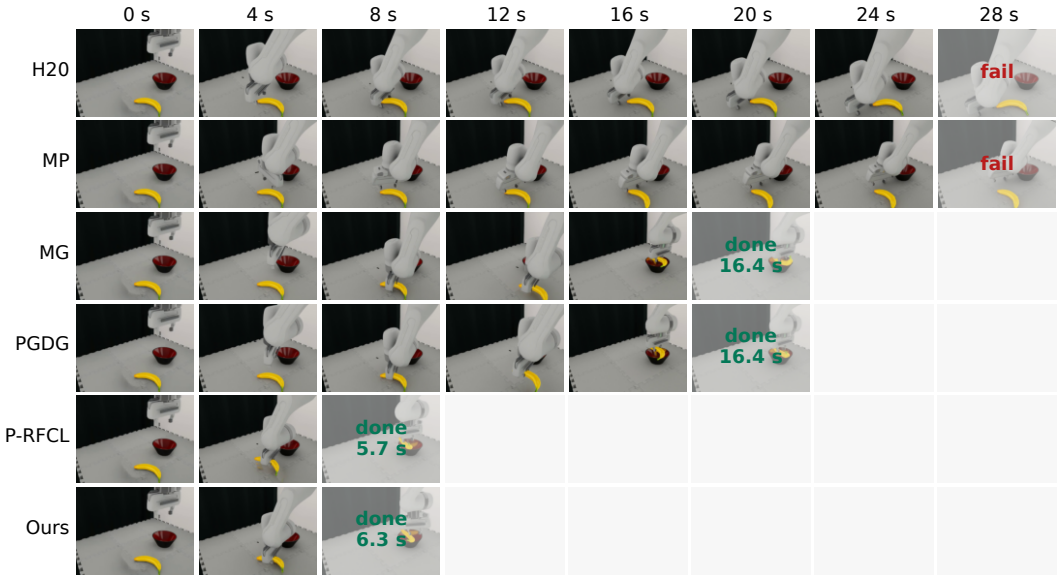


Figure 18: **Downstream policies on PnP-Banana.** Rollouts on the selected shared initial state.

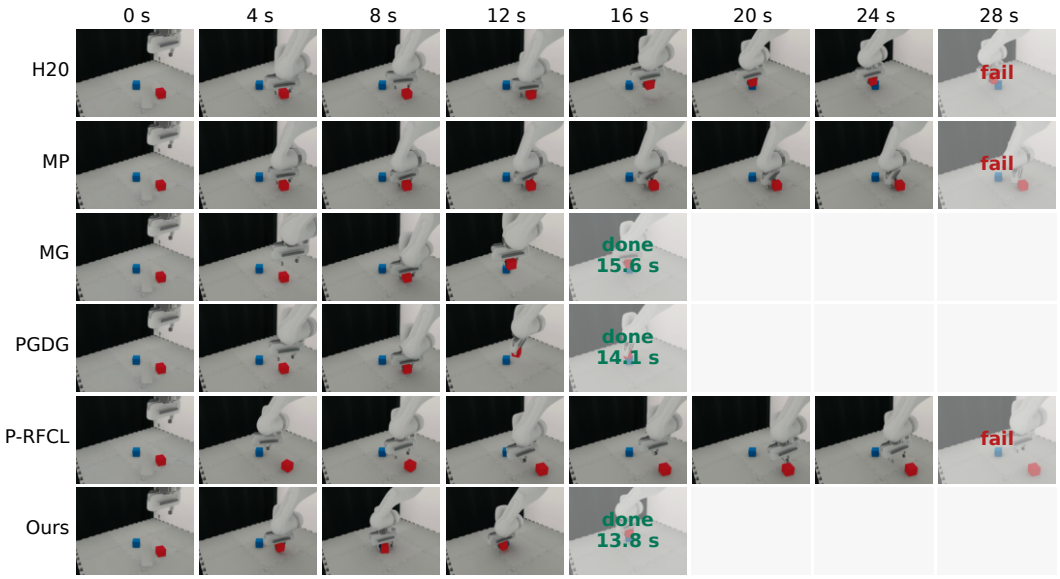


Figure 19: **Downstream policies on StackCube.** Rollouts on the selected shared initial state.

