ONEDEMO: UNIFIED POLICY LEARNING AND DATA GENERATION FROM A SINGLE DEMONSTRATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning robot policies from demonstrations is often limited by the cost of collecting diverse, high-quality expert data. While an appealing alternative is to expand a single successful demonstration into a diverse set of successful trajectories, existing augmentation methods typically rely on manually designed transformation rules or complex trajectory planning, requiring substantial task-specific engineering and human effort. We introduce OneDemo, a reinforcement learning framework that *learns to edit a single successful demonstration* for new task configurations. We formulate demonstration editing as a structured reinforcement learning problem, in which an editor learns, from sparse task-success rewards, both when to transition between object-centric trajectory segments and how to geometrically adapt each segment. The learned editor serves both as a task policy and as a reusable demonstration generator. Across 15 Meta-World tasks, OneDemo reaches a 92.7% success rate within 53.7k environment interactions on average. Across 9 DexMimicGen tasks, it generates successful demonstrations with a 98.4% average success rate, and BC-RNN policies trained on the generated data achieve 99.6% average success at matched dataset scale. These results show that learning to edit a single successful demonstration provides a structured and effective basis for both policy learning and scalable demonstration generation.

1 INTRODUCTION

Learning manipulation policies from demonstrations has emerged as a powerful paradigm for complex robotic tasks, including long-horizon and bimanual manipulation. Modern imitation learning methods (Zhao et al., 2023; Chi et al., 2025; Ze et al., 2024; Liu et al., 2025) can reproduce sophisticated behaviors when provided with sufficiently large and diverse demonstration datasets. Collecting such datasets through human teleoperation, however, remains expensive and time-consuming, while the resulting demonstrations typically cover only a small fraction of the task configurations a robot may encounter.

A natural alternative is to generate additional demonstrations by adapting a small number of successful trajectories to new object poses, scene configurations, or task initializations. If these adaptations can be performed reliably, a single demonstration can provide supervision over a substantially broader task distribution. Existing approaches toward this goal face an important trade-off. Simple perturbation-based augmentation is easy to apply (Wang et al., 2024; Zhou et al., 2023), but can produce invalid or unsuccessful trajectories when the target task configuration deviates substantially from the source demonstration. More structured approaches, such as MimicGen (Mandlekar et al., 2023), obtain substantially higher reliability by decomposing demonstrations into object-centric segments and geometrically transforming them to new scenes. Their behavior, however, is largely determined by manually specified task structure and trajectory-generation procedures.

In this work, we ask whether the edits required to adapt a successful demonstration to a new task configuration can instead be *learned*. Recent work has begun to explore related directions: DemoGrasp (Yuan et al., 2026) learns a single global edit to a demonstrated dexterous grasp using reinforcement learning, while DexFlyWheel (Zhu et al., 2025) combines demonstration-based augmentation with residual reinforcement learning in an iterative data-generation pipeline. Building on this emerging direction, we investigate a more structured form of demonstration editing that extends to long-horizon, bimanual, and dexterous manipulation.

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.

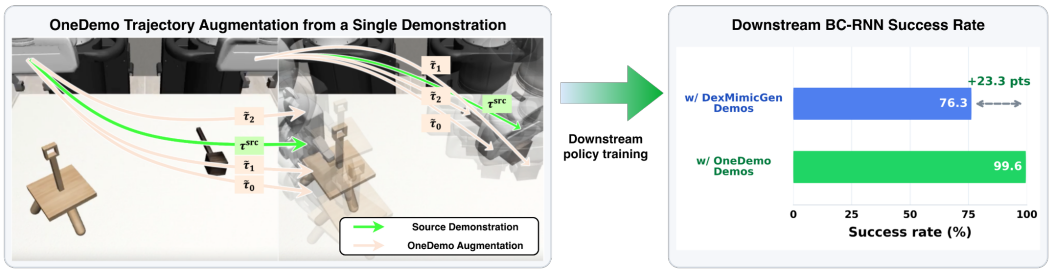

Figure 1: **OneDemo learns to edit a single demonstration to generate varying successful trajectories under new spatial configurations. Left:** OneDemo adapts the source trajectory to generate successful augmentations. **Right:** Recurrent behavior cloning (BC-RNN) trained on demonstrations augmented by OneDemo improves success from 76.3% to 99.6% compared with DexMimicGen data.

Starting from a single source trajectory, we formulate demonstration editing as a structured reinforcement learning problem. Rather than exploring directly over low-level robot actions, the policy operates over edits to an existing successful motion: it learns when to transition between object-centric trajectory segments, and how to apply compact geometric residual corrections to their motion. Specifically, we represent the source demonstration as an ordered sequence of object-centric stages, each associated with an anchor object or frame. At the beginning of each stage, the policy predicts both a duration, which determines the number of timesteps assigned to that stage, and a rigid-body residual transformation that adapts the corresponding motion to the current scene. We optimize these decisions using PPO (Schulman et al., 2017) with only sparse task-success rewards, with the anchor sequence providing only a coarse initialization.

A key consequence of this formulation is that the learned trajectory editor serves both as a task policy and as a demonstration generator. As a task policy, it adapts and executes the source demonstration under new task configurations, while restricting exploration to structured modifications of an already successful trajectory rather than low-level robot actions. This structured exploration can substantially simplify the reinforcement learning problem. The same editor can then be rolled out across task configurations to generate successful trajectories for downstream learning.

Our experiments evaluate both roles of the proposed system. When deployed directly as a policy, OneDemo learns high-performing task behavior from a single source demonstration within a small interaction budget. When used as a demonstration generator, it produces high-quality trajectories that substantially improve downstream imitation learning at matched dataset scale.

Our main contributions are:

1. We introduce ONEDEMO, a structured reinforcement learning framework that uses a single successful demonstration to define a compact search space for adapting behavior to new spatial task configurations.
2. We formulate demonstration adaptation as learning two complementary edits from sparse task-success rewards: temporal segmentation of the source trajectory and object-centric residual transformations that geometrically adapt each trajectory segment to the current task configuration.
3. We show that the learned trajectory editor serves both as a sample-efficient task policy and as a high-quality demonstration generator. When used directly as a policy, it achieves strong performance with only a small budget of online environment interactions. When used for data generation, it produces successful demonstrations across long-horizon, bimanual, and dexterous manipulation tasks that substantially improve downstream policy learning.

2 RELATED WORK

Imitation Learning. Imitation learning aims to learn policies from demonstrations of desired behavior (Osa et al., 2018). Common approaches include behavior cloning with recurrent poli-

cies (Mandlekar et al., 2021), action-chunking methods such as ACT (Zhao et al., 2023), and diffusion-based policies (Chi et al., 2025; Ze et al., 2024). While these methods differ in architecture and prediction strategy, their performance depends strongly on the coverage and diversity of the demonstration dataset (Mandlekar et al., 2021; Jiang et al., 2025). Interactive methods such as DAgger (Ross et al., 2011) and human-in-the-loop correction (Mandlekar et al., 2020) improve coverage by collecting expert supervision on states encountered by the learned policy, but typically assume continued access to an expert. Our system can be seen as a learned expert policy from a single source demonstration, which can both generate diverse offline training trajectories and potentially provide supervision within interactive imitation-learning schemes such as DAgger.

