

EXPERTGEN: Scalable Sim-to-Real Expert Policy Learning from Imperfect Behavior Priors

Zifan Xu^{1,2}, Ran Gong¹, Maria Vittoria Minniti¹, Kausik Sivakumar¹,
Ahmet Salih Gundogdu¹, Eric Rosen¹, Riedana Yan¹, Tushar Kusnur¹, Zixing Wang¹, Di Deng¹,
Peter Stone², Xiaohan Zhang^{*,1}, Karl Schmeckpeper^{*,1}

¹Robotics and AI Institute, ²University of Texas at Austin, *Equal Advising
pages.rai-inst.com/expertgen

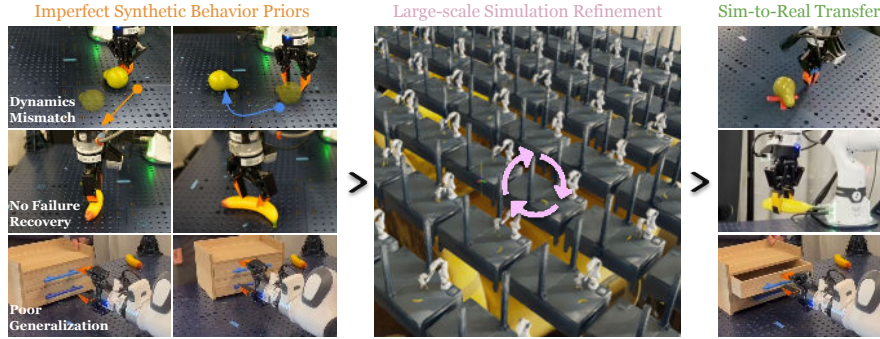


Figure 1: EXPERTGEN pipeline: (left) generative modeling of imperfect behavior priors; (middle) steering prior model in massively parallel simulation using reinforcement learning; and (right) visual distillation in simulation with DAgger for zero-shot sim-to-real transfer.

Abstract: Learning generalizable and robust behavior cloning policies requires large volumes of high-quality robotics data. While human demonstrations (e.g., through teleoperation) serve as the standard source for expert behaviors, acquiring such data at scale in the real world is prohibitively expensive. This paper introduces EXPERTGEN, a framework that automates expert policy learning in simulation without any human demonstrations, while preserving strong sim-to-real transfer performance. EXPERTGEN first initializes a behavior prior in using a diffusion policy trained on synthetic data (e.g., Code as Policy). Reinforcement learning is then used to steer this prior *in massively parallel simulation* toward high task success by optimizing the diffusion model’s initial noise while keeping the denosing process frozen. In this way, EXPERTGEN regularizes exploration to remain within safe behavior manifolds, enabling stable learning from sparse rewards without additional reward engineering. Extensive experiments on challenging manipulation benchmarks demonstrate that EXPERTGEN reliably acquires high-quality policies with strong robustness and generalization. On long-horizon manipulation tasks, EXPERTGEN achieves success rates of up to 85%, while on industrial assembly benchmarks it achieves a 91.1% overall success rate, outperforming all baseline methods. To validate sim-to-real transfer, the learned state-based expert policies are further distilled into visuomotor policies via DAgger and successfully deployed on real hardware.

Keywords: offline-to-online reinforcement learning, synthetic data generation

1 Introduction

Recent progress in synthetic robotics data generation has enabled scalable creation of manipulation demonstrations in simulation. Existing approaches span several paradigms. Demonstration

augmentation methods, such as MimicGen [1], expand a small number of human demonstrations through demonstration perturbation, geometric retargeting, and skill composition [1, 2, 3]. More recent approaches further reduce human effort by generating demonstrations from task-and-motion planning (TAMP) or Code as Policies (CaP) built on reusable motion primitives [4, 5]. Together, these approaches significantly reduce the cost of robotics data generation.

However, simply scaling demonstration quantity does not guarantee robust learned behaviors. Existing synthetic data generation pipelines primarily generate trajectories around nominal successful executions. MimicGen-style approaches rely on heuristically designed perturbation strategies around demonstrations, while TAMP and CaP are constrained by predefined skills and planning heuristics. Consequently, these methods provide limited coverage of failure states and recovery behaviors. In principle, reinforcement learning (RL) can address these limitations by directly optimizing closed-loop task success and discovering recovery behaviors through trial-and-errors. However, large-scale RL-based robotics data generation typically requires carefully engineered dense rewards, which substantially limits scalability across tasks and domains [6, 7]. To address these challenges, this paper proposes to *treat synthetic demonstrations as a scalable behavior prior and subsequently refines this prior through RL using only sparse task-level rewards*.

To this end, this paper introduces EXPERTGEN, a scalable framework for sim-to-real expert policy acquisition from *synthetic behavior priors*. EXPERTGEN first trains a diffusion policy on demonstrations synthesized from scripted policies. These synthetic demonstrations typically encode safe and feasible behaviors while achieving non-trivial task success rates; however, they may still exhibit limited state coverage, incomplete recovery behaviors, or dynamics mismatch. EXPERTGEN then adopts diffusion steering reinforcement learning (DSRL) [8], which optimizes only the diffusion model’s initial noise while keeping the pretrained denoising process frozen. Building on this formulation, EXPERTGEN extends diffusion steering to massively parallel robotics simulation using an off-policy RL framework designed for high-throughput large-batch training. One of our key insights is that the behavior prior need only be *roughly* correct; as long as expert behaviors lie somewhere on the prior manifold, large-scale RL can still efficiently find such expert policies under sparse rewards.

Empirical evaluations on the ANYTASK [4] and AutoMate [9] benchmarks demonstrate that EXPERTGEN consistently improves task success rates across long-horizon manipulation and dexterous assembly tasks, outperforming alternative policy fine-tuning paradigms such as residual RL while requiring only sparse rewards. For sim-to-real transfer, the learned expert policies are distilled into visuomotor policies through large-scale DAgger [10] training with point-cloud or RGB observations under domain randomization, enabling robust zero-shot deployment on real robotic hardware.

The major contributions of the paper are

- We introduce EXPERTGEN, a scalable framework that transforms imperfect synthetic behavior priors into sim-to-real-ready expert policies.
- We showcase that DSRL can be effectively extended to massively parallel simulation, enabling high success, robustness, and generalization, while preserving behavior manifolds.
- We demonstrate robust zero-shot sim-to-real transfer of visuomotor policies distilled from expert teacher policies, including point-cloud- and RGB-based policies.

2 Related Work

Offline-to-Online Reinforcement Learning. A growing body of work studies how to refine policies learned from offline demonstrations using online interaction. Residual RL [11] constrains exploration through residual action spaces and has shown promising results in dexterous manipulation and loco-manipulation [12, 13, 14]. However, these approaches are often sensitive to residual action magnitudes and exploration schedules [15, 16], making them difficult to scale across diverse tasks. More recently, diffusion policies [17] have motivated specialized offline-to-online refinement methods [18, 8, 19, 20, 21, 22]. In particular, diffusion steering reinforcement learning (DSRL) [8, 23] refines behavior by steering the initial diffusion noise while keeping the pretrained denoising process

frozen, preserving the learned action manifold during online optimization. EXPERTGEN builds on this formulation and extends it to massively parallel simulation for scalable expert policy learning.

Synthetic Robotics Data Generation. Another line of work addresses the scalability bottleneck of robotics data collection through synthetic demonstration generation and data augmentation. Prior approaches expand small human datasets through trajectory perturbation, geometric retargeting, skill composition, or reusable motion primitives [1, 2, 3, 24, 25], while more recent methods synthesize demonstrations directly from scripted policies, planning systems, or large language model-generated robot programs [5, 26, 27, 4, 6]. These methods substantially reduce the cost of collecting robotics demonstrations and enable scalable generation of long-horizon and dexterous manipulation data. However, because generated trajectories are typically centered around nominal successful executions, they often provide limited coverage of recovery behaviors, failure states, and out-of-distribution configurations. EXPERTGEN treats synthetic demonstrations as scalable behavior priors and subsequently refines these priors with RL using only sparse task-level rewards.

3 Preliminaries

This section formalizes the learning problem and introduces the notation used throughout the paper.

Markov Decision Process The manipulation tasks are formulated as a Markov decision process (MDP), defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T}(s_{t+1} | s_t, a_t)$ is the transition function, $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, and $\gamma \in (0, 1]$ is the discount factor. The objective is to learn an expert policy π_E that maximizes the expected discounted return: $\max_{\pi} \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^T \gamma^t r(s_t, a_t) \right]$, where $\tau = (s_0, a_0, s_1, a_1, \dots, s_T, a_T)$ denotes a trajectory induced by the policy interacting with the environment.

In this work, the reward function is defined as a sparse binary task success signal. Let $\mathbb{I}_{\text{succ}}(s_t)$ denote an indicator function that evaluates to 1 if the task is successfully completed at state s_t and 0 otherwise. The reward is given by $r(s_t, a_t) = \mathbb{I}_{\text{succ}}(s_t)$, which provides sparse supervision by assigning nonzero reward only upon successful task completion.