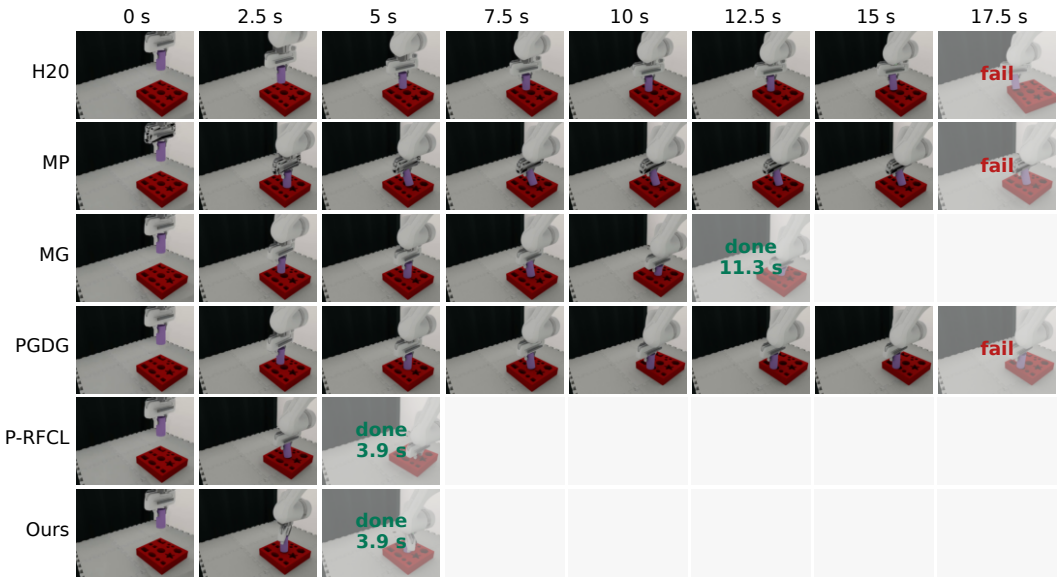


Figure 20: **Downstream policies on FMB-Round.** Rollouts on the selected shared initial state.