Reinforcement Learning with Demonstrations. Reinforcement learning can learn task policies without expert demonstrations, but exploration becomes difficult for long-horizon manipulation tasks with sparse rewards. To address this problem, prior work improves sample efficiency by using demonstrations to initialize policies (Rajeswaran et al., 2017), populate replay buffers (Vecerik et al., 2017), provide auxiliary imitation losses (Rajeswaran et al., 2017; Nair et al., 2018), or to initialize episodes from favorable states (Nair et al., 2018). Residual RL (Johannink et al., 2019), similarly, learns corrections on top of an existing policy to restrict policy search space. Our method also uses demonstrated behavior as a strong prior, but directly learns how the demonstrated trajectory itself can be modified to generalize to new task configurations.

Synthetic Demonstration Generation. Another line of work reduces teleoperation cost by generating additional trajectories from a small set of source demonstrations. Simple perturbation-based methods (Wang et al., 2024; Zhou et al., 2023) introduce variations around existing trajectories but may produce mostly unsuccessful rollouts. More structured methods such as MimicGen (Mandlekar et al., 2023) decompose demonstrations into object-centric segments and transform them geometrically to match new scene configurations, enabling much more reliable synthesis. Such approaches (Jiang et al., 2025; Garrett et al., 2025), however, rely on task-specific structure and transformation rules. Our work instead learns to do trajectory adaptation through reinforcement learning, allowing successful transformations to emerge from interactions rather than being explicitly prescribed based on prior knowledge.

3 METHOD

Given a single successful source demonstration, our goal is to learn to adapt it to new spatial configurations of the same task using only sparse task-success rewards. In this work, we define a new task configuration as one in which the object geometry and task requirements remain fixed, while the poses of task-relevant objects vary. To make this adaptation practical for long-horizon tasks and tasks requiring high spatial precision, we avoid reinforcement learning directly over low-level robot actions. Such exploration can require hundreds of coordinated decisions before receiving a sparse success signal. Instead, we restrict exploration to structured edits of an existing successful trajectory.

Our method represents the source demonstration as an ordered sequence of object-centric trajectory segments. For each segment, the policy decides *when the segment should end* and *how its motion should be geometrically adapted* to the current task configuration. The resulting trajectory is then executed by the low-level controller of the robot. We first formulate this representation as a structured specialization of the options framework (Sections 3.1 and 3.2), and then describe its realization and initialization (Section 3.3). Figure 2 provides an overview of the full pipeline.

3.1 OPTION-BASED FORMULATION

We first present the formulation for a single manipulator, where the standard call-and-return interpretation of options applies directly. We describe the bimanual extension used in our experiments later in Section 3.2.

Our formulation builds on the options framework (Sutton et al., 1999), which augments a Markov decision process with temporally extended actions. We provide a brief overview in Appendix A.1. A Markovian option ω is conventionally defined by

$$\omega = (I_\omega, \pi_\omega, \beta_\omega),$$

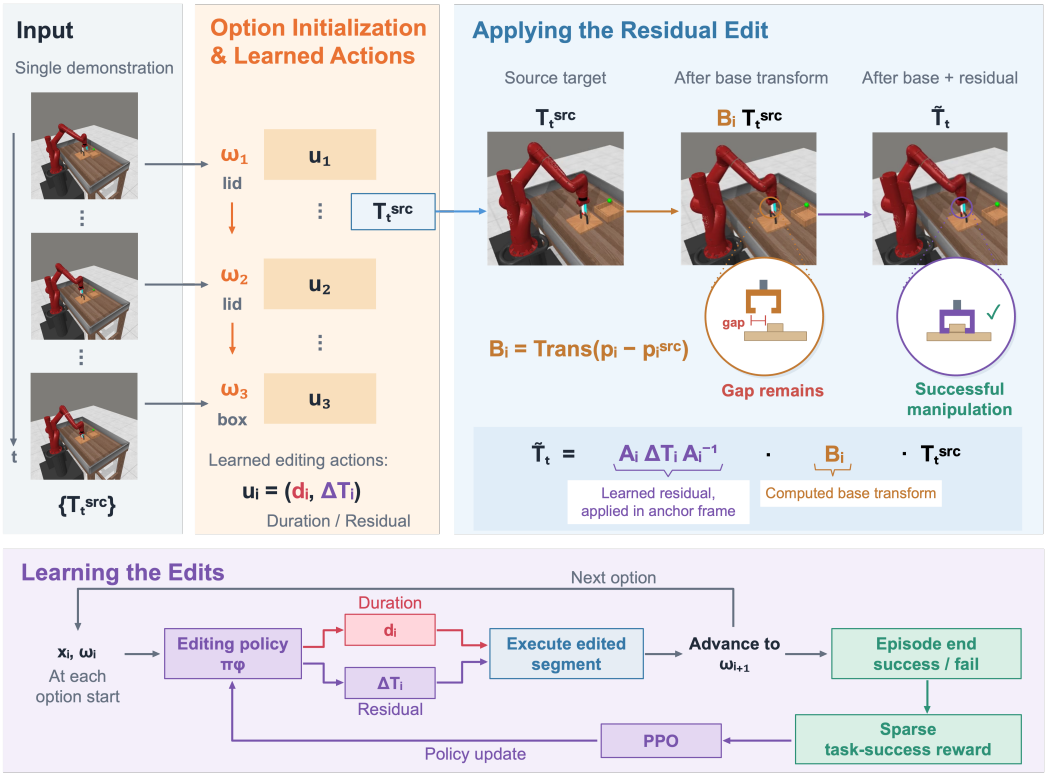

Figure 2: **Overview of OneDemo**. Semantic options define anchor objects and base transform, while the learned policy predicts an option duration and residual transformation at each option boundary. The residual is applied on top of the base transform and optimized from sparse task-success rewards.

where I_ω is its initiation set, π_ω is the intra-option policy, and β_ω determines when the option terminates. Option-Critic (Bacon et al., 2017) showed that the intra-option policy and termination mechanism can be optimized jointly from task return, without requiring option-specific rewards.

We specialize this abstraction to demonstration adaptation. We assume an ordered sequence of semantic options

$$\omega_1 \to \omega_2 \to \cdots \to \omega_K.$$

When ω_i terminates, execution advances deterministically to ω_{i+1}, so no policy over options needs to be learned. The remaining option components correspond to two trajectory-editing decisions:

$$\beta_\omega \;\mapsto\; d_i \quad \text{(transition timing)}, \qquad \pi_\omega \;\mapsto\; \Delta T_i \quad \text{(trajectory adaptation)}.$$

At the start of option ω_i, the intra-option policy predicts both a duration d_i and a rigid-body transformation ΔT_i applied to the corresponding source-demonstration segment. Thus, each option-level action is

$$u_i = (d_i, \Delta T_i).$$

Since u_i persists for a variable number of primitive control steps, the resulting option-level process is naturally semi-Markov (Sutton et al., 1999). This preserves the temporal abstraction while restricting exploration to structured trajectory edits rather than low-level robot actions.