Diffusion Policy This paper considers diffusion policy for all imitation learning training. Specifically, diffusion policy considers demonstrations as samples from a diffusion denoising process. Given a state s_t at time step t , the policy represents the conditional distribution of an action chunk [28] $\mathbf{a}_t = a_{t:t+H}$ of length H via a diffusion model [17]. During training, Gaussian noise is progressively added to expert trajectories according to a forward process $q(\mathbf{a}_t^k | \mathbf{a}_t^0) = \mathcal{N}(\mathbf{a}_t^k; \sqrt{\bar{\alpha}_k} \mathbf{a}_t^0, (1 - \bar{\alpha}_k)I)$, where $\bar{\alpha}_k = \prod_{i=1}^k \alpha_i$ defines the noise schedule. The diffusion policy is trained to predict the injected noise using a denoising network ϵ_{θ} , by minimizing the standard diffusion objective:

$$\mathcal{L}_{\text{diff}} = \mathbb{E}_{\mathbf{a}_t^0, s_t, k, \epsilon \sim \mathcal{N}(0, I)} \left[\left\| \epsilon - \epsilon_{\theta}(\mathbf{a}_t^k, s_t, k) \right\|^2 \right]. \quad (1)$$

At inference time, action trajectories are generated by iteratively applying the reverse diffusion process starting from Gaussian noise $\mathbf{a}_t^K \sim \mathcal{N}(0, I)$. To reduce denoising steps, we adopt Denoising Diffusion Implicit Models (DDIM) [29] sampling, which uses deterministic denoising process.

4 EXPERTGEN

This section details EXPERTGEN, a three-stage framework (Fig. 2) for acquiring sim-to-real-ready expert policies from imperfect but scalable synthetic demonstrations. The key idea is to first learn a structured behavior prior from purely synthetic data, then improve this prior through sparse-reward reinforcement learning in simulation, and finally distill the resulting state-based expert into deployable visuomotor policies.

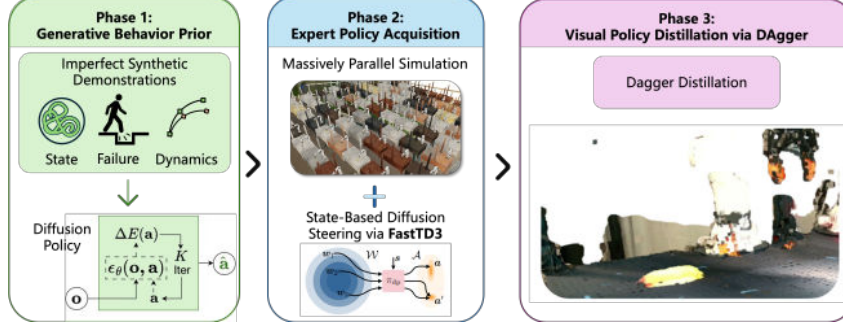


Figure 2: EXPERTGEN training pipeline: generative modeling of imperfect behavior priors using a state-based diffusion policy (Phase 1); steering diffusion policy in massively parallel simulation using FastTD3 (phase 2); visual policy distillation using DAgger from expert teachers (phase 3).

4.1 Generative Behavior Prior Modeling

EXPERTGEN assumes access to a dataset of synthetic demonstration trajectories $\mathcal{D}_{\text{syn}} = \{\tau_i\}_{i=1}^N$. Each trajectory $\tau_i = \{(s_t, a_t)\}_{t=0}^T$ is generated entirely in simulation rather than collected through real-world teleoperation. Specifically, EXPERTGEN considers synthetic data generated from both AnyTask [4] and SkillMimicGen [2] for tabletop manipulation tasks, as well as handcrafted policies for contact-rich assembly tasks.

Although these synthetic demonstrations often exhibit safe and structured behaviors with nontrivial task success, they typically suffer from limited state coverage, weak recovery behaviors, and mismatch between the synthetic generation process and the target task distribution. Consequently, directly cloning such demonstrations does not produce robust expert policies. To model this imperfect behavior prior, EXPERTGEN trains a state-conditioned diffusion policy $\pi_{\text{prior}}(\mathbf{a}_t | s_t)$ on $\mathcal{D}_{\text{syn}}$. The diffusion policy serves as a structured generative prior that captures the motion manifold of the synthetic demonstrations while remaining amenable to RL-based steering.

4.2 Expert Policy Acquisition

This subsection describes how EXPERTGEN acquires task-specialized expert policies by combining diffusion steering with massively parallel simulation and off-policy RL.

Diffusion steering reinforcement learning Let z_t denote the initial noise input to the diffusion model. A steering policy $q_\phi(z_t | s_t)$ selects this noise conditioned on the current state. The diffusion prior $\pi_{\text{prior}}(\mathbf{a}_t | s_t)$ then maps (s_t, z_t) to an action chunk. RL optimizes q_ϕ using sparse task-level rewards, while the frozen diffusion decoder constrains the resulting actions to remain close to the behavior distribution learned from synthetic data.

Massively parallel off-policy RL While the original DSRL formulation focuses on Soft Actor-Critic (SAC) [30], which prioritizes sample efficiency, EXPERTGEN targets a fundamentally different regime: *massively parallel, large-scale simulation training*. To support this setting, EXPERTGEN adopts an off-policy RL training inspired by FastTD3 [31], which integrates several design choices for high-throughput robotic learning in massively parallel simulation.

Specifically, EXPERTGEN incorporates several design choices. First, policy optimization is performed using large-batch updates collected from thousands of parallel environments, substantially improving training throughput and effective learning under sparse rewards. Second, a distributional critic [32] is employed to improve value estimation in long-horizon sparse-reward settings. Third, mixed Gaussian noise [33] is introduced in the steering latent space to maintain exploration diversity across parallel environments while regularizing the steering policy to be close to the normal distribution used to train the base diffusion policy. Empirically, this mixed-noise strategy is critical for preserving the prior behavior manifold after RL steering.

4.3 Visuomotor Policy Distillation through DAgger

The expert policies learned in simulation operate on privileged states that are unavailable at deployment time. To obtain deployable visuomotor policies, the expert is distilled into an observation-based policy $\pi(a \mid o)$ using Dataset Aggregation (DAgger) [10], where observations $o = f(s)$ consist of RGB images or point clouds. To improve sim-to-real transfer, extensive visual domain randomization is applied during distillation, including randomized textures, lighting, object appearances, and camera poses. More details are presented in Sec. 6.5.

5 Experiment Setup

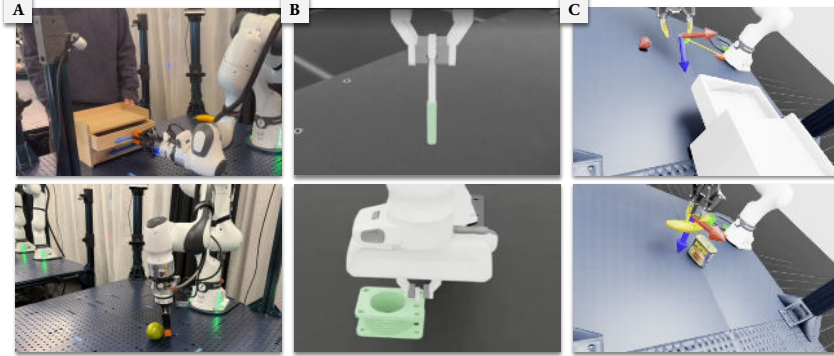


Figure 3: Illustrations of the tasks in our experiments. (A) Real-world manipulation tasks. (B) Industrial assembly tasks from AutoMate. (C) Long-horizon tasks from ANYTASK.

5.1 Benchmarks - ANYTASK

Tasks ANYTASK [4] is an automated framework for task design and large-scale synthetic data generation in robotic manipulation. For benchmarking, we select eight tabletop manipulation tasks spanning a broad range of behaviors, including lifting, pushing, stacking, pick-and-place, and drawer opening. These tasks cover both short-horizon and long-horizon interactions, as well as contact-rich manipulation scenarios. Figure 3 (C) visualizes two tasks from ANYTASK.

Synthetic data The synthetic data is synthesized by AnyTask [4], which uses a LLM to program scripted policies that invokes an existing skill library. We collect 1000 demonstrations for each task from the scripted policies. The scripted policies are listed in Appendix C.1.

Baselines We compare EXPERTGEN against several representative baselines. **Scripted Policy** directly executes the LLM-synthesized scripted policies provided by ANYTASK. **Diffusion Policy** evaluates policies trained purely via behavior cloning on synthetic demonstrations. Offline-to-Online Refinement includes Residual RL [15] and SMP [34], which both initialize from pretrained diffusion priors and improve performance through online RL. We additionally compare against the original **DSRL** formulation [8], which performs diffusion steering using SAC. **ExpertGen-PPO** further replaces off-policy RL with PPO to study the effect of off-policy RL in massively parallel simulation steering. Finally, **FastTD3** trains directly from sparse rewards without demonstrations or pretrained priors, measuring the difficulty of solving these tasks from scratch. More implementation details are provided in the Appendix.

5.2 Benchmarks - AutoMate

Tasks AutoMate [9] is a simulation benchmark for robotic dexterous manipulation in industrial assembly, with a particular emphasis on high-precision peg-insertion tasks. The benchmark consists of a collection of insertion scenarios, as shown in Figure 3 (B), with tight geometric tolerances, demanding accurate pose alignment and dexterous control.