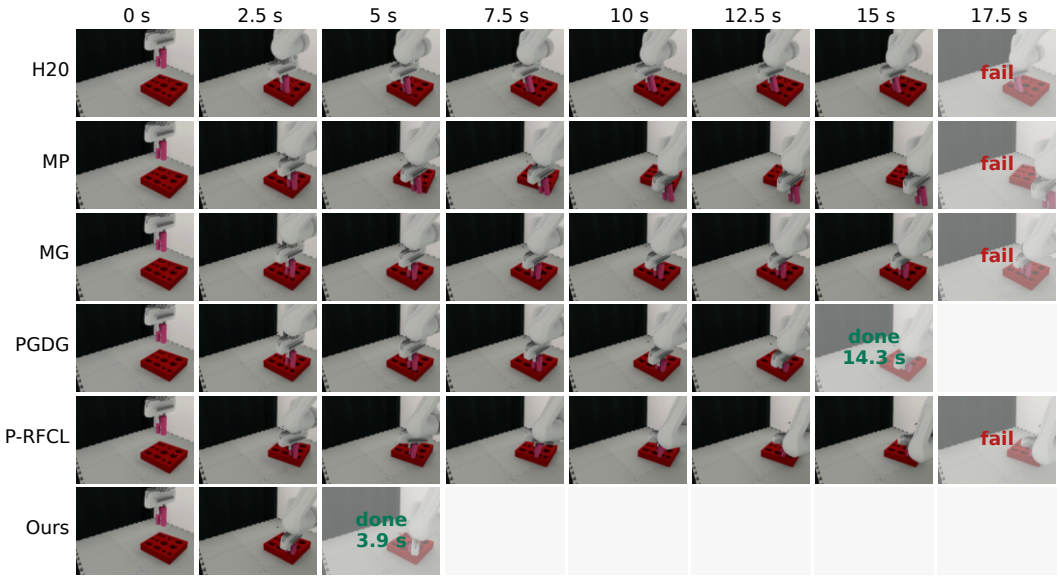


Figure 21: **Downstream policies on FMB-SqCircle.** Rollouts on the selected shared initial state.

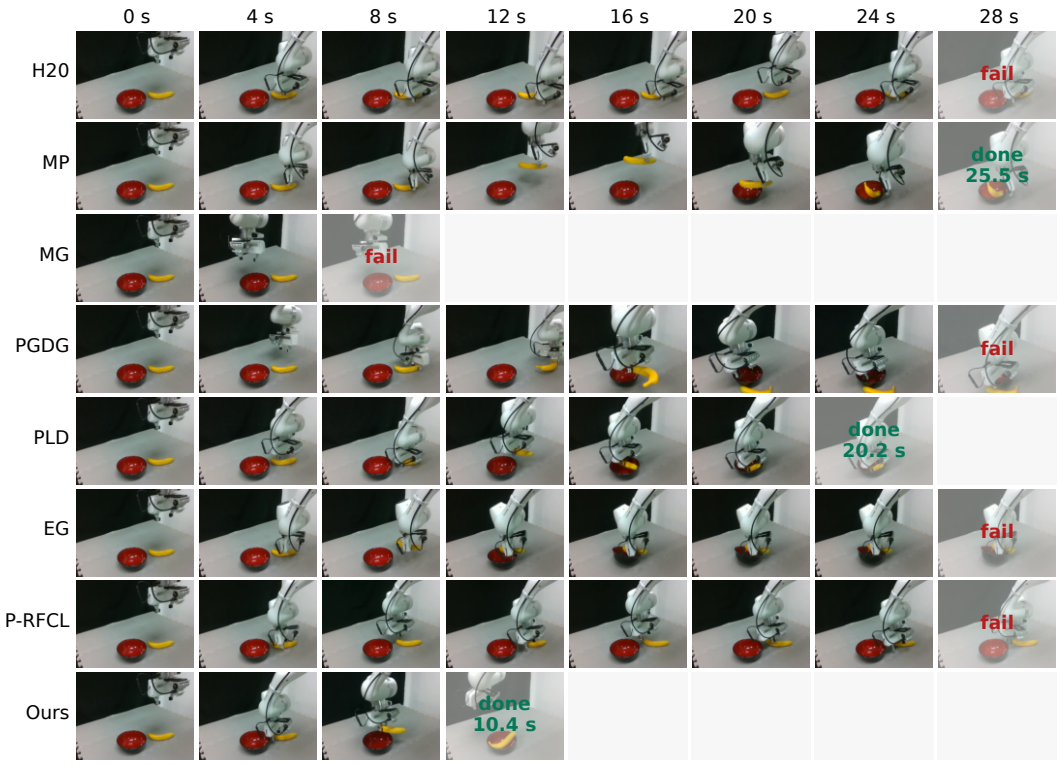


Figure 22: **Real-robot rollouts on PnP-Banana.** Rollouts on the scene used in the supplementary video.