3.2 Object-Centric Trajectory Editing

Given a task, let a successful source demonstration τ^{src} contain a sequence of end-effector target poses:

$$\tau^{\text{src}} = \{T_t^{\text{src}}\}_{t=0}^{H-1}, \qquad T_t^{\text{src}} \in SE(3),$$

where T_t^{src} denotes the end-effector target pose at time step t. As introduced above, we represent the task as an ordered sequence of semantic options Ω:

$$\Omega = (\omega_1, \ldots, \omega_K),$$

where each option ω_i is associated with an anchor object or task frame corresponding to a particular stage of the manipulation.

Our key geometric assumption is that, although relevant object poses vary across task configurations, successful motion relative to those objects is approximately reusable. For option ω_i, let A_i^{src} denote the pose of its associated anchor object in the source demonstration, and let p_i^{src} be the translation component of A_i^{src}. Correspondingly, let A_i denote the live pose of the same anchor during execution, with translation component p_i.

For a given task configuration, we initialize trajectory adaption for option ω_i using translation-only anchor alignment,

$$B_i = \text{Trans}(p_i - p_i^{\text{src}}).$$

The editing policy then predicts an additional residual transformation

$$\Delta T_i \in SE(3),$$

expressed in the live anchor frame and held fixed throughout the option.

Let $\mathcal{T}_i$ denote the source-trajectory indices assigned to ω_i. For $t \in \mathcal{T}_i$, the edited target is

$$\widetilde{T}_t = A_i \Delta T_i A_i^{-1} B_i T_t^{\text{src}}.$$

Only the end-effector pose trajectory is geometrically edited. Gripper and dexterous-hand commands are replayed unchanged from the source demonstration. Here, B_i provides a coarse adaptation to the new anchor location, while ΔT_i learns the remaining geometric correction required for successful execution.

This construction is related to the object-centric trajectory transformations used by MimicGen (Mandlekar et al., 2023) and DexMimicGen (Jiang et al., 2025). The key distinction is that OneDemo treats the temporal segmentation and residual geometric adaptation as learned decisions rather than prescribing them in advance. Additional motivation for our translation-only initialization is provided in Appendix A.2.

Bimanual Extension. We extend the single-arm formulation to the bimanual setting by maintaining a separate ordered semantic phase sequence and anchor list for each arm. Each arm has its own active phase, duration, and held residual warp, while both arms execute on the same synchronized control clock. Phase boundaries may occur asynchronously across arms.

3.3 Learning Temporal Boundaries and Geometric Adaptations

Suppose option ω_i begins at source-trajectory time index κ_i. At option initiation, the policy π, parameterized by ϕ, samples the option-level action

$$u_i = (d_i, \Delta T_i) \sim \pi_\phi(\cdot \mid x_i, \omega_i),$$

where x_i denotes the policy observation at the beginning of the option. The duration d_i determines the source-trajectory segment assigned to option ω_i:

$$\mathcal{T}_i = \{\kappa_i, \ldots, \kappa_i + d_i - 1\}, \qquad \kappa_{i+1} = \kappa_i + d_i.$$

The residual transformation ΔT_i is held fixed while the transformed targets indexed by $\mathcal{T}_i$ are executed. Once this segment is completed, execution advances to the next option ω_{i+1}.

Conditioned on the active option, we factorize the editing policy as

$$\pi_\phi(d, \Delta T \mid x, \omega) = \pi_{\vartheta_\omega}(d \mid x)\, \pi_{\theta_\omega}(\Delta T \mid x),$$

where ϑ_ω parameterizes the duration distribution and θ_ω parameterizes the residual transformation distribution for option ω. The complete policy parameters are

$$\phi = \{\vartheta_\omega, \theta_\omega\}_{\omega \in \Omega}.$$

We optimize the expected task return

$$J(\phi) = \mathbb{E}_{\pi_\phi}[R]$$

using PPO, where R is the sparse task-success return. Both temporal segmentation and geometric adaptation are therefore learned from end-task performance only.

The resulting search space contains only a small number of structured option-level decisions per episode,

$$\{u_i\}_{i=1}^{K},$$

rather than a long sequence of unconstrained low-level actions. This reduction is particularly useful for manipulation tasks in which successful trajectories require coordinated motion over long horizons but remain locally reusable relative to task-relevant objects.

Automatic Option Initialization. OneDemo requires an ordered sequence of semantic options and associated task-relevant anchors for initialization. We obtain this sequence automatically using an LLM given task metadata and downsampled telemetry from the source demonstration. The LLM predicts only a coarse semantic decomposition and the anchor associated with each option; precise temporal boundaries and geometric corrections are learned by OneDemo from task return. The initialization need not be minimal, since redundant options can be effectively shortened by the learned duration policy. Full prompting details are provided in Appendix A.3.

4 Experiments & Results

We evaluate OneDemo along two complementary roles, and perform additional ablation studies that analyze its key components:

1. **OneDemo as a policy:** We study whether the structured trajectory-editing action space enables sample-efficient reinforcement learning when OneDemo itself is deployed as the task policy.
2. **OneDemo for trajectory generation:** We evaluate whether the trajectories generated by OneDemo provide high-quality supervision for downstream policy learning.
3. **Ablation studies:** We analyze the respective contributions of learned temporal segmentation and geometric residual adaptation to option-level policy learning.

4.1 Experimental Setup

We evaluate OneDemo primarily on two manipulation benchmarks spanning single-arm, bimanual, and dexterous manipulation, with additional complementary results on RoboTwin. Each benchmark is used to evaluate a different aspect of the method. Full implementation details, network architectures, and training hyperparameters are provided in Appendix A.5.

Meta-World. Meta-World (Yu et al., 2020) contains single-arm manipulation tasks with a parallel-jaw gripper under randomized task configurations. We evaluate on 15 challenging tasks matching the subset used by MoDem (Hansen et al., 2022), a demonstration-guided model-based RL method designed for high sample efficiency under sparse rewards. We use this benchmark to evaluate OneDemo directly as a task policy, considering both the success rate of the resulting policy and the number of online environment interactions required for training.

DexMimicGen. DexMimicGen (Jiang et al., 2025) contains 9 complex bimanual manipulation tasks spanning three robot embodiments, including gripper-based, dexterous-hand, and humanoid manipulation. We use this benchmark to evaluate OneDemo as a demonstration generator. For each task, we generate a new dataset from a single source demonstration and evaluate its quality through the performance of downstream BC-RNN policies trained on the generated trajectories.

Additional Benchmark. We provide complementary results on RoboTwin (Chen et al., 2025) in the appendix. On selected bimanual tasks, we evaluate OneDemo-generated data in a low-data regime using Diffusion Policy trained from only 50 generated trajectories per task.

Table 1: **Meta-World success rate on the 15 tasks used by MoDem. Naive** uses the randomly initialized duration policy with greedy sampling and applies only the baseline transformation, with no residual warping. **Seg.-only** uses the learned duration policy with no residual warping. **OneDemo** uses both the learned duration policy and learned residual warping. Values are mean ± 95% confidence-interval half-width over five seeds; each evaluation uses 50 episodes.