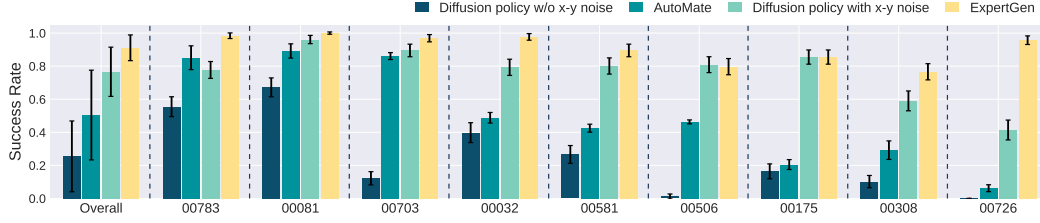


Figure 4: The success rates (%) of the evaluated approaches on assets from the AutoMate benchmark. EXPERTGEN outperforms all other baselines with an overall success of 91.1%. By introducing x - y noise, the diffusion policy demonstrates higher success rates compared to the no x - y noise.

Synthetic data In the AutoMate benchmark, synthetic demonstrations are generated using a scripted disassembly policy that starts from already-assembled configurations and disassemble the parts from the sockets. Then, the disassembly trajectories are reversed and used for assembly policy learning. Because assembly and disassembly dynamics are inherently asymmetric, reversing the trajectories introduces dynamics mismatch and imperfect contact interactions during insertion. We collect 500 demonstrations for each task. Additionally, we consider two variants of the demonstrations depending on whether small noise is injected in the x - y plane during the lifting phase of the disassembly trajectories. This perturbation increases trajectory diversity near insertion regions, producing datasets with and without x - y noise augmentation.

Baselines We compare against two baselines on AutoMate. **Diffusion Policy** trains diffusion policies purely via imitation learning on the reversed demonstrations, without RL refinement. We evaluate variants trained with and without the additional x - y noise augmentation. **AutoMate** reproduces the single-task expert policies proposed in AutoMate using the original benchmark training setup.

5.3 Evaluation Metrics

Success Rate For the ANYTASK and AutoMate benchmark, we report the success rate evaluated over 1024 and 256 rollout trajectories respectively from each baseline approach, with error bars denoting 95% Wilson binomial confidence intervals [35].

Wasserstein-2 Distance To quantify the distributional shift from the base diffusion policy, we measure the Wasserstein-2 distance between the action distributions induced by each evaluated approach and the pretrained diffusion model. For each task from ANYTASK, we randomly sample 5 states from each of 10 demonstration trajectories, resulting in a total of 100 evaluation states. At every sampled state, we generate 512 action samples from the pretrained diffusion policy and 512 action samples from the evaluated method. The empirical Wasserstein-2 distance is then computed between the two action distributions and averaged across all sampled states.

6 Experimental Results

Methods \ Tasks	Lift Banana	Lift Brick	Lift Peach	Open Drawer	Push Pear to Center	Stack Banana on Can	Put Object In Closed Drawer	Place Strawberry In Bowl
Scripted Policy	56.3 $\pm$ 3.1	66.4 $\pm$ 3.0	60.9 $\pm$ 3.1	28.1 $\pm$ 2.8	18.0 $\pm$ 2.4	26.6 $\pm$ 2.7	9.4 $\pm$ 1.8	43.0 $\pm$ 2.9
Diffusion Policy	69.8 $\pm$ 2.8	72.6 $\pm$ 2.7	28.5 $\pm$ 2.8	77.6 $\pm$ 2.6	50.5 $\pm$ 3.1	16.3 $\pm$ 2.3	0.5 $\pm$ 0.5	6.6 $\pm$ 1.5
Residual RL	84.3 $\pm$ 2.0	98.4 $\pm$ 0.8	75.9 $\pm$ 2.6	99.5 $\pm$ 0.5	56.5 $\pm$ 3.0	0.0 $\pm$ 0.2	13.6 $\pm$ 2.1	1.3 $\pm$ 0.7
SMP	0.0 $\pm$ 0.2	0.0 $\pm$ 0.2	0.0 $\pm$ 0.2	99.9 $\pm$ 0.3	2.1 $\pm$ 0.2	0.0 $\pm$ 0.2	0.0 $\pm$ 0.2	1.2 $\pm$ 0.7
DPPO	96.0 $\pm$ 1.0	96.7 $\pm$ 1.1	95.2 $\pm$ 1.3	98.1 $\pm$ 0.9	26.7 $\pm$ 3.2	7.8 $\pm$ 0.8	1.0 $\pm$ 0.7	4.0 $\pm$ 1.4
DSRL	72.4 $\pm$ 2.6	77.8 $\pm$ 2.5	1.2 $\pm$ 0.7	94.9 $\pm$ 1.4	44.8 $\pm$ 3.0	6.5 $\pm$ 1.5	1.2 $\pm$ 0.7	3.0 $\pm$ 1.2
FastTD3	0.0 $\pm$ 0.2	0.0 $\pm$ 0.2	0.0 $\pm$ 0.2	0.0 $\pm$ 0.2	0.0 $\pm$ 0.2	0.4 $\pm$ 0.4	0.0 $\pm$ 0.2	1.5 $\pm$ 0.8
ExpertGen-PPO	89.6 $\pm$ 1.9	90.0 $\pm$ 1.8	81.9 $\pm$ 2.4	96.7 $\pm$ 1.1	69.8 $\pm$ 2.8	57.0 $\pm$ 3.0	54.9 $\pm$ 3.0	44.3 $\pm$ 3.0
ExpertGen (ours)	99.8 $\pm$ 0.3	99.7 $\pm$ 0.4	99.3 $\pm$ 0.0	100 $\pm$ 0.1	83.3 $\pm$ 2.0	67.2 $\pm$ 2.9	80.7 $\pm$ 2.4	52.1 $\pm$ 3.1

Table 1: The success rates (%) of the evaluated state-based policies on ANYTASK benchmark. Bold number indicates the best number across all the approaches.

In this section, we discuss the major results including the expert performance, recovery capacity, trajectory smoothness, and visuomotor policy performance after transferring to the real world.

6.1 Expert Policy Acquisition

Starting from imperfect synthetic behavior priors, EXPERTGEN consistently learns near-perfect expert policies on both benchmarks. On the ANYTASK benchmark (Table 1), EXPERTGEN achieves the highest success rates across all eight evaluation tasks. For short-horizon tasks, success rates approach 100%. Performance on long-horizon tasks is comparatively lower, which is primarily attributed to inherently unsolvable configurations in the benchmark. Failure cases include objects being placed too close to the drawer frame, causing blockage during opening. Despite these challenges, EXPERTGEN consistently outperforms all baseline methods. The strong improvement over DSRL reveals the importance of the design choices in Sec. 4.2 that support massively-parallel simulation. Beyond tabletop manipulation, as shown in Figure 4, EXPERTGEN attains an average success rate of 90.5% across nine high-precision industrial assembly tasks in AutoMate, demonstrating its ability to solve dexterous manipulation tasks from behavior priors with inaccurate contact dynamics.

6.2 Failure Recovery Capacity

Methods	<i>Lift Banana</i>		<i>Open Drawer</i>		<i>Push Pear</i>	
	Open	Force	Open	Force	Open	Force
Diffusion Policy	72.5 (7.6↓)	1.9 (79.9↓)	78.2 (8.7↓)	58.0 (28.9↓)	47.1 (1.9↓)	15.8 (33.1↓)
Residual RL	83.8 (1.0↓)	46.7 (37.5↓)	96.9 (2.6↓)	99.7 (0.2↑)	57.1 (0.7↑)	38.4 (18.1↓)
EXPERTGEN	99.4 (0.3↓)	56.6 (43.1↓)	99.9 (0.0↓)	99.7 (0.2↓)	85.4 (1.2↑)	38.7 (45.3↓)

Table 2: Success rates (%) with two perturbations: random gripper opening and random external force applied to the end-effector. Number in parentheses indicate the performance drop (↓) / increase (↑) compared to no perturbation. Bold number indicates the best success rates.

To evaluate failure recovery capacity, we introduce targeted perturbations to tasks from ANYTASK, including random gripper opening events and random external forces applied to the end-effector in the x - y plane. Under these perturbations, EXPERTGEN exhibits only minor performance degradation (Table 2), with average success-rate drops of 0.5% and 28.6% across three tasks for gripper-opening and external-force perturbations, respectively. In contrast, the base diffusion policy is highly sensitive to such disturbances, with success rates dropping to nearly zero when random external forces are applied. Residual RL suffers from similar amount of performance drop compared to EXPERTGEN, because they are both trained with RL under perturbation. However, EXPERTGEN consistently showcases higher success rate than Residual RL even with the perturbation. These results demonstrate that EXPERTGEN policies possess strong failure recovery capabilities and can provide high-quality supervision for DAgger distillation.

6.3 Preserving the Action Manifold

Tasks \ Methods	<i>Lift Banana</i>	<i>Lift Brick</i>	<i>Lift Peach</i>	<i>Open Drawer</i>	<i>Push Pear to Center</i>	<i>Stack Banana on Can</i>	<i>Put Object In Closed Drawer</i>	<i>Place Strawberry In Bowl</i>
Residual RL	0.23 ± 0.04	0.22 ± 0.02	0.20 ± 0.02	0.22 ± 0.02	0.23 ± 0.01	0.23 ± 0.02	0.24 ± 0.01	0.20 ± 0.02
ExpertGen (ours)	0.08 ± 0.03	0.15 ± 0.04	0.04 ± 0.04	0.09 ± 0.11	0.14 ± 0.05	0.38 ± 0.15	0.39 ± 0.12	0.18 ± 0.08

Table 3: The Wasserstein-2 distance between the evaluated policies and base diffusion policies.

