TRACK: OBJECT KEYPOINTS AND END-EFFECTOR WAYPOINTS AS GEOMETRIC TASK REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Executing a new manipulation task requires translating a task specification into actions suited to the robot. Geometric specifications obtained from demonstrations or generated by planners can describe the intended behavior, but their hand or end-effector trajectories generally cannot be replayed directly to accomplish the task. We propose TRACK, a framework for specifying and executing manipulation tasks through object keypoint trajectories, abstract grasp flags, and approximate hand or end-effector waypoints. Object trajectories and grasp flags specify desired object motion and grasping interactions, while hand waypoints provide soft guidance that allows deviations needed for successful manipulation. This shared representation accommodates specifications from scripted experts, motion planners, and vision-language models. We train a policy across multiple tasks using reinforcement learning, with reference-derived rewards for object motion, grasping, and end-effector tracking within a positional tolerance, without task-specific success rewards. A state-based progress tracker aligns the policy's reference window and training reward with the robot's execution progress. At test time, the policy receives a single task specification and executes the task without additional training. Experiments on single-arm and bimanual manipulation demonstrate generalization to unseen tasks from a single task specification. TRACK outperforms reference replay with approximate end-effector trajectories and imitation learning-based baselines, and also executes specifications generated by a vision-language model.

1 INTRODUCTION

Many manipulation tasks share a common structure: the hand approaches an object, grasps or touches it, and moves it along a path toward a goal. Humans exploit this structure to learn a new task from a single demonstration by watching another person demonstrate it once and then reproducing the motion. A robot policy with the same ability could be deployed on a new task by showing it a task only once, without collecting large amounts of task-specific data and (re-)training. In this work, we therefore aim to train a robot policy that performs a manipulation task given a single task specification from a source other than the policy itself, such as a scripted expert, a motion planner, or a vision-language model (VLM).

In this paper, we propose **TRACK** (**T**ask **R**epresentations and **A**ction **C**onditioning with **K**eypoints), in which a task is specified by a waypoint trajectory of the demonstrator's hand or end-effector, annotated with a binary grasping flag, together with 3D keypoint tracks on the manipulated object. We call this the task specification, or the reference for short. All three quantities are embodiment-independent, allowing task specifications to come from any embodiment or even higher-level planners. Crucially, the grasping flag represents an abstract grasping intention rather than a simple gripper action, allowing transfer between, for example, dexterous hands and jaw grippers.

TRACK addresses two difficulties in executing a geometric specification. The reference may not provide precise end-effector paths, and its timing may not match the robot's execution. Object motion and grasp intent provide interaction targets, while a positional tolerance allows deviations from approximate end-effector waypoints. Through state-based progress tracking, we align the reference with the policy's execution speed. A window of future goals exposes the policy to upcoming interactions. Together, these components let the policy use the reference as guidance rather than follow it rigidly. Within each robot and task pool, we train one policy with a shared reference-tracking

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.

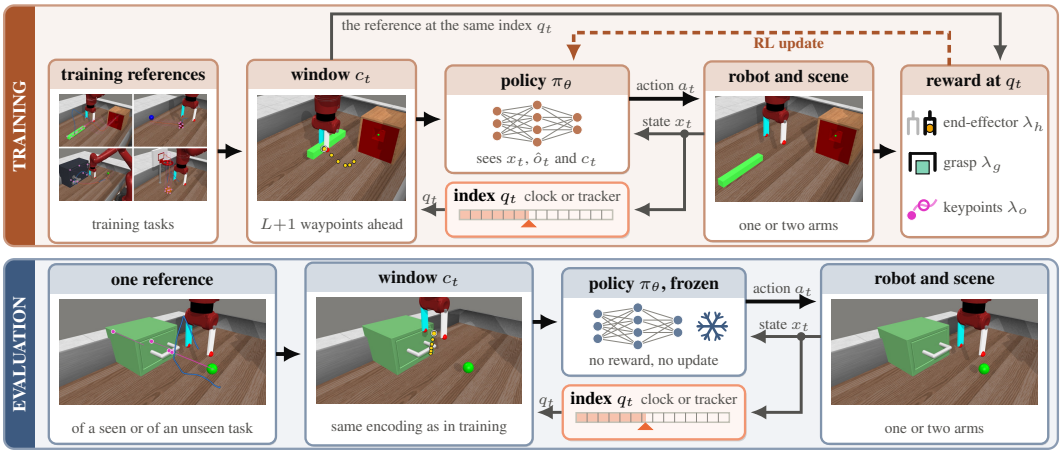


Figure 1: **TRACK in training and evaluation.** The policy observes state x_t, live keypoints $\hat{o}_t$, and a reference window c_t anchored at q_t. During training, a shared reward scores end-effector tracking, grasping, and object motion at that index (Equation 3). During evaluation, the frozen policy executes a single reference of a seen or unseen task. The clock or state-based tracker selects q_t in both settings (Equations 4–7).

reward, without explicit task identifiers or task-specific success rewards (Figure 1). At test time, the frozen policy receives one reference, with no additional training.