Task	Naive	Seg.-only	OneDemo
Assembly	46.4 ± 42.9	$\mathbf{100.0 \pm 0.0}$	$\mathbf{100.0 \pm 0.0}$
Basketball	68.8 ± 13.2	76.8 ± 5.2	$\mathbf{84.4 \pm 5.5}$
Box Close	57.6 ± 8.5	66.8 ± 13.5	$\mathbf{83.6 \pm 14.2}$
Coffee Push	64.8 ± 41.7	99.2 ± 1.0	$\mathbf{100.0 \pm 0.0}$
Hand Insert	77.2 ± 22.7	99.2 ± 1.6	$\mathbf{100.0 \pm 0.0}$
Peg Insert Side	43.6 ± 16.4	$\mathbf{98.4 \pm 3.1}$	$\mathbf{98.4 \pm 3.1}$
Pick Place	89.6 ± 20.4	$\mathbf{100.0 \pm 0.0}$	$\mathbf{100.0 \pm 0.0}$
Pick Place Wall	93.2 ± 13.3	$\mathbf{100.0 \pm 0.0}$	$\mathbf{100.0 \pm 0.0}$
Push	82.0 ± 34.3	$\mathbf{100.0 \pm 0.0}$	$\mathbf{100.0 \pm 0.0}$
Push Wall	47.6 ± 31.4	92.8 ± 4.7	$\mathbf{97.6 \pm 2.9}$
Soccer	60.0 ± 19.8	73.6 ± 5.0	$\mathbf{75.2 \pm 4.6}$
Stick Pull	22.8 ± 15.3	45.2 ± 7.1	$\mathbf{56.8 \pm 12.7}$
Stick Push	$\mathbf{100.0 \pm 0.0}$	$\mathbf{100.0 \pm 0.0}$	$\mathbf{100.0 \pm 0.0}$
Sweep	68.4 ± 26.9	$\mathbf{100.0 \pm 0.0}$	$\mathbf{100.0 \pm 0.0}$
Sweep Into	87.6 ± 8.5	94.0 ± 4.5	$\mathbf{95.2 \pm 4.4}$
Average	67.3	89.7	**92.7**

4.2 OneDemo as a Sample-Efficient Policy

We first evaluate OneDemo directly as a task policy on the 15 Meta-World tasks used by MoDem (Hansen et al., 2022). For each task, we initialize from a single successful scripted demonstration and train using only sparse task-success rewards. We compare against our local reproduction of MoDem, a demonstration-guided model-based RL method, to contextualize the interaction budget required for sparse-reward policy learning. Additional details on the baseline and comparison protocol are provided in Appendix A.5.

Sample Efficiency. Table 1 summarizes the results. Under a budget of 200k environment interactions, OneDemo achieves an average best success rate of 92.7% and reaches its best checkpoint after only 53.7k interactions on average.

Figure 3 shows the corresponding learning curves. OneDemo improves rapidly from a strong initialization, exceeding 80% average success after roughly 15k interactions and approaching 90% within 100–150k interactions. Our MoDem reproduction improves more gradually over the same budget.

While we observe consistent improvement across all tasks with OneDemo, the magnitude of the gain varies by task. This variation reflects the inductive bias imposed by our trajectory-editing formulation. OneDemo makes only a small number of high-level decisions at segment boundaries, while the action sequence within each segment does not change in response to any new states or observations. It is therefore most effective when task variation can be captured by a small number of changes to segment timing and object-relative geometry. Gains are more modest on tasks such as Soccer and Stick Pull, where success benefits from continual state-dependent correction.

OneDemo is also robust to source-demonstration choice; results using randomly selected successful demonstrations are provided in Appendix A.4.

Ablations. The ablations in Table 1 show that temporal and geometric adaptation contribute differently across tasks. Learning segment timing alone increases average success from 67.3% to 89.7%, accounting for most of the improvement of the full model (92.7%). This indicates that, when the demonstrated geometry remains reusable, adapting *when* the trajectory transitions between semantic stages is often sufficient.

Residual geometric adaptation provides additional gains when translation-only anchor alignment is insufficient, most notably on Box Close ($66.8\% \rightarrow 83.6\%$) and Stick Pull ($45.2\% \rightarrow 56.8\%$). On

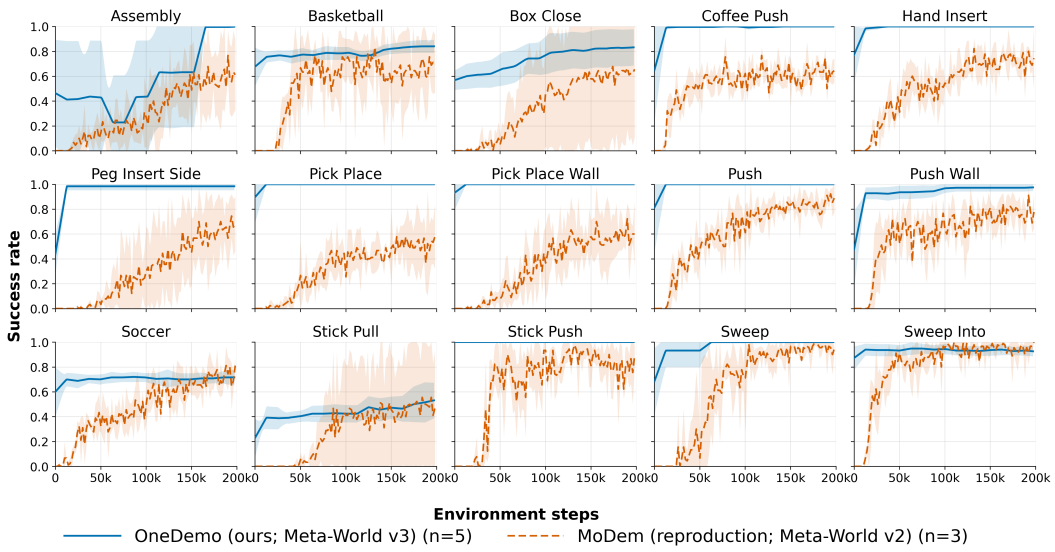

Figure 3: **Meta-World sample efficiency aggregated over 15 tasks.** OneDemo directly executes the edited demonstration as the task policy and achieves high success with substantially fewer environment interactions than our MoDem reproduction. Shaded regions denote 95% normal-approximation confidence intervals.

more precision-sensitive tasks, geometric adaptation can be essential: on DexMimicGen Threading, residual learning improves adaptation success substantially over segmentation alone; we provide this ablation in Appendix A.6. While the relative importance of temporal and geometric adaptation varies across tasks, OneDemo does not need to assume which will dominate; both are optimized within the same structured editing space.

4.3 OneDemo as a Demonstration Generator

We evaluate OneDemo as a demonstration generator on DexMimicGen using downstream imitation learning. The benchmark features long-horizon bimanual tasks across multiple robot embodiments, including both parallel-jaw grippers and dexterous hands.

For each task, we select one reliably replayable trajectory from the released dataset as our source demonstration. We do option-PPO training using only this demonstration and then execute the learned policy under randomized task configurations. Successful rollouts are retained until we collect 1,000 trajectories per task, matching the scale of the released DexMimicGen datasets.