Before sim-to-real transfer, Table 3 evaluates how well different fine-tuning approaches preserve the pretrained diffusion-policy action manifold. Specifically, we measure the Wasserstein-2 distance between the action distributions generated by the learned policy and the base diffusion policy.

We train Residual RL with a relatively small residual action magnitude (0.02), as larger residual actions substantially reduce the initial success rate and lead to unstable learning. Despite remaining closer to the behavior prior, Residual RL still exhibits higher Wasserstein-2 distances on short-horizon tasks and fails to achieve meaningful performance gains on long-horizon tasks, where

success rates remain near zero. In contrast, EXPERTGEN consistently achieves among the lowest Wasserstein-2 distances across tasks while attaining substantially higher task success rates, demonstrating that diffusion steering effectively preserves the pretrained action manifold while enabling effective policy improvement through RL.

6.4 Ablating the mixed Gaussian noise

We claim that a key design choice enabling manifold-preserving steering at scale is the mixed Gaussian exploration noise in the steering latent, with per-environment std drawn from $[\sigma_{\min}, \sigma_{\max}] = [0.005, 0.4]$: its high-noise components force the steering policy to succeed under latents near the $\mathcal{N}(0, I)$ distribution used to train the diffusion prior, implicitly regularizing steered latents to remain in-distribution. To isolate this effect, we shrink the noise range by one and two orders of magnitude on three tasks, reporting final success rate (SR) and Wasserstein-2 distance to the prior:

Tasks $[\sigma_{\min}, \sigma_{\max}]$	Lift banana		Open drawer		Stack banana	
	SR $\uparrow$	$W_2\downarrow$	SR $\uparrow$	$W_2\downarrow$	SR $\uparrow$	$W_2\downarrow$
$[0.005, 0.4]$ (ours)	99.8	0.08	100	0.09	67.2	0.38
$[0.001, 0.01]$	99.8	0.37	99.6	0.19	75.9	0.65
$[0.0001, 0.002]$	99.6	1.09	100	0.50	56.4	0.84

Reducing the noise range removes this regularization: asymptotic success increases slightly as RL exploits the latent space more freely, but W_2 drastically grows by 2-10 $\times$ — the expert drifts off the prior manifold and mitigate the sim-to-real transfer capacity. Thus our mixed range attains comparable success while staying closest to the prior, directly substantiating the Sec. 4.2 claim and showing that the contribution is not a backbone swap but the design choices that make noise-space steering simultaneously scalable and manifold-preserving.

6.5 Sim-to-Real Transfer of the Visuomotor Policies

Observation	Method	Number of Demos.	Lift Banana	Lift Brick	Push Pear to Center	Open Drawer
Point-cloud	EXPERTGEN	Dagger	75.0%	—	65.0%	85.0%
	ANYTASK [4]	20k	73.3%	—	16.7%	42.5%
RGB	EXPERTGEN	20k	80.0%	65.0%	45.0%	65.0%
	ANYTASK [4]	20k	80.0%	55.0%	35.0%	10.0%

Table 4: Real-world evaluation results of visuomotor policies. Policies are trained from either EXPERTGEN or ANYTASK [4] scripted-policy rollouts. While ANYTASK achieve reasonable performance on simple pick tasks, they struggle in contact-rich manipulation, highlighting the importance of EXPERTGEN’s RL-refined teacher policies for robust sim-to-real transfer.

In this section, we present real-world results for EXPERTGEN on selected tasks from ANYTASK. Details of the real-world setup can be found in the Appendix C.3.

Real-world transfer of visuomotor policies EXPERTGEN teacher policies are distilled into point-cloud-based policies using DAgger and into RGB-based policies using standard behavioral cloning (BC). These policies are compared against counterparts trained from ANYTASK [4] scripted-policy demonstrations. For RGB policies, training uses 20k offline demonstrations with extensive visual domain randomization [36, 37], including randomized camera extrinsics, dome light textures, object textures, table textures, and HSV jitter. The RGB policy takes 224×224 RGB images together with end-effector poses as input. All methods use the same policy architecture and domain randomization strategy to ensure a fair comparison. As shown in Table 4, while both approaches achieve comparable performance on simple pick-and-place tasks, EXPERTGEN consistently outperforms the ANYTASK on tasks involving articulation or complex contact dynamics, such as *Push Pear to Center* and *Open Drawer*. These results suggest that RL-refined teacher policies are critical for providing supervision for complex interactions that are difficult to capture from scripted policies.

DAgger distillation To verify the necessity of DAgger training, we ablate the schedule of the teacher rollout ratio α on *Lift Banana* task. Specifically, α is linearly annealed from 1.0 to a final

value between 50k and 100k global training steps, and training continues until 200k steps. While simulation success rates remain similar across different schedules, real-world performance exhibits a clear sweet spot at a low, non-zero final α . The highest success rate of 75.0% is achieved with a final $\alpha = 0.2$, compared to 65.0% for pure student training ($\alpha = 0.0$), 63.3% for $\alpha = 0.4$, and 55.0% for maintaining full teacher influence ($\alpha = 1.0$). These results suggest that retaining a small amount of teacher correction improves robustness to real-world noise, while both fully removing the teacher and excessively relying on teacher rollouts degrade performance.

7 Conclusion and Limitations

This paper introduced EXPERTGEN, a scalable framework for sim-to-real expert policy acquisition from imperfect synthetic behavior priors. EXPERTGEN first learns a structured diffusion-policy prior from synthetic demonstrations and subsequently refines this prior through diffusion steering reinforcement learning in massively parallel simulation using only sparse task-level rewards. Across long-horizon manipulation and dexterous assembly benchmarks, EXPERTGEN consistently improves robustness, recovery behaviors, and task success rates compared to the pretrained policy while preserving the action manifold. Furthermore, the resulting expert policies can be distilled into visuomotor policies through DAgger and transferred zero-shot to real robotic hardware. These results indicate that *learning from imperfect synthetic priors and refining them through large-scale sim-to-real RL is a scalable strategy for robotics expert policy acquisition*. The main limitation of EXPERTGEN lies in that the performance is bounded by the quality and diversity of the synthetic behavior priors. While RL steering can substantially improve robustness and recovery behaviors, extremely poor or highly biased priors may limit the performance and exploration coverage.

Acknowledgments

This work is done by Zifan Xu during his internship at Robotics and AI Institute. This work has also partly taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by the National Science Foundation (FAIN-2019844, NRT-2125858, OIC-2535195), the Office of Naval Research (N00014-24-1-2550), Army Research Office (W911NF-17-2-0181, W911NF-23-2-0004, W911NF-25-1-0065), Lockheed Martin, Lyda Hill Philanthropies, and Good Systems, a research grand challenge at the University of Texas at Austin. The views and conclusions contained in this document are those of the authors alone. Peter Stone serves as the Chief Scientist of Sony Corporate Technology Center and receives financial compensation for that role. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

References

- [1] A. Mandlekar, S. Nasiriany, B. Wen, I. Akinola, Y. Narang, L. Fan, Y. Zhu, and D. Fox. Mimicgen: A data generation system for scalable robot learning using human demonstrations. *arXiv preprint arXiv:2310.17596*, 2023.
- [2] C. Garrett, A. Mandlekar, B. Wen, and D. Fox. Skillmimicgen: Automated demonstration generation for efficient skill learning and deployment. *arXiv preprint arXiv:2410.18907*, 2024.
- [3] Z. Jiang, Y. Xie, K. Lin, Z. Xu, W. Wan, A. Mandlekar, L. J. Fan, and Y. Zhu. Dexmimicgen: Automated data generation for bimanual dexterous manipulation via imitation learning. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16923–16930. IEEE, 2025.
- [4] R. Gong*, X. Zhang*, J. Shang*, M. V. Minniti*, J. Patel, V. Pepe, R. Yan, A. Gundogdu, I. Kapelyukh, A. Abbas, X. Yan, H. Patel, L. Herlant, and K. Schmeckpeper. Anytask: an automated task and data generation framework for advancing sim-to-real policy learning. *arXiv preprint arXiv:2512.17853*, 2025.
- [5] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022.
- [6] P. Hua, M. Liu, A. Macaluso, Y. Lin, W. Zhang, H. Xu, and L. Wang. Gensim2: Scaling robot data generation with multi-modal and reasoning llms. In *8th Annual Conference on Robot Learning*, 2024.
- [7] S. Tao, F. Xiang, A. Shukla, Y. Qin, X. Hinrichsen, X. Yuan, C. Bao, X. Lin, Y. Liu, and T. kai Chan et al. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *Robotics: Science and Systems*, 2025.
- [8] A. Wagenmaker, M. Nakamoto, Y. Zhang, S. Park, W. Yagoub, A. Nagabandi, A. Gupta, and S. Levine. Steering your diffusion policy with latent space reinforcement learning. *arXiv preprint arXiv:2506.15799*, 2025.
- [9] B. Tang, I. Akinola, J. Xu, B. Wen, A. Handa, K. Van Wyk, D. Fox, G. S. Sukhatme, F. Ramos, and Y. Narang. Automate: Specialist and generalist assembly policies over diverse geometries. *arXiv preprint arXiv:2407.08028*, 1(2), 2024.
- [10] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