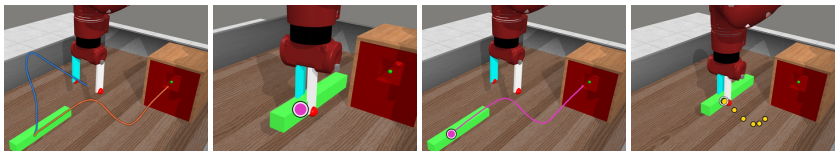
During training, the task specification supplies goal states that the policy needs to reach. Determining the appropriate next goal from the reference is not trivial, as simply advancing the goal each timestep can lead to the policy falling far behind the reference, making the supplied future goal trajectory meaningless. We therefore propose a progress tracker that uses the environment state to determine how far along the reference the policy currently is, and re-indexes the reference to supply appropriate next goals for the observations and, during training, the reward function. Our progress tracker further ensures forward progress of the policy by not letting it fall too far behind real-time reference following.

We evaluate TRACK on single-arm manipulation in Meta-World (Yu et al., 2019) and bimanual manipulation in RoboTwin 2.0 (Chen et al., 2026). The experiments test transfer to unseen tasks, execution of references with corrupted end-effector paths, and execution of VLM-generated specifications. Together, they study how a learned controller can reuse geometric task descriptions across tasks and specification sources.

To summarize, we make three main contributions: **(1)** We propose an embodiment-independent demonstration representation consisting of an end-effector trajectory, an abstract grasp flag, and object keypoint tracks. **(2)** We present a multi-task reinforcement learning approach in which tasks are defined only through such task specifications. **(3)** We introduce a progress tracker that allows the policy to follow the reference in a closed-loop manner, even when the robot's timing differs from that of the reference. We further extend our approach to bimanual manipulation, enabling coordinated actions between two arms using a shared reference index.

2 Related Work

Trajectory Tracking. Tracking a provided reference motion is a typical task in whole-body humanoid control (Peng et al., 2018; Liao et al., 2026) and robotic manipulation (Mandi et al., 2026; Li et al., 2025) settings. De Bakker et al. (2025) train a task-specific policy on a set of keypoint trajectories generated by a vision-language model planner. The policy then follows new plans of the same task. Most of the other methods train a single policy to track a single, fixed reference motion, which can come from a different embodiment than the one the policy controls. Liu et al. (2025) additionally distill motion-specific policies into a unified tracking controller for dexterous hands via reinforcement and imitation learning, but require a separate hand-specific retargeting step. Luo et al. (2026)



End-Effector Path h_k | Object Keypoint | Object Keypoint Track o_k | Reference Window at q_t

Figure 2: A reference trajectory of peg-insert-side on Meta-World with its corresponding end-effector path, grasp flag, and object keypoint track (left three images). The end-effector path is orange while the grasp flag is one and blue otherwise. The object carries a single keypoint ($K = 1$), placed where the left fingertip touches the peg (second image). The rightmost image shows the window from the reference end-effector trajectory observed by the policy at an intermediate execution state.

develop a single motion imitation policy for whole-body humanoid control, trained with RL without distillation. Xu et al. (2025) do not require a separate retargeting step and directly train a general trajectory-following policy with RL that receives a sequence of future goal states, but this policy is formulated for a single rigid object. More general Anypose-to-Anypose methods (Kuang et al., 2026; Kedia et al., 2026) train a single RL policy to move an object to arbitrary, sparsely sampled goal poses. Such methods are limited to single-object manipulation and do not support articulated objects. In both methods, an episode starts with the policy grasping an object on the table and then keeping it lifted throughout the episode. The implementations of Kuang et al. (2026) and Kedia et al. (2026) require the policy to reach each goal pose before switching to a new goal, without allowing the policy to adapt its tracking precision to improve overall trajectory tracking accuracy or providing a sequence of future goals. Im2Flow2Act (Xu et al., 2024) supplies a policy with the keypoint flow of target objects, but uses imitation learning on play data for training. TRACK conditions an RL policy on a window of densely sampled waypoints that come from coherent, task-solving trajectories. Through a grasp flag (Section 3.1), TRACK can perform more general tasks, including object pushing. We further introduce a flexible progress tracker (Section 3.4) that adapts to policy execution speed, in contrast to constant time-based replays (Liu et al., 2025), while ensuring forward progress and allowing flexible trajectory following.