We train image-based BC-RNN policies on the generated trajectories using the DexMimicGen robomimic pipeline, with absolute end-effector target poses as actions and all other training settings unchanged. Training is repeated over three seeds; following the DexMimicGen evaluation protocol, we report the best 50-episode periodic evaluation for each seed and aggregate performance across seeds.

Results. Table 2 summarizes both the success rate of the demonstration generator and the performance of policies trained on the resulting data. OneDemo achieves over 90% trajectory edit success rate on all nine DexMimicGen tasks, with an average of 98.4%. This reliability extends to diverse manipulation regimes. For example, *Transport* requires a long-horizon asymmetric bimanual sequence involving opening a container, manipulating multiple objects, and transferring an object between arms, whereas *Threading* requires precise coordinated motion between the two manipulators.

More importantly, our generated trajectories provide substantially better supervision for downstream policy learning. At matched dataset scale, BC-RNN trained on our demonstrations achieves near-perfect performance across all nine tasks, averaging 99.6% success compared with 76.3% reported

Table 2: **Demonstration-generation and downstream-policy results in DexMimicGen tasks (success rate, %).** Trajectory edit success is the fraction of randomized task configurations for which OneDemo successfully edits the source demonstration into a successful trajectory. Bold indicates the best downstream result per task.

Embodiment	Task	Trajectory Edit Success	Downstream Policy Success		
		OneDemo	DexMimicGen Data (Jiang et al., 2025)		OneDemo Data
			BC-RNN	Diffusion Policy	BC-RNN
Bimanual Panda + parallel-jaw grippers	Piece Assembly	94.1	66.0 ± 2.0	80.7 ± 0.9	$\mathbf{97.3 \pm 0.9}$
	Threading	100.0	68.0 ± 1.6	69.3 ± 1.9	$\mathbf{100.0 \pm 0.0}$
	Transport	99.7	57.3 ± 4.1	83.3 ± 0.9	$\mathbf{100.0 \pm 0.0}$
Bimanual Panda + dexterous hands	Box Cleanup	100.0	94.7 ± 0.9	92.0 ± 4.3	$\mathbf{100.0 \pm 0.0}$
	Drawer Cleanup	99.0	80.0 ± 0.0	76.0 ± 0.0	$\mathbf{100.0 \pm 0.0}$
	Tray Lift	99.9	78.0 ± 2.0	88.7 ± 0.9	$\mathbf{100.0 \pm 0.0}$
GR-1 humanoid + dexterous hands	Pouring	100.0	62.0 ± 7.5	79.3 ± 0.9	$\mathbf{100.0 \pm 0.0}$
	Coffee	93.5	84.7 ± 4.9	77.3 ± 0.9	$\mathbf{99.3 \pm 0.9}$
	Can Sorting	99.8	96.0 ± 4.3	97.3 ± 0.9	$\mathbf{100.0 \pm 0.0}$
Mean		98.4	76.3	82.7	**99.6**

by DexMimicGen. Our BC-RNN also outperforms the reported Diffusion Policy on every task, despite using a simpler recurrent behavior-cloning architecture.

The learned BC-RNN policy also matches or exceeds the rollout success rate of the data generator across all tasks, indicating that successful edited trajectories can be consolidated into a more reliable downstream policy. We hypothesize that the strong downstream performance arises from a combination of *broad task-configuration coverage* and *behavioral consistency*. Although OneDemo starts from only a single source demonstration, its generated data spans a task-configuration distribution comparable to DexMimicGen while exhibiting lower object-centric trajectory dispersion. Appendix A.7 provides a detailed analysis across all nine tasks.

Additional Results. We additionally evaluate OneDemo as a demonstration generator in a low-data regime on selected RoboTwin tasks. Diffusion Policy trained on only 50 OneDemo-generated trajectories per task improves average success from 28.5% to 48.3% across four bimanual tasks. Full results are provided in Appendix A.8.

5 Limitations and Future Work

OneDemo is designed for tasks in which successful behavior can be adapted through a sequence of object-centric trajectory edits. Extending it to richer perceptual inputs, such as images and point clouds, and to within-segment closed-loop correction could broaden its applicability to dynamic and contact-sensitive settings. Another extension is to handle changes in interaction mode, such as switching which arm manipulates an object when the original assignment becomes unreachable. The learned editor could also serve as a DAgger-style data-collection policy, providing corrective demonstrations on states visited by the downstream policy.

6 Conclusion

We introduced OneDemo, a reinforcement learning framework that turns a single successful demonstration into a structured search space for adapting behavior to new task configurations. Through learning when to transition between object-centric trajectory segments and how to geometrically edit their motion, OneDemo avoids searching directly over long sequences of low-level actions. The resulting trajectory editor serves both as a sample-efficient task policy and as a high-reliability demonstration generator. Across multiple benchmarks, we show that this formulation enables strong policy learning with limited online interaction and produces high-quality training data for downstream policies. Overall, our results suggest that a single successful demonstration can serve not only as supervision, but as a reusable behavioral scaffold for searching both policies and new training data.

AI USE STATEMENT

Generative AI tools were used to assist with writing refinement and figure presentation. An LLM was also used as part of the proposed method to automatically initialize the coarse semantic option sequence and associated task-relevant anchors from task metadata and downsampled telemetry, as described in Section 3.3 and Appendix A.3. All AI-assisted outputs were reviewed by the authors, who take responsibility for the scientific content, experiments, analyses, results, and conclusions.

REFERENCES

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

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*, 2025.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Caelan Reed Garrett, Ajay Mandlekar, Bowen Wen, and Dieter Fox. Skillmimicgen: Automated demonstration generation for efficient skill learning and deployment. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2750–2790. PMLR, 06–09 Nov 2025. URL https://proceedings.mlr.press/v270/garrett25a.html.

Nicklas Hansen, Yixin Lin, Hao Su, Xiaolong Wang, Vikash Kumar, and Aravind Rajeswaran. Modem: Accelerating visual model-based reinforcement learning with demonstrations. *arXiv preprint arXiv:2212.05698*, 2022.

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 *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 16923–16930. IEEE, 2025.

Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 international conference on robotics and automation (ICRA)*, pp. 6023–6029. IEEE, 2019.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. In *International Conference on Learning Representations*, volume 2025, pp. 29982–30009, 2025.

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

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021.

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. *arXiv preprint arXiv:2310.17596*, 2023.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 6292–6299. IEEE, 2018.

Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, and Jan Peters. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2): 1–179, 2018.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

Jun Wang, Yuzhe Qin, Kaiming Kuang, Yigit Korkmaz, Akhilan Gurumoorthy, Hao Su, and Xiaolong Wang. Cyberdemo: Augmenting simulated human demonstration for real-world dexterous manipulation. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 17952–17963. IEEE, 2024.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pp. 1094–1100. PMLR, 2020.

Haoqi Yuan, Ziye Huang, Ye Wang, Chuan Mao, Chaoyi Xu, and Zongqing Lu. Demograsp: Universal dexterous grasping from a single demonstration. In *International Conference on Learning Representations*, volume 2026, pp. 64609–64633, 2026.

Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. *arXiv preprint arXiv:2403.03954*, 2024.

Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

Allan Zhou, Moo Jin Kim, Lirui Wang, Pete Florence, and Chelsea Finn. Nerf in the palm of your hand: Corrective augmentation for robotics via novel-view synthesis. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 17907–17917. IEEE, 2023.