- [11] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.
- [12] L. Ankile, A. Simeonov, I. Shenfeld, M. Torne, and P. Agrawal. From imitation to refinement-residual rl for precise assembly. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 01–08. IEEE, 2025.
- [13] S. Zhao, Y. Ze, Y. Wang, C. K. Liu, P. Abbeel, G. Shi, and R. Duan. Resmimic: From general motion tracking to humanoid whole-body loco-manipulation via residual learning. *arXiv preprint arXiv:2510.05070*, 2025.
- [14] Y. Wang, S. Zhu, P. Zhi, Y. Li, J. Li, Y.-L. Li, Y. Xiao, X. Wang, B. Jia, and S. Huang. Omnixtreme: Breaking the generality barrier in high-dynamic humanoid control. *arXiv preprint arXiv:2602.23843*, 2026.
- [15] X. Yuan, T. Mu, S. Tao, Y. Fang, M. Zhang, and H. Su. Policy decorator: Model-agnostic online refinement for large policy model. *arXiv preprint arXiv:2412.13630*, 2024.
- [16] L. Dodeja, K. Schmeckpeper, S. Vats, T. Weng, M. Jia, G. Konidaris, and S. Tellex. Accelerating residual reinforcement learning with uncertainty estimation. *arXiv preprint arXiv:2506.17564*, 2025.
- [17] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.
- [18] A. Z. Ren, J. Lidard, L. L. Ankile, A. Simeonov, P. Agrawal, A. Majumdar, B. Burchfiel, H. Dai, and M. Simchowitz. Diffusion policy policy optimization. *arXiv preprint arXiv:2409.00588*, 2024.
- [19] B. Yi, H. Choi, H. G. Singh, X. Huang, T. E. Truong, C. Sferrazza, Y. Ma, R. Duan, P. Abbeel, G. Shi, et al. Flow policy gradients for robot control. *arXiv preprint arXiv:2602.02481*, 2026.
- [20] D. McAllister, S. Ge, B. Yi, C. M. Kim, E. Weber, H. Choi, H. Feng, and A. Kanazawa. Flow matching policy gradients. *arXiv preprint arXiv:2507.21053*, 2025.
- [21] L. Lyu, Y. Li, Y. Luo, F. Sun, T. Kong, J. Xu, and X. Ma. Flow-based policy for online reinforcement learning. *Advances in Neural Information Processing Systems*, 38:93967–93990, 2026.
- [22] O. Celik, Z. Li, D. Blessing, G. Li, D. Palenicek, J. Peters, G. Chalvatzaki, and G. Neumann. Dime: Diffusion-based maximum entropy reinforcement learning. *arXiv preprint arXiv:2502.02316*, 2025.
- [23] O. Patil, O. Biza, T. Weng, K. Schmeckpeper, W. Thomason, X. Zhang, K. Sivakumar, R. Walters, N. Gopalan, S. Castro, et al. You’ve got a golden ticket: Improving generative robot policies with a single noise vector. *arXiv preprint arXiv:2603.15757*, 2026.
- [24] Z. Xue, S. Deng, Z. Chen, Y. Wang, Z. Yuan, and H. Xu. Demogen: Synthetic demonstration generation for data-efficient visuomotor policy learning. *arXiv preprint arXiv:2502.16932*, 2025.
- [25] Z. Zhou, A. Garg, A. Mandlekar, and C. Garrett. Reinforcegen: Hybrid skill policies with automated data generation and reinforcement learning. *arXiv preprint arXiv:2512.16861*, 2025.
- [26] T. Chen, Z. Chen, B. Chen, Z. Cai, Y. Liu, Q. Liang, Z. Li, X. Lin, Y. Ge, Z. 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*, 2025.

- [27] Y. Tian, Y. Yang, Y. Xie, Z. Cai, X. Shi, N. Gao, H. Liu, X. Jiang, Z. Qiu, F. Yuan, et al. Interndata-a1: Pioneering high-fidelity synthetic data for pre-training generalist policy. *arXiv preprint arXiv:2511.16651*, 2025.
- [28] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.
- [29] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.
- [30] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. Pmlr, 2018.
- [31] Y. Seo, C. Sferrazza, H. Geng, M. Nauman, Z.-H. Yin, and P. Abbeel. Fasttd3: Simple, fast, and capable reinforcement learning for humanoid control. *arXiv preprint arXiv:2505.22642*, 2025.
- [32] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages 449–458. PMLR, 2017.
- [33] Z. Li, T. Chen, Z.-W. Hong, A. Ajay, and P. Agrawal. Parallel q -learning: Scaling off-policy reinforcement learning under massively parallel simulation. In *International Conference on Machine Learning*, pages 19440–19459. PMLR, 2023.
- [34] Y. Mu, Z. Zhang, Y. Shi, M. Matsumoto, K. Imamura, G. Tevet, C. Guo, M. Taylor, C. Shu, P. Xi, et al. Smp: Reusable score-matching motion priors for physics-based character control. *arXiv preprint arXiv:2512.03028*, 2025.
- [35] Wikipedia contributors. Binomial proportion confidence interval — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Binomial_proportion_confidence_interval&oldid=1333177192, 2026. [Online; accessed 31-January-2026].
- [36] R. Singh, A. Allshire, A. Handa, N. Ratliff, and K. V. Wyk. Dextrah-rgb: Visuomotor policies to grasp anything with dexterous hands, 2025. URL <https://arxiv.org/abs/2412.01791>.
- [37] P. Yin, T. Westenbroek, Z. Zhang, J. Tran, I. Dagnino, E. Shilamkar, N. Mbiziwo-Tiapo, S. Bagaria, X. Liu, G. Mullins, A. Kolobov, and A. Gupta. Emergent dexterity via diverse resets and large-scale reinforcement learning, 2026. URL <https://arxiv.org/abs/2603.15789>.
- [38] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

A Additional Analysis

A.1 Imperfect Synthetic Behavior Priors

The scripted policies provide incomplete state coverage, which directly limits the quality of the learned diffusion policy. Consequently, the resulting behavior priors are imperfect and exhibit low success rates for certain initial configurations, as shown in Figure 5. This trend is also consistent on the AutoMate benchmark, where the diffusion policy without added x - y noise achieves the lowest success rates among all evaluated methods. Injecting x - y noise improves spatial coverage and partially mitigates this issue; however, the resulting policy remains suboptimal. Furthermore, Table 3 shows that the diffusion policy lacks strong failure recovery capability, as evidenced by the substantial performance degradation under external perturbations.

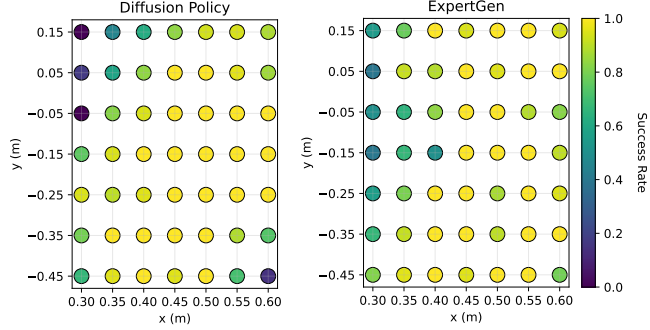


Figure 5: Success rate (%) distribution of imperfect prior diffusion policy (left) and EXPERTGEN (right) over different initial object configurations for *Push Pear to Center*.

To further isolate the impact of imperfect priors, we conduct two controlled experiments summarized in Table 5. First, to study limited state coverage, the *Lift Banana* task is trained using demonstrations in which the initial banana pose is restricted to only the left half of the workspace. Under this setting, the base diffusion policy experiences a substantial performance degradation outside the demonstrated region, whereas EXPERTGEN is able to partially extrapolate beyond the demonstrated initial-state distribution, achieving over 50% success. Second, to evaluate robustness to dynamics mismatch, the prismatic drawer-joint friction in the *Open Drawer* task is increased from 0.2 to 0.5, requiring a firmer grasp during drawer opening. While this modification significantly degrades the base diffusion policy, EXPERTGEN exhibits only negligible performance drop after steering. This robustness to dynamics mismatch is consistent with the AutoMate experiments, where priors generated by reversing disassembly trajectories contain substantial dynamics discrepancies relative to the assembly process. Overall, these results demonstrate that EXPERTGEN can effectively refine imperfect synthetic behavior priors by improving spatial generalization, robustness, and recovery behaviors through reinforcement learning.

Table 5: Success rates under prior deficiencies. Arrows indicate performance drops compared to the setting without prior deficiency.

Deficiencies	Base Policy	EXPERTGEN
Limited state coverage	39.2 (30.6↓)	76.8 (23.2↓)
Joint dynamics mismatch	31.4 (41.2↓)	99.2 (0.1↓)

A.2 Number of Demonstrations

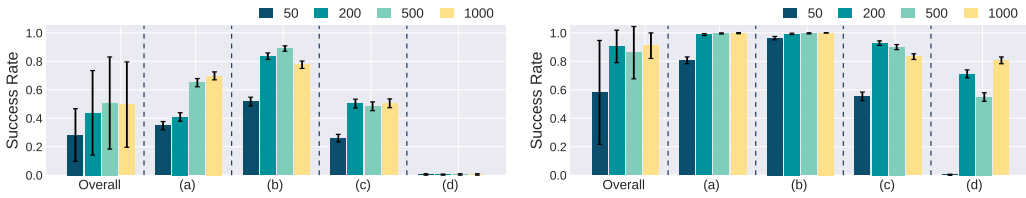


Figure 6: Success rates of base diffusion policies (left) and the EXPERTGEN policies (right) trained with 50, 200, 500, and 1000 imperfect demonstrations on (a) *Lift Banana*, (b) *Open Drawer*, (c) *Push Pear to Center*, and (d) *Put Object in Closed Drawer*.