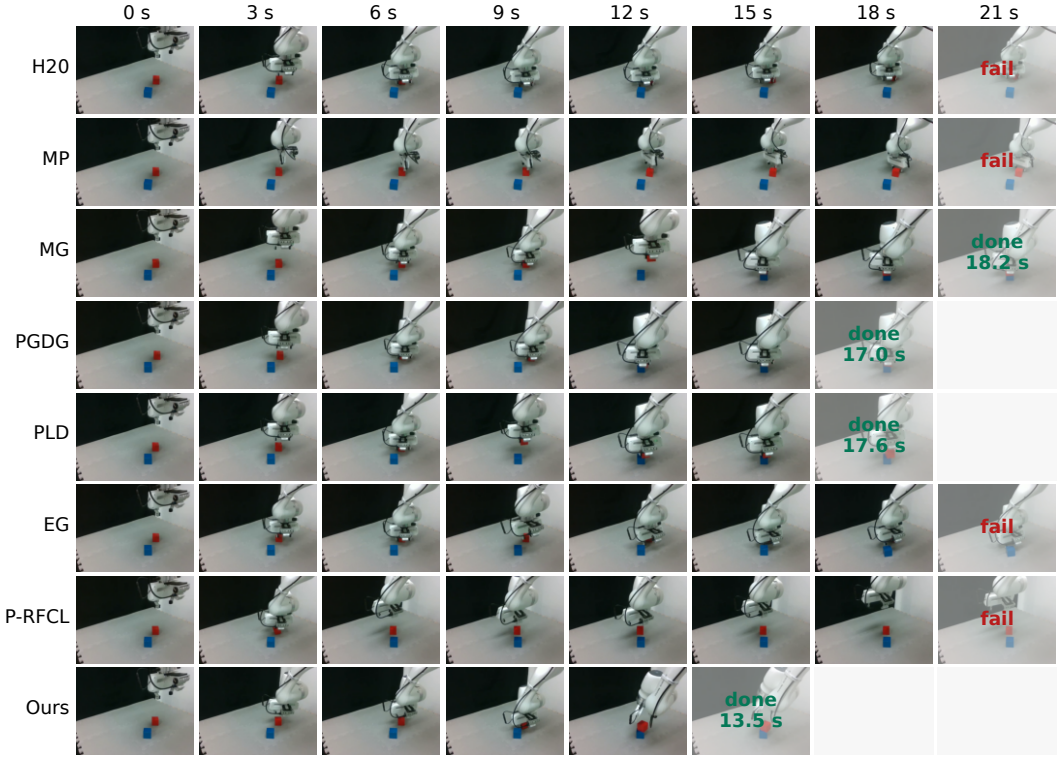


Figure 23: **Real-robot rollouts on StackCube.** Rollouts on the scene used in the supplementary video.