Kefei Zhu, Fengshuo Bai, YuanHao Xiang, Yishuai Cai, Xinglin Chen, Ruochong Li, Xingtao Wang, Hao Dong, Yaodong Yang, Xiaopeng Fan, and Yuanpei Chen. Dexflywheel: A scalable and self-improving data generation framework for dexterous manipulation. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 4236–4265. Curran Associates, Inc., 2025. doi: 10.52202/085713-0150. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/066c5542795287822f4f076cf330e5a2-Paper-Conference.pdf.

A Appendix

A.1 Background on the Options Framework

The options framework (Sutton et al., 1999) extends a Markov decision process (MDP) with temporally extended actions. Consider an MDP with state space $\mathcal{S}$ and primitive action space $\mathcal{A}$. A Markovian option $\omega \in \Omega$ is defined as a tuple

$$\omega = (I_\omega, \pi_\omega, \beta_\omega),$$

where $I_\omega \subseteq \mathcal{S}$ is the *initiation set*, $\pi_\omega(a \mid s)$ is the *intra-option policy*, and $\beta_\omega(s) \in [0, 1]$ is the *termination function*. The initiation set specifies the states in which the option may be started, the intra-option policy determines the primitive actions executed while the option is active, and the termination function gives the probability that the option terminates upon reaching a state.

Given a set of options Ω, a separate *policy over options*

$$\pi_\Omega(\omega \mid s)$$

determines which option to initiate. Under the standard *call-and-return* execution model, an option is sampled according to π_Ω and its intra-option policy is then followed until the option terminates. More precisely, if ω_t is active in state s_t, a primitive action is sampled according to the intra-option policy as

$$a_t \sim \pi_{\omega_t}(\cdot \mid s_t).$$

After the environment transitions to s_{t+1}, the active option terminates with probability $\beta_{\omega_t}(s_{t+1})$. If it does not terminate, execution continues under the same option; otherwise, a new option is selected according to $\pi_\Omega(\cdot \mid s_{t+1})$.

Thus, the options framework introduces two levels of decision making: the policy over options determines *which intra-option policy to execute*, while the intra-option policy determines *the actual primitive action carried out at each time step*. The termination function determines when to sample a new option. Because an option may persist for a variable number of primitive time steps, an MDP augmented with options naturally induces a semi-Markov decision process (SMDP) (Sutton et al., 1999) at the option-decision level.

The Option-Critic architecture (Bacon et al., 2017) shows that the intra-option policies and termination functions can be optimized directly from task return. These components are learned jointly with a policy over options, whose learning rule can be chosen separately.

A.2 Choice of Anchor Alignment

For each semantic option, OneDemo uses translation-only alignment of the associated anchor as its deterministic geometric initialization:

$$B_i = \mathrm{Trans}(p_i - p_i^{\mathrm{src}}).$$

We deliberately do not directly align the full source and live anchor poses.

There are two motivations for this choice. First, changes in anchor orientation do not always imply that the demonstrated end-effector motion should undergo the same rotation. This is particularly common for objects with rotational symmetries, where multiple object orientations correspond to equivalent manipulation geometry. Directly inheriting the full anchor rotation can therefore introduce unnecessary changes to an otherwise reusable trajectory.

Second, full pose alignment can rotate the demonstrated trajectory into kinematically difficult or infeasible end-effector configurations. Existing trajectory-generation systems such as MimicGen (Mandlekar et al., 2023) and DexMimicGen (Jiang et al., 2025) mitigate this issue in part by drawing from multiple source demonstrations and selecting source segments that are compatible with the current task configuration. OneDemo instead assumes only a single source demonstration.

We therefore use translation-only alignment as a conservative initialization and let the learned residual

$$\Delta T_i \in SE(3)$$

supply any additional translational or rotational correction required by the task. This preserves the demonstrated motion whenever it remains reusable while still allowing the policy to depart from the initialization when task success requires stronger geometric adaptation.

A.3 AUTOMATIC SEMANTIC OPTION INITIALIZATION

OneDemo requires an ordered sequence of task-relevant anchors that defines the semantic option sequence $\Omega = (\omega_1, \ldots, \omega_K)$. Each option ω_i is associated with an anchor object or task frame, whose source and live poses determine the baseline geometric transformation used during trajectory editing.

In principle, this structure could be specified manually. In our implementation, we initialize it automatically using an LLM. The model receives task metadata and downsampled telemetry from the successful source demonstration and proposes an ordered semantic decomposition together with the anchor associated with each option. This task requires only coarse semantic interpretation of the demonstration; the LLM does not infer precise temporal boundaries, contact dynamics, or low-level robot actions.

The predicted option sequence need not be minimal. For example, the LLM may separate reaching and grasping into two options even when both use the same object anchor. Such redundancy is tolerated by the learned duration policy, which can assign near-minimal duration to unnecessary options. The LLM therefore provides only a semantic scaffold, while OneDemo learns the temporal segmentation and geometric adaptation from task return.