We further examine how many demonstrations are required to learn a behavior prior that is sufficiently diverse to support effective refinement in simulation. Figure 6 reports the success rates of both the base diffusion policies and the EXPERTGEN policies trained with 50, 200, 500, and 1000 imperfect demonstrations on four ANYTASK tasks. Performance remains largely unchanged once the dataset contains more than 200 demonstrations. A noticeable performance drop occurs only when the dataset is reduced to 50 demonstrations, where the average success rate decreases to 58.2%. These results suggest that EXPERTGEN does not require large-scale demonstration datasets; instead, a relatively small number of imperfect demonstrations (approximately 200) is sufficient to initialize a behavior prior that can be effectively refined through reinforcement learning.

A.3 Choice of RL algorithms

We adopt FastTD3 as the underlying RL algorithm for expert policy acquisition. As shown in Table 1, while ExpertGen-PPO achieves reasonable success rates on several tasks, it consistently underperforms FastTD3 across all eight tasks in the ANYTASK benchmark. FastTD3’s off-policy formulation enables more efficient reuse of expensive action-chunked rollouts and more stable optimization under sparse, task-level rewards, both of which are essential to the success of EXPERTGEN.

In addition, preliminary experiments with the original DSRL formulation based on Soft Actor-Critic (SAC) exhibit substantially lower asymptotic performance under the extensive domain randomization and perturbation settings used in EXPERTGEN.

A.4 Human Motions as Behavior Priors

While the main results use scripted policies for generating synthetic data, human demonstrations may provide richer motion diversity and more adaptive behaviors. To evaluate this hypothesis, we apply SkillMimicGen [2] to *Stack Banana on Can* task, on which EXPERTGEN trained with scripted policies achieves a relatively low success rate (67%). Specifically, six human teleoperation trajectories are collected in IsaacLab and subsequently augmented to 1,000 demonstrations using SkillMimicGen. A single-task diffusion policy is then trained on this dataset, while all other components of the EXPERTGEN pipeline remain unchanged. This variant is referred to as EXPERTGEN w/ SkillMimicGen.

The results, summarized in Figure 7, show that both the diffusion policy and EXPERTGEN trained with SkillMimicGen data substantially outperform their counterparts trained using scripted policy demonstrations. These results suggest that human motion priors provide more diverse and adaptable behavioral patterns, which improve both imitation learning and subsequent RL refinement.

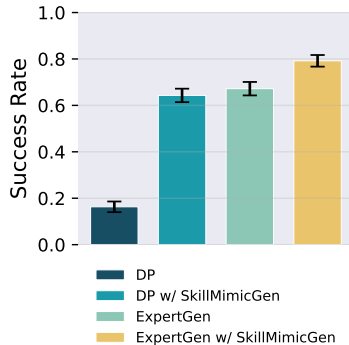


Figure 7: Success rates (%) of the evaluated state-based policies on the *Stack Banana on Can* task. Diffusion Policy w/ SkillMimicGen trains a diffusion policy on human demonstrations augmented with SkillMimicGen [2], while EXPERTGEN w/ SkillMimicGen further refines the diffusion policy trained on SkillMimicGen data. Using human motions as behavior priors significantly improves the performance of both the diffusion policy and EXPERTGEN.

B Hyper-parameters

B.1 Diffusion Policy

This paper considers diffusion policy for all imitation learning training. Specifically, diffusion policy considers demonstrations as samples from a diffusion denoising process. Given a state s_t at time step t , the policy represents the conditional distribution of an action chunk [28] $\mathbf{a}_t = a_{t:t+H}$ of length H via a diffusion model [17]. During training, Gaussian noise is progressively added to expert trajectories according to a forward process

$$q(\mathbf{a}_t^k | \mathbf{a}_t^0) = \mathcal{N}(\mathbf{a}_t^k; \sqrt{\bar{\alpha}_k} \mathbf{a}_t^0, (1 - \bar{\alpha}_k)I), \quad (2)$$

where $\bar{\alpha}_k = \prod_{i=1}^k \alpha_i$ defines the noise schedule. The diffusion policy is trained to predict the injected noise using a denoising network ϵ_θ , by minimizing the standard diffusion objective:

$$\mathcal{L}_{\text{diff}} = \mathbb{E}_{\mathbf{a}_t^0, s_t, k, \epsilon \sim \mathcal{N}(0, I)} \left[\|\epsilon - \epsilon_\theta(\mathbf{a}_t^k, s_t, k)\|^2 \right]. \quad (3)$$

At inference time, action trajectories are generated by iteratively applying the reverse diffusion process starting from Gaussian noise $\mathbf{a}_t^K \sim \mathcal{N}(0, I)$. To reduce denoising steps, we adopt Denoising Diffusion Implicit Models (DDIM) [29] sampling, which replaces the stochastic reverse process with a deterministic update:

$$\mathbf{a}_t^{k-1} = \sqrt{\bar{\alpha}_{k-1}} \hat{\mathbf{a}}_t^0 + \sqrt{1 - \bar{\alpha}_{k-1}} \epsilon_\theta(\mathbf{a}_t^k, s_t, k), \quad (4)$$

where $\hat{\mathbf{a}}_t^0$ denotes the prediction of the clean action chunk.

B.2 EXPERTGEN

The set of hyper-parameters used in this paper is shown in Table 6

B.3 EXPERTGEN-PPO

EXPERTGEN-PPO uses PPO for expert policy Acquisition instead of FastTD3. We list the PPO related hyper-parameters in Table 7. We found it important to use a large initial log-standard deviation (e.g., 0.0) to matches with the training initial noise distribution to maintain a relative high initial task success.

B.4 Residual RL

Residual RL augments a fixed base prior policy π_P with a learnable residual policy π_θ that predicts corrective actions. Given a state s_t , the base policy outputs a nominal action $a_t^0 = \pi_0(s_t)$, while the residual policy predicts an additive correction conditioned on both the state and the base action $\delta a_t = \pi_\theta(s_t, a_t^0)$. The final action executed in the environment is given by $a_t = a_t^0 + \delta a_t$.

The critic evaluates the composed action under the environment dynamics. The state-action value function is defined as

$$Q_\phi(s_t, a_t^0, \delta a_t) \triangleq Q_\phi(s_t, a_t), \quad (5)$$

and is trained using standard temporal-difference learning with targets computed from the combined action at the next state.

The residual actor is optimized to maximize the critic while keeping the base policy fixed:

$$\max_{\theta} \mathbb{E}_{s_t \sim \mathcal{D}} [Q_\phi(s_t, \pi_P(s_t), \pi_\theta(s_t, \pi_P(s_t)))]. \quad (6)$$

We also adapt FastTD for learning the residual policy π_θ . The list of hyper-parameters used for residual RL is shown in Table 8.

Hyperparameters	ANYTASK	AutoMate
Generative Behavior Prior Modeling		
Architecture	U-Net [38]	U-Net
Number of groups	8	8
Kernel size	5	5
Number of channels	[128, 256]	[128, 256]
Diffusion step embedding dim.	16	16
Condition type	FiLM	FiLM
Action chunk length	16	8
Number of epochs	500	500
Batch size	256	256
Number of demonstrations	1000	500
Expert Policy Acquisition		
Receding horizon	8	8
Critic learning rate	3e-4	3e-4
Actor learning rate	3e-4	3e-4
Buffer size	8196 × 1024	8196 × 256
Batch size	8196	8196
Policy noise	0.001	0.001
std_min	0.005	0.005
std_max	0.4	0.4
Number of updates	8	8
Number of steps	4	4
Number of atoms	101	101
v_min	0.0	0.0
v_max	100.0	400.0
Discount factor	0.99	0.99

Table 6: EXPERTGEN hyperparameters. The right two columns indicate the hyperparameters for ANYTASK and AutoMate, respectively.

Hyperparameters	ANYTASK
Expert Policy Acquisition	
Receding horizon	8
Number of steps	64
Learning rate	5e-4
Learning rate schedule	fixed
Discount factor	0.995
e_clip	0.2
Entropy coefficient	-2e-4
Initial logstd	0.0

Table 7: EXPERTGEN-PPO hyperparameters.

B.5 Score-Matching Motion Priors (SMP) [34]

Given a pretrained diffusion policy ϵ_θ that models the distribution of expert action chunks, Score-Matching Motion Priors (SMP) repurpose this frozen diffusion model as a task-agnostic motion prior via score distillation sampling (SDS). During policy learning, an action chunk $\tilde{\mathbf{a}}_t^0$ generated by the current policy is evaluated under the diffusion prior by injecting Gaussian noise $\epsilon \sim \mathcal{N}(0, I)$ using the same forward diffusion process,

$$\tilde{\mathbf{a}}_t^k = \sqrt{\alpha_k} \tilde{\mathbf{a}}_t^0 + \sqrt{1 - \alpha_k} \epsilon. \quad (7)$$

Hyperparameters	ANYTASK
Residual RL	
Warm-start steps	10240
Action scale	0.02
Receding horizon	8
FastTD3	
Critic learning rate	3e-4
Actor learning rate	3e-4
Buffer size	8196×1024
Batch size	8196
Policy noise	0.001
std_min	0.005
std_max	0.4
Number of updates	8
Number of steps	4
Number of atoms	101
v_min	0.0
v_max	100.0
Discount factor	0.995

Table 8: Residual RL hyperparameters.