One-Shot Learning. Lum et al. (2025) transfer a single demonstration into a simulation environment and then train a policy to track this demonstration, thus requiring training per demonstration. Anypose-to-Anypose policies (Kuang et al., 2026; Kedia et al., 2026) natively support one-shot generalization through trajectory following, without requiring per-demonstration training. TRACK places itself within this paradigm, but uses task-solving reference motions instead of random goal sequences. Dance et al. (2021) similarly condition a multitask policy on task-solving demonstrations, but train it with task-specific rewards instead of a tracking reward. In contrast, TRACK does not require any task-specific labels or rewards.

Temporally Aligned Tracking. Dynamic Time Warping (DTW) (Sakoe & Chiba, 1978) tackles the general problem of aligning two sequences progressing at different speeds using a dynamic programming approach. Recognizing that DTW does not ensure sufficient forward progress when aligning demonstrations in robot learning, Huey et al. (2025) introduce a per-timestep reward that encourages the policy to complete subgoals in the correct order. Haldar et al. (2022) use Optimal Transport between agent and reference trajectories as an RL reward, which matches the reference without selecting a single per-timestep goal, but does not enforce temporal order (Huey et al., 2025). TRACK uses a heuristic, DTW-inspired approach with the addition of a progress floor to ensure forward progress. To the best of our knowledge, TRACK is the first method that selects the reference index by state matching with a progress floor in multitask trajectory tracking for robot learning.

3 METHOD

We consider robotic manipulation tasks, where the goal is to control a robot through actions a, so that it achieves an arbitrary success criterion. We assume that the task is given by a task specification τ, i.e., a reference trajectory that solves the task. Crucially, τ may come from a different embodiment than the one we want to control and, in general, solves a new, unseen task. We then train an embodiment-specific policy that executes the reference τ.

3.1 Setting and Task Specification

The task specification supplies three quantities per time step that describe how the reference embodiment and task-relevant object moved and when objects were grasped:

$$\tau = \big(h_k,\, g_k,\, o_k\big)_{k=0}^{T-1}, \qquad h_k \in \mathbb{R}^{A\times 3}, \quad g_k \in \{0,1\}^A, \quad o_k \in \mathbb{R}^{N_o\times 3}, \tag{1}$$

where h_k are the end-effector positions of the demonstrator, g_k are the grasp flags, and o_k denotes the positions of $N_o = MK$ keypoints on the M manipulated objects. Figure 2 visualizes such a reference.

Grasp Flag. The grasp flag is a binary indicator showing whether the reference embodiment grasps the object at a specific time step. For example, for a parallel jaw gripper, it is one when the gripper pads exert force on an object, and zero otherwise. Compared to a simple gripper state, the grasp flag is transferable between embodiments, as it can also indicate a more general "grasping" state of a dexterous hand, or even an abstract grasp intent of a high-level task planner.

Object Keypoints. For each object, we sample K keypoints on its surface and track them throughout the episode while they move with the object. In simulation, we perform greedy farthest-point sampling, starting from contact sites and restricting to points visible to the camera.

Policy. Our goal is to learn a trajectory-conditioned policy $\pi_\theta(a_t \mid x_t, \hat{o}_t, c_t)$ that predicts robot actions given the observed state x_t, object keypoints $\hat{o}_t$, and a reference window c_t (Section 3.2). While π is embodiment-specific, it is not task-specific. It is trained on a dataset of references $\mathcal{D} = \{\tau_i\}$ that solve arbitrary tasks and can come from arbitrary embodiments. The reference specifies the task without an explicit task identifier, allowing a trained policy to execute new tasks from a single reference without further training.

3.2 Conditioning and Observations

The reference window c_t is a short local segment of the reference motion, anchored at the reference index q_t (Section 3.4):

$$c_t = \Big(\big(h_{q_t+\ell\delta},\, g_{q_t+\ell\delta},\, o_{q_t+\ell\delta}\big)_{\ell=0}^{L},\ \tfrac{q_t}{T}\Big), \tag{2}$$

as determined by a stride δ and lookahead L and clipped to the end of the reference. We use $\delta = 4$ and $L = 5$, yielding a total of twenty reference steps and allowing the policy to plan ahead, in particular with respect to future grasps. The first waypoint in the window corresponds to the one used by the reward function (Section 3.3). The state x_t is the native state of the respective benchmarks, but with goal information masked to prevent the policy from inferring the task from the state.