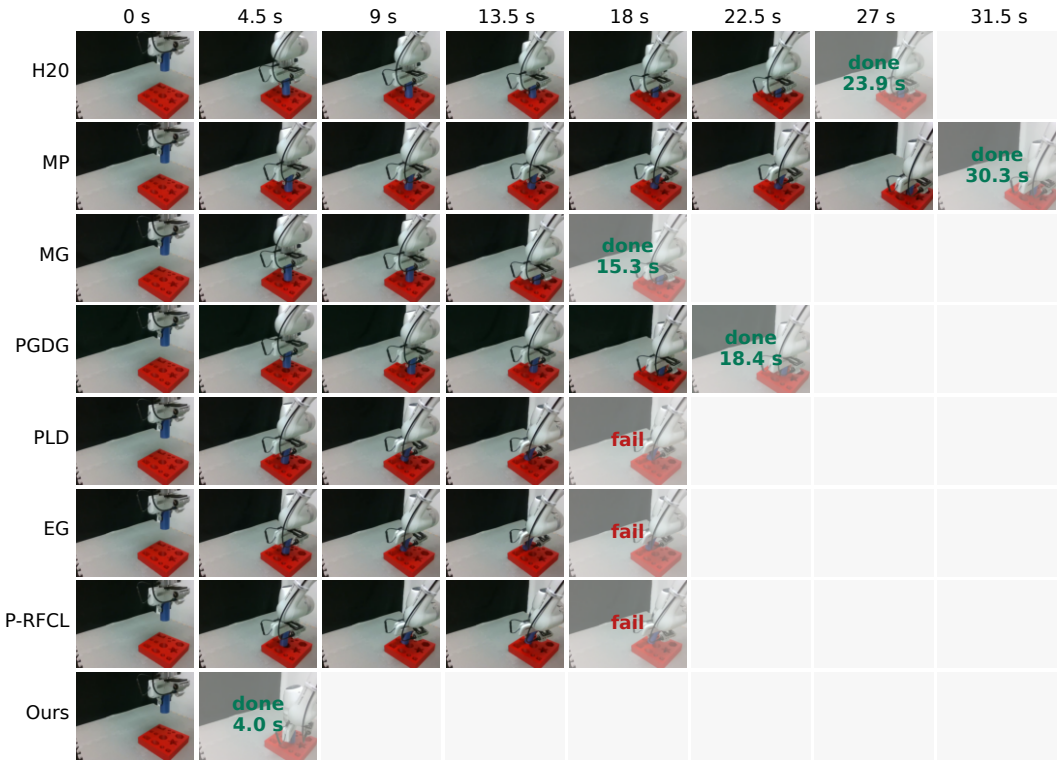


Figure 24: **Real-robot rollouts on FMB-Round.** Rollouts on the scene used in the supplementary video.

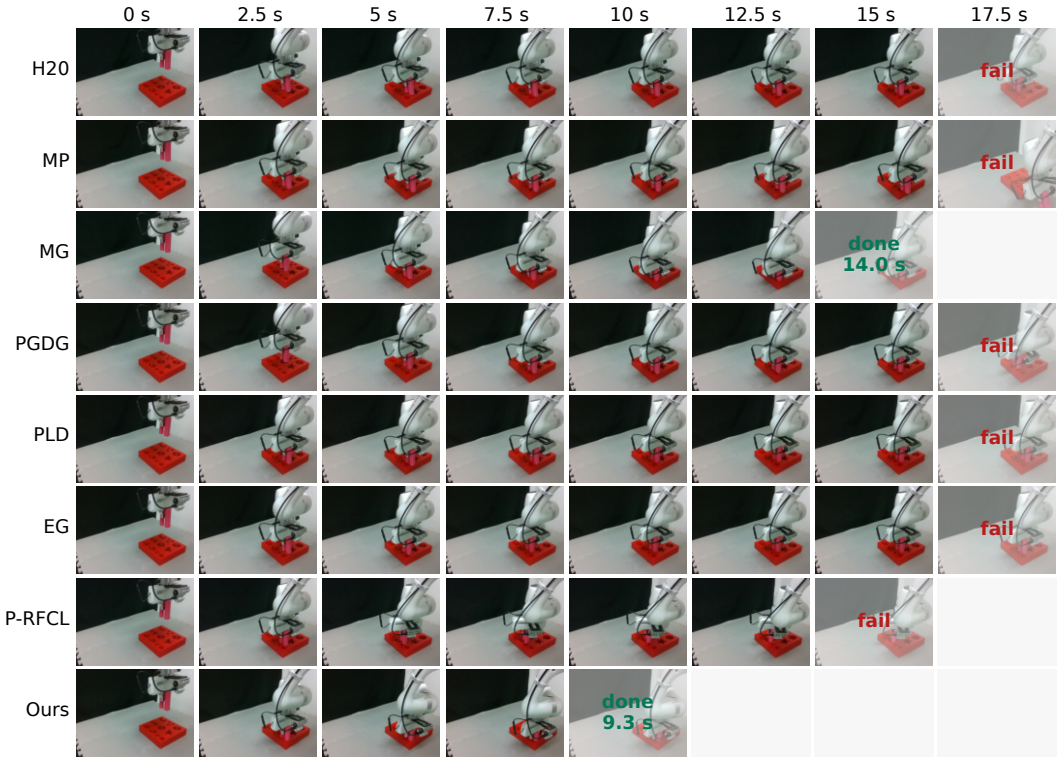


Figure 25: **Real-robot rollouts on FMB-SqCircle.** Rollouts on the scene used in the supplementary video.