Prompt. The exact prompt used in our DexMimicGen experiments is reproduced below. The bracketed metadata and telemetry fields are populated automatically for each source demonstration.

```
You are an expert in robosuite and DexMimicGen demonstration composition.

Analyze the successful bimanual demonstration below and generate a COMPLETE,
DIRECTLY LOADABLE DexMimicGen experiment JSON. Infer the semantic option
program from telemetry; no existing generation JSON is provided.

### Hand and telemetry conventions
- `right_*` telemetry is `right_hand`; `left_*` telemetry is `left_hand`.
- Infer each hand's ordered semantic options, their anchor objects, and their
  coordination. Do not infer frame indices, durations, or temporal boundaries;
  Option PPO learns option transitions.
- Use object poses, EEF motion, gripper motion, and termination signals as
  semantic evidence. Associate a termination signal with the option whose
  completion it describes, but do not estimate when the option ends.
- Treat signal ownership as a causal question. At each `signals_rising` event,
  use `eef_object_distances`, object motion, and gripper motion to identify
  which hand is actually manipulating the relevant object. Assign a global
  task signal to that hand's option; assign it to both hands only when both
  are still causally engaged in the same coordinated operation at activation.
- Respect explicit hand semantics in signal names such as `_right` and `_left`
  unless the telemetry clearly contradicts them.
- `object_ref` MUST exactly match one name in `object_refs_in_demo`. Never use
  a robot, EEF, TCP, gripper, action channel, or invented alias.
- A non-null `subtask_term_signal` MUST exactly match one name in
  `subtask_term_signals`. Do not invent signals.
- Associate each available termination signal with the option whose completed
  semantics it describes. This is a label association, not boundary inference.
  The final option for each hand MUST have `subtask_term_signal: null`.
- Merge approach and grasp preparation into an acquire phase anchored to the
  manipulated object. After acquisition, change the anchor to the receiving
  object, fixture, or goal that governs placement/insertion.
- Keep grasp, lift, carry, receive, and handoff options anchored to the
  manipulated or transferred object. Use a destination, receptacle, fixture,
  or goal as the anchor only for an option whose semantics are governed by
  placement, insertion, alignment, or release at that destination.
- Apply a conditional destination selector only to destination-anchored
  options. Do not replace the manipulated-object anchor of grasp, lift, carry,
  receive, or handoff options merely because the eventual destination is
  conditional.
- Preserve temporal order. Use concise snake_case phase names such as
  `acquire_object`, `place_object`, or `insert_object`.
- When both hands participate in the same coupled phase, give both
  corresponding phase entries the exact same snake_case `coordination_group`.
  Omit `coordination_group` for independent phases.
- If an option's live anchor depends on an episode condition, include
  `live_object_ref` with an exact condition name and a complete mapping from
  every observed condition value to an object in `object_refs_in_demo`.
  Infer this mapping from the condition-stratified semantic examples.
  Keep `object_ref` equal to the anchor used by the selected source episode.
  Omit `live_object_ref` for unconditional options.
- Before returning JSON, silently audit every option: verify that its anchor
```

```
  matches its semantic verb, every signal is assigned to the hand or
  coordinated hands causally responsible at activation, and conditional
  selectors appear only where the live anchor truly changes with the condition.
- Do not add tuning fields such as action_noise, offsets, interpolation steps,
  or selection strategies.

### Exact output contract
Return JSON only. Do not add `trajectory_status`, `task_name`, `evidence`,
comments, markdown, or any other fields. Copy the fixed boilerplate values
below exactly. Replace only angle-bracket placeholders and dynamically
populate the ordered phase mappings.

{
  "name": "<suggested_config_name from HDF5 metadata>",
  "type": "robosuite",
  "experiment": {
    "name": "demo",
    "source": {
      "dataset_path": null,
      "filter_key": null,
      "n": null,
      "start": null
    },
    "generation": {
      "path": null,
      "guarantee": false,
      "keep_failed": true,
      "num_trials": 10,
      "select_src_per_subtask": false,
      "transform_first_robot_pose": false,
      "interpolate_from_last_target_pose": true
    },
    "task": {
      "name": "<env_name from HDF5 metadata>",
      "robot": null,
      "gripper": null,
      "env_meta_update_kwargs": {},
      "interface": "<env_interface_name from HDF5 metadata>",
      "interface_type": "<env_interface_type from HDF5 metadata>"
    },
    "max_num_failures": 50,
    "render_video": true,
    "num_demo_to_render": 50,
    "num_fail_demo_to_render": 50,
    "log_every_n_attempts": 50,
    "seed": 1
  },
  "obs": {
    "collect_obs": true,
    "camera_names": [],
    "camera_height": 84,
    "camera_width": 84
  },
  "task": {
    "task_spec": {
      "right_hand": {
        "<ordered_phase_name>": {
          "object_ref": "<exact object_refs_in_demo name>",
          "subtask_term_signal":
              "<exact subtask_term_signals name or null>",
          "coordination_group":
              "<shared group only when coordinated; otherwise omit this field>",
          "live_object_ref": {
            "condition": "<exact observed condition name>",
            "cases": {
              "<observed value>": "<exact object_refs_in_demo name>"
            }
          }
        }
      },
      "left_hand": {
        "<ordered_phase_name>": {
          "object_ref": "<exact object_refs_in_demo name>",
          "subtask_term_signal":
              "<exact subtask_term_signals name or null>",
          "coordination_group":
              "<shared group only when coordinated; otherwise omit this field>",
          "live_object_ref": {
            "condition": "<exact observed condition name>",
            "cases": {
              "<observed value>": "<exact object_refs_in_demo name>"
```

```
            }
          }
        }
      }
    }
  }
}

[TRAIN CONFIG METADATA START]
<automatically populated training configuration metadata>
[TRAIN CONFIG METADATA END]

[HDF5 METADATA START]
<automatically populated HDF5 metadata>
[HDF5 METADATA END]

[CONDITION-STRATIFIED SEMANTIC EXAMPLES START]
<automatically populated condition-stratified semantic examples>
[CONDITION-STRATIFIED SEMANTIC EXAMPLES END]

[DOWNSAMPLED TELEMETRY START]
<automatically populated trajectory telemetry>
[DOWNSAMPLED TELEMETRY END]
```

A.4 Source Demo Selection

The least-motion demonstrations used in our main experiments are not particularly favorable source trajectories: several randomly selected sources have substantially higher naive success. Nevertheless, OneDemo substantially reduces sensitivity to source choice. The performance range across the four sources decreases from 34.0 to 14.0 percentage points on Box Close and from 94.4 to 14.4 points on Basketball after training. In the most extreme case, a Basketball source with only 3.2% naive success reaches 98.0% best success after adaptation.

Table 3: Sensitivity to source-demonstration selection on Meta-World Box Close and Basketball. We compare the least-motion demonstration used in the main experiments with three randomly selected successful source demonstrations. Values are mean $\pm$ standard deviation over five training seeds. Naive denotes performance before RL adaptation, and OneDemo denotes best checkpoint performance, consistent with Table 1.

Task	Source Demonstration	Naive	OneDemo
Box Close	Least-motion demo	56.8 ± 8.1	83.6 ± 14.5
	Random demo 1	88.4 ± 1.5	97.2 ± 1.6
	Random demo 2	72.8 ± 5.2	97.6 ± 1.5
	Random demo 3	90.8 ± 5.2	96.4 ± 1.5
Basketball	Least-motion demo	67.6 ± 15.7	84.4 ± 5.6
	Random demo 1	3.2 ± 2.4	98.0 ± 2.5
	Random demo 2	97.6 ± 2.3	98.8 ± 1.6
	Random demo 3	78.0 ± 3.3	88.0 ± 2.8

A.5 Implementation Details

Option-PPO Architecture. Across all benchmarks, we use the same decision-level actor–critic architecture for Option-PPO. At each option boundary, the policy observation is encoded by a two-layer MLP with 256 hidden units per layer and Tanh activations. A categorical duration head predicts a distribution over discrete option-duration bins; in bimanual settings, each arm has a separate duration head. Duration bin b advances the source trajectory by $(b + 1)c$ steps, where c is the benchmark-specific commitment interval.

A linear value head predicts the scalar state value. The residual policy concatenates the shared trunk features with the raw policy observation, passes the result through an additional 256-unit Tanh layer, and predicts the mean of a diagonal Gaussian residual distribution. The residual policy uses a learned, state-independent log standard deviation for each action dimension.

Table 4: Hyperparameters shared by Option-PPO across benchmarks.

Hyperparameter	Value
Policy/value trunk	2-layer MLP
Hidden width	256
Activation	Tanh
Optimizer	Adam
Adam ϵ	10^{-5}
PPO clipping coefficient	0.2
Value-loss coefficient	0.5
Maximum gradient norm	0.5
Advantage normalization	per batch
Discount factor / GAE	not used
Return	terminal success
Residual distribution	diagonal Gaussian
Residual log std.	learned, state-independent

Table 5: Benchmark-specific Option-PPO settings.