The denoising network then predicts the injected noise as $\hat{\epsilon} = \epsilon_{\theta}(\tilde{\mathbf{a}}_t^k, s_t, k)$. The squared error $\|\hat{\epsilon} - \epsilon\|_2^2$ provides a score-matching signal that measures how well the policy-generated action aligns with the expert demonstration distribution captured by the diffusion model.

To integrate this signal into reinforcement learning, SMP converts the SDS loss into a bounded prior reward,

$$r_{\text{smp}} = \exp(-w_s \|\hat{\epsilon} - \epsilon\|_2^2), \quad (8)$$

where w_s controls the strength of the motion prior. To reduce variance arising from a single diffusion timestep, the SDS loss is evaluated over a fixed set of diffusion steps and averaged before computing the reward. This prior reward is combined with a task-specific reward to train the policy using standard reinforcement learning algorithms, while keeping the diffusion model fixed throughout training. As a result, SMP provides a reusable and stationary behavior regularizer that encourages policy-generated actions to remain in-distribution with respect to expert demonstrations, without requiring adversarial training or access to the original dataset during policy optimization.

To maintain fair comparison, SMP also employs FastTD3 as the underlying RL algorithm. Table 9 lists all the hyper-parameters for our SMP implementation.

Hyperparameters	ANYTASK
Warm-start steps	10240
Action scale	0.02
Receding horizon	8

Table 9: SMP hyperparameters.

C Benchmarks

C.1 ANYTASK

ANYTASK [4] is an automated framework for automatic task design and large-scale synthetic data generation in robotic manipulation. For benchmarking, eight tabletop manipulation tasks are selected (an overview is shown in Fig. 8), covering a diverse set of behaviors including lifting, push-

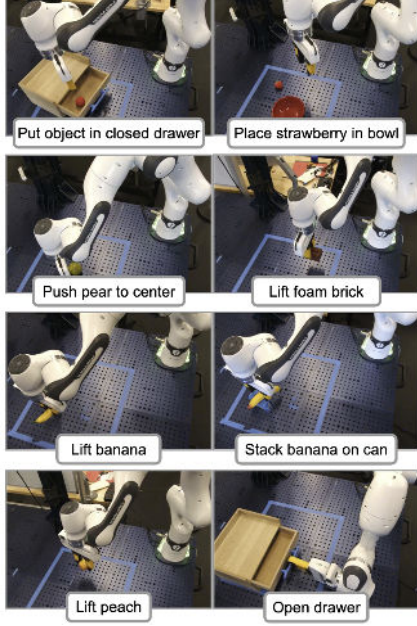


Figure 8: An overview of the selected eight tasks from ANYTASK benchmark.

ing, stacking, pick-and-place, and drawer opening. Together, these tasks span short-horizon and long-horizon interactions, articulated objects, and contact-rich manipulation scenarios.

Imperfect behavior priors are some scripted policies [5] synthesized by an LLM that invokes an existing skill library. These policies provide incomplete state coverage and lack explicit failure recovery behaviors, which constitutes the source of imperfectness. All scripted policies are shown in Fig. 17. We collect 1000 demonstrations for each task from the scripted policies.

A state-based diffusion policy is trained to generate action chunks with a horizon of 16, conditioned on a structured state vector. The definitions of the state and action representations are summarized in Table 10. For tasks involving fewer than two rigid objects or no articulated objects, the corresponding state dimensions are zero-masked. Task specification is encoded via a learned task embedding, obtained as the output of a lightweight MLP applied to a one-hot task index.

State	Dimension
End-effector pose	16
robot arm joint positions	9
Two object positions	6
Two object 6D rotations	12
Articulated object position	3
Articulated object 6D rotation	6
Articulated joint position	1
Task embedding	16
Action	Dimension
End-effector position	3
End-effector 6D rotation	6
Gripper action	1

Table 10: State condition and action definitions for state-based diffusion policy for the ANYTASK benchmark.

C.2 AutoMate

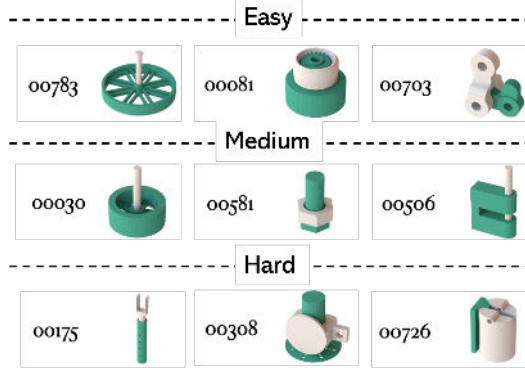


Figure 9: Visualization of the selected nine assets from AutoMate benchmark covering three difficulty levels (easy, medium, and hard).

AutoMate [9] is a simulation benchmark for robotic dexterous manipulation in industrial assembly, with a particular emphasis on high-precision peg-insertion tasks. The benchmark consists of a collection of insertion scenarios with tight geometric tolerances, demanding accurate pose alignment, fine-grained contact reasoning, and force-sensitive control during execution. For evaluation, nine assets are selected from AutoMate, evenly spanning three difficulty levels: easy (success rate $> 80\%$), medium (success rate between 40% and 80%), and hard (success rate $< 40\%$), where difficulty is defined based on the performance of single-task expert policies provided by AutoMate. An overview of the selected assets is shown in Fig. 9.

In this benchmark, imperfect behavior priors are generated from a scripted policy that performs peg disassembly starting from an already assembled configuration. Imperfect demonstrations are obtained by reversing these disassembly trajectories to approximate the assembly process. Because assembly and disassembly dynamics are inherently asymmetric, this inversion induces a dynamics mismatch, resulting in demonstrations with imperfect contact interactions. We collect 500 demonstrations for each task from the scripted policy. The scripted policy for disassembly is shown in Fig. 19 including three skill functions, `_disassemble_plug_from_socket`, `_lift_gripper`, and `_randomize_gripper_pose`.

A state-based diffusion policy is trained to generate action chunks with a horizon of 8, conditioned on a structured state vector. The definitions of the state and action representations are summarized in Table 11. The 32-dimensional task embedding is computed from a pretrained point-cloud autoencoder similar to AutoMate [9].

C.3 Real-World Evaluation

Our sim-to-real policies are deployed on a single-arm Franka robot equipped with a Robotiq 2F-85 gripper. Perception is provided by three calibrated Intel RealSense D435i cameras, which yield a merged point-cloud and RGB observations. The RGB observations additionally includes a RealSense D405 wrist mounted camera. All policy inference is performed on a local workstation with an AMD Ryzen Threadripper PRO 5955WX CPU and an NVIDIA RTX 6000 Ada GPU.

State	Dimension
End-effector pose	16
Held positions	3
Held 6D rotation	6
Task embedding	32
Action	Dimension
End-effector position	3
End-effector 6D rotation	6

Table 11: State condition and action definitions for state-based diffusion policy for the AutoMate benchmark.

Scripted policy - *Lift Banana*

```
def scripted_policy(env):
    import torch

    # Object id for banana
    banana_id = 1

    # Step 1: Pick up the banana. This skill will grasp the banana
    and lift it with an internally defined distance.
    pick(env, banana_id)

    # Step 2: Compute the target position for lifting.
    # We want to ensure the banana is lifted upward by at least 20 cm
    relative to its initial position.
    # Get the initial position of the banana (batched tensor of shape
    (N, 3)).
    initial_banana_pos = get_object_initial_position(env, banana_id)

    # Define a lift offset of 0.25 m (25 cm) to ensure it meets the
    20 cm requirement.
    lift_offset = torch.tensor([0.0, 0.0, 0.25],
    device=initial_banana_pos.device).reshape(1, 3)

    # Broadcast the lift offset and add to the initial banana
    position
    target_pos = initial_banana_pos + lift_offset

    # Step 3: Move the robot's end effector to the target position
    while keeping the gripper closed.
    move_to(env, target_pos, target_orientation=None,
    gripper_open=False)

    # Step 4: Ensure the gripper remains closed to keep a firm
    grasp.
    close_gripper(env)
```

Figure 10: Scripted policy for *Lift Banana*

Scripted policy - *Lift Brick*

```
def scripted_policy(env):
    import torch

    # Object id for soft_brick
    soft_brick_id = 1

    # Step 1: Pick up the soft_brick. This skill will grasp the
    # soft_brick and lift it with an internally defined distance.
    pick(env, soft_brick_id)

    # Step 2: Compute the target position for lifting.
    # We want to ensure the soft_brick is lifted upward by at least
    # 20 cm relative to its initial position.
    # Get the initial position of the soft_brick (batched tensor of
    # shape (N, 3)).
    initial_soft_brick_pos = get_object_initial_position(env,
    soft_brick_id)

    # Define a lift offset of 0.25 m (25 cm) to ensure it meets the
    # 20 cm requirement.
    lift_offset = torch.tensor([0.0, 0.0, 0.25],
    device=initial_soft_brick_pos.device).reshape(1, 3)

    # Broadcast the lift offset and add to the initial soft_brick
    # position
    target_pos = initial_soft_brick_pos + lift_offset

    # Step 3: Move the robot's end effector to the target
    # position while keeping the gripper closed.
    move_to(env, target_pos, target_orientation=None,
    gripper_open=False)

    # Step 4: Ensure the gripper remains closed to keep a firm
    # grasp.
    close_gripper(env)
```

Figure 11: Scripted policy for *Lift Brick*

Scripted policy - *Lift Peach*

```
def scripted_policy(env):
    import torch

    # Object id for peach
    peach_id = 1

    # Step 1: Pick up the peach. This skill will grasp the peach
    and lift it with an internally defined distance.
    pick(env, peach_id)

    # Step 2: Compute the target position for lifting.
    # We want to ensure the peach is lifted upward by at least 20 cm
    relative to its initial position.
    # Get the initial position of the peach (batched tensor of shape
    (N, 3)).
    initial_peach_pos = get_object_initial_position(env, peach_id)

    # Define a lift offset of 0.25 m (25 cm) to ensure it meets the
    20 cm requirement.
    lift_offset = torch.tensor([0.0, 0.0, 0.25],
    device=initial_peach_pos.device).reshape(1, 3)

    # Broadcast the lift offset and add to the initial peach position
    target_pos = initial_peach_pos + lift_offset

    # Step 3: Move the robot's end effector to the target
    position while keeping the gripper closed.
    move_to(env, target_pos, target_orientation=None,
    gripper_open=False)

    # Step 4: Ensure the gripper remains closed to keep a firm
    grasp.
    close_gripper(env)
```

Figure 12: Scripted policy for *Lift Peach*

Scripted policy - *Open Drawer*

```
def scripted_policy(env):
    obj_id = 1
    open_drawer(env, obj_id)
```

Figure 13: Scripted policy for *Open Drawer*

Scripted policy - *Push Pear to Center*

```
def scripted_policy(env):
    import torch
    # Push the pear (object_id=1) to target xy (0.4, -0.5) using
    iterative, state-based pushes
    pear_id = 1
    target_xy = make_atensor(env, [0.4, -0.44])
    push_or_pull_object_to_xy(env, pear_id, target_xy)
```

Figure 14: Scripted policy for *Push Pear to Center*

Scripted policy - *Stack Banana on Can*

```
def scripted_policy(env):
    import torch
    banana_id = 2
    can_id = 1

    # 1) Pick the banana
    pick(env, banana_id)

    # 2) Lift to 25 cm above initial height
    init_banana = get_object_initial_position(env, banana_id) # (N,3)
    lift_offset = torch.tensor([0.0, 0.0, 0.18],
device=init_banana.device).reshape(1, 3)
    lifted_pos = init_banana + lift_offset
    move_to(env, lifted_pos, target_orientation=None,
gripper_open=False)
    can_position = get_object_position(env, can_id)
    target_position = can_position.clone()
    place(env, target_position)
```

Figure 15: Scripted policy for *Stack Banana on Can*

Scripted policy - *Put Object in Closed Drawer*

```
def scripted_policy(env):
    import torch
    obj_id = 1 # drawer
    strawberry_id = 2

    # 1) Open the upper drawer
    open_drawer(env, obj_id, reset_after_open=True)

    # 2) Pick the strawberry
    pick(env, strawberry_id)

    # 3) Lift strawberry 23 cm above its initial position
    init_straw = get_object_initial_position(env, strawberry_id) #
(N,3)
    lift_offset = torch.tensor([0.0, 0.0, 0.23],
device=init_straw.device).reshape(1, 3)
    lifted_pos = init_straw + lift_offset
    move_to(env, lifted_pos, target_orientation=None,
gripper_open=False)

    # 4) Place location: use upper drawer's (x, y) and fixed z =
0.23
    drawer_pos = get_articulated_obj_part_pos(env, obj_id,
'upper_drawer', use_bbox_center=True) # (N,3)
    z_const = torch.full((drawer_pos.shape[0], 1), 0.23,
device=drawer_pos.device)
    place_target = torch.cat([drawer_pos[:, :2], z_const], dim=1) #
(N,3)

    # Move to place location and open gripper directly (no pre-place)
    move_to(env, place_target, target_orientation=None,
gripper_open=False)
    open_gripper(env)
```

Figure 16: Scripted policy for *Put Object in Closed Drawer*

Scripted policy - *Place Strawberry in Bowl*

```
def scripted_policy(env):
    import torch
    strawberry_id = 2
    bowl_id = 1

    # 1) Pick the strawberry
    pick(env, strawberry_id)

    # 2) Lift to 25 cm above initial height
    init_strawberry = get_object_initial_position(env, strawberry_id) #
    (N,3)
    lift_offset = torch.tensor([0.0, 0.0, 0.18],
    device=init_strawberry.device).reshape(1, 3)
    lifted_pos = init_strawberry + lift_offset
    move_to(env, lifted_pos, target_orientation=None,
    gripper_open=False)
    bowl_position = get_object_position(env, bowl_id)
    target_position = bowl_position.clone()
    place(env, target_position)
```

Figure 17: Scripted policy for *Put Object in Closed Drawer*

Scripted policy - _disassemble_plug_from_socket

```
def _disassemble_plug_from_socket(self):
    """Lift plug from socket till disassembly and then randomize
    end-effector pose."""

    if_intersect = np.ones(self.num_envs, dtype=np.float32)

    env_ids = np.argwhere(if_intersect == 1).reshape(-1)
    lift_distance = self.disassembly_dists * 3.0
    self._lift_gripper(lift_distance, self.cfg.task.disassemble_sim_steps,
env_ids)

    self.step_sim_no_action()

    if_intersect = (self.held_pos[:, 2] < self.fixed_pos[:, 2] +
self.disassembly_dists).cpu().numpy()
    env_ids = np.argwhere(if_intersect == 0).reshape(-1)
    self._randomize_gripper_pose(self.cfg.task.move_gripper_sim_steps,
env_ids)
```

Scripted policy - _lift_gripper

```
def _lift_gripper(self, lift_distance, sim_steps, env_ids=None):
    """Lift gripper by specified distance. Called outside RL loop
    (i.e., after last step of
    episode)."""

    ctrl_tgt_pos = torch.empty_like(self.fingertip_midpoint_pos).
copy_(self.fingertip_midpoint_pos)
    ctrl_tgt_quat = torch.empty_like(self.fingertip_midpoint_quat).
copy_(self.fingertip_midpoint_quat)
    ctrl_tgt_pos[:, 2] += lift_distance
    lifted = self.fixed_pos[:, 2] > self.disassembly_dists * 0.95
    ctrl_tgt_pos[lifted, :2] += torch.randn((self.num_envs, 2),
dtype=torch.float32, device=self.device) * 0.02
    if len(env_ids) == 0:
        env_ids = np.array(range(self.num_envs)).reshape(-1)

    self._move_gripper_to_eef_pose(env_ids, ctrl_tgt_pos, ctrl_tgt_quat,
sim_steps, if_log=True)
```

Figure 18: Scripted policy for AutoMate disassembly tasks.

Scripted policy - _randomize_gripper_pose

```
def _randomize_gripper_pose(self, sim_steps, env_ids):
    """Move gripper to random pose."""

    ctrltgt_pos = torch.empty_like(self.gripper_goal_pos).copy_(
self.gripper_goal_pos )
    ctrltgt_pos[:, 2] += self.cfg_task.gripper_randz_offset

    # ctrltgt_pos = torch.empty_like(self.fingertip_midpoint_pos).copy
    _ ( self.fingertip_midpoint_pos )

    fingertip_centered_pos_noise = 2 * (
        torch.rand((self.num_envs, 3), dtype=torch.float32,
device=self.device) - 0.5
    ) # [-1, 1]
    pos_noise = torch.tensor([
        0.03, 0.03, 0.02 + self.disassembly_dists[0]
    ], device=self.device)
    fingertip_centered_pos_noise = fingertip_centered_pos_noise @
torch.diag(pos_noise)
    fingertip_centered_pos_noise[:, 2] -= (0.02 +
self.disassembly_dists[0]) / 2.0
    ctrltgt_pos += fingertip_centered_pos_noise

    # Set target rot
    ctrltgt_target_fingertip_centered_euler = (
        torch.tensor(self.cfg_task.fingertip_centered_rot_initial,
device=self.device)
        .unsqueeze(0)
        .repeat(self.num_envs, 1)
    )

    fingertip_centered_rot_noise = 2 * (
        torch.rand((self.num_envs, 3), dtype=torch.float32,
device=self.device) - 0.5
    ) # [-1, 1]
    fingertip_centered_rot_noise = fingertip_centered_rot_noise @
torch.diag(
        torch.tensor(self.cfg_task.gripper_rand_rot_noise,
device=self.device)
    )
    ctrltgt_target_fingertip_centered_euler += fingertip_centered_rot_noise
    ctrltgt_quat = torch_utils.quat_from_euler_xyz(
        ctrltgt_target_fingertip_centered_euler[:, 0],
        ctrltgt_target_fingertip_centered_euler[:, 1],
        ctrltgt_target_fingertip_centered_euler[:, 2],
    )

    self.move_gripper_to_eef_pose(env_ids, ctrltgt_pos, ctrltgt_quat,
sim_steps, if_log=True)
```

Figure 19: Scripted policy for AutoMate disassembly tasks.