Hyperparameter	Meta-World	DexMimicGen	RoboTwin
Learning rate	3×10^{-4}	3×10^{-4}	3×10^{-4}
LR schedule	constant	constant	linear decay
PPO epochs/update	4	4	4
Minibatches/update	2	32	2
Episodes per PPO update	100–188, task-dependent	2,048	60 or 120, task-dependent
Number of PPO updates	16	100	10 or 20, task-dependent
Commitment interval c	10	5	20
Residual action	3-D translation	6-DoF translation + rotation	6-DoF translation + rotation
Residual frame	target XYZ offset	anchor frame	anchor frame

We optimize the duration and residual policies using clipped PPO. Training alternates between segmentation updates, which optimize the duration policy, and warp updates, which optimize the residual policy; the inactive component is frozen during each stage. During segmentation updates, the residual action is set to the mean of the current Gaussian residual policy.

PPO Optimization. Table 4 summarizes the shared setup parameters across benchmarks. The network architecture is shared across benchmarks, while the residual action representation, commitment interval, optimization schedule, and rollout budget reflect the corresponding control interface. Table 5 summarizes the principal differences.

MoDem Baseline. We use MoDem (Hansen et al., 2022) as a reference baseline for the Meta-World experiments. MoDem is a demonstration-guided reinforcement learning method designed for sample-efficient learning under sparse rewards. It uses demonstrations to initialize the policy and guide subsequent online reinforcement learning, but does not constrain exploration to structured edits of a demonstrated trajectory. We therefore use our local reproduction of MoDem to contextualize the interaction scale required when demonstrations guide sparse-reward policy learning without imposing the reduced trajectory-editing action space from OneDemo.

A.6 Geometric Adaptation on DexMimicGen Threading

To further isolate the role of geometric residual adaptation, we compare the segmentation-only and full OneDemo policies on DexMimicGen Threading. Threading requires precise coordinated bimanual motion, making translation-only anchor alignment substantially less reliable than on many Meta-World tasks.

Under the same 20M environment-interaction budget, the segmentation-only variant reaches 32.1% success, whereas the full OneDemo policy with learned geometric residuals reaches 81.8%. This large gap shows that temporal adaptation alone is insufficient on this task: successful execution requires substantial geometric correction of the demonstrated trajectory.

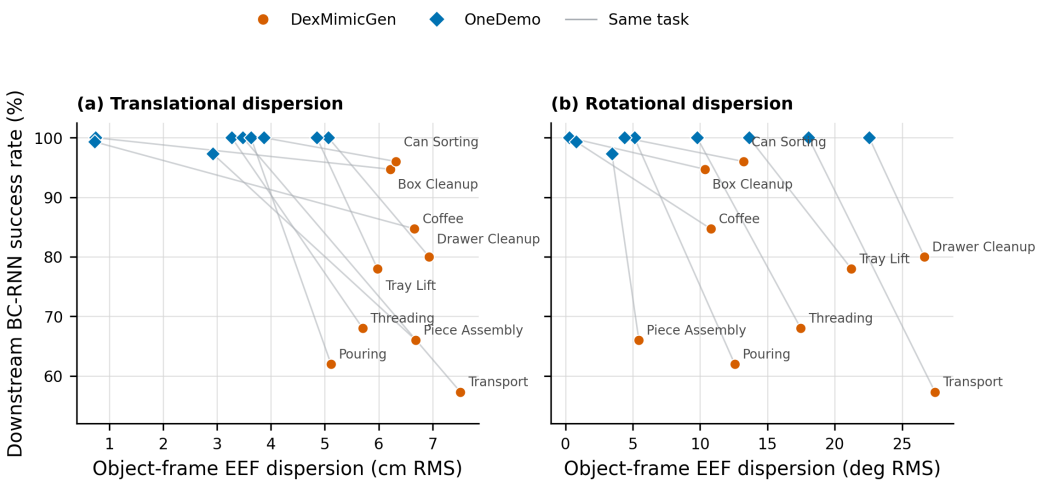

Figure 4: Object-centric behavioral dispersion and downstream imitation-learning performance across all nine DexMimicGen tasks. Each gray line connects DexMimicGen and OneDemo data for the same task. The left and right panels show mean phase-wise translational and rotational end-effector trajectory dispersion, respectively. OneDemo consistently produces lower behavioral dispersion while improving downstream BC-RNN success.

A.7 Generated Data Analysis

To better understand the strong downstream imitation-learning performance of OneDemo-generated data, we compare its task-configuration coverage and behavioral variation against the released DexMimicGen demonstrations.

Despite using only a single source demonstration, OneDemo preserves essentially the full task-configuration diversity of the released DexMimicGen datasets. Across all nine tasks, the OneDemo-to-DexMimicGen task-configuration coverage ratio ranges from $0.990\times$ to $1.016\times$, with a mean of $1.003\times$. Here, coverage is measured from the 95% span of each randomized object-pose dimension and aggregated across dimensions within each task.

We next analyze variation in the demonstrated robot behavior after factoring out this task-configuration variation. We express each end-effector trajectory in its corresponding object-anchor frame and temporally normalize each semantic phase. For each phase, we measure the RMS translational and rotational deviation from the cross-demonstration mean trajectory, and average this dispersion over normalized time and phases.

Figure 4 relates this object-centric behavioral dispersion to downstream BC-RNN performance. Each line connects DexMimicGen and OneDemo for the same task. Across all nine tasks, OneDemo-generated data exhibits lower average translational and rotational dispersion while the corresponding downstream BC-RNN also achieves higher success.

These results are consistent with the hypothesis that OneDemo achieves a favorable combination of *coverage* and *consistency*: starting from a single source demonstration, it generates trajectories across a broad range of task configurations while largely preserving a consistent object-relative manipulation strategy. This provides a plausible explanation for why the generated demonstrations can be more easily consolidated into an imitation-learning policy.

A.8 RoboTwin

RoboTwin provides a challenging setting for OneDemo because many tasks admit multiple behavior modes that cannot be obtained through geometric adaptation of a single trajectory. In particular, a task may require different arm assignments depending on the scene configuration, whereas OneDemo currently adapts behavior within a fixed interaction mode rather than switching between modes.

For *Handover Block* and *Pick Dual Bottles*, a single behavior mode was sufficient, so we use one source demonstration to train Option-PPO and generate trajectories. *Beat Block Hammer* and *Place Empty Cup* require two behavior modes, corresponding to performing the manipulation with the left or right arm. For these tasks, we select one source demonstration per mode and treat the two modes as separate adaptation problems, training an independent trajectory editor for each. The generated trajectories are then used to train the downstream Diffusion Policy. Results are shown in Table 6.

Table 6: RoboTwin results on four bimanual tasks. *Trajectory edit success* is the fraction of randomized task configurations for which OneDemo successfully edits the source demonstration into a successful trajectory. *DP Base* denotes our reproduced Diffusion Policy performance using RoboTwin's released demonstrations, while *DP OneDemo* denotes Diffusion Policy performance using OneDemo-generated demonstrations. Numbers in parentheses indicate absolute percentage-point improvement over *DP Base*.

Task	**Trajectory Edit Success**	**DP Base**	**DP OneDemo**
Beat Block Hammer	96.7	42	**56** (+14)
Handover Block	98.3	15	**20** (+5)
Pick Dual Bottles	98.3	25	**52** (+27)
Place Empty Cup	96.7	32	**65** (+33)
Average	97.5	28.5	**48.3** (+19.8)