3.3 Reward

The reward function encourages the policy to track the reference motion. It is specified only through the current reference index q_t (Section 3.4). In particular, there is no task-specific dense reward or success bonus.

Using the shaping function $\rho_\varepsilon(d) = \exp\big(-\alpha \max(d-\varepsilon, 0)\big)$, the total reward for a robot with A arms that observes N_o keypoints is given by

$$r_t = \underbrace{\frac{\lambda_h}{A}\sum_{a=1}^{A}\rho_\varepsilon\big(\|p_t^a - h_{q_t}^a\|\big)}_{\text{End-Effector Tracking}} + \underbrace{\frac{\lambda_g}{A}\sum_{a=1}^{A} g_{q_t}^a\, \hat{g}_t^a\, \phi(F_t^a)}_{\text{Grasp Flag Tracking}} + \underbrace{\frac{\lambda_o}{N_o}\sum_{i=1}^{N_o}\rho_0\big(\|\hat{o}_t^i - o_{q_t}^i\|\big)}_{\text{Object Keypoint Tracking}} - \underbrace{\lambda_a\|a_t\|^2}_{\text{Action Penalty}} \tag{3}$$

where p_t^a is the end-effector position of arm a, $\hat{g}_t^a$ its achieved grasp flag and F_t^a the contact force on its gripper pads, and $\lambda_h, \lambda_g, \lambda_o$, and λ_a are scalar weights. The ε tolerance in the end-effector tracking reward allows the policy to deviate slightly from the reference end-effector path without being penalized, so it can better track the object keypoints and grasp flags, which more directly determine task success. The grasp flag tracking reward is based on a product of the reference grasp flag, the current observed grasp flag, and a shaped pad contact force $\phi(F) = \exp(-1/F)$. It thus rewards the policy for grasping when the reference grasped and for maintaining a high contact force.

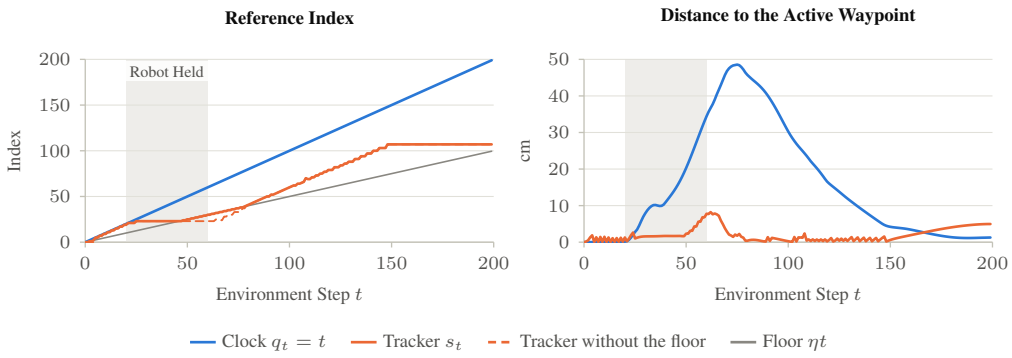


Figure 3: **Behavior of the clock and the tracker.** A reference is replayed on the robot, but the robot is held fixed for some time (shaded), simulating a slow policy. With the clock, execution starts to lag behind the reference, leading to large tracking error (right plot). Our tracker stops advancing goals as soon as the robot pauses, leading to lower tracking error and more meaningful goals. After roughly $2/3$ of the pause, the tracker hits the progress floor, forcing it to slowly advance again, ensuring forward progress.

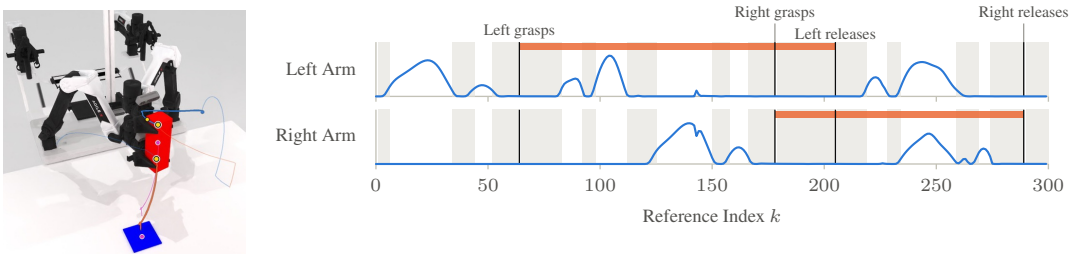


Figure 4: **Synchronization in a bimanual task.** A bimanual handover (left) with a corresponding reference (right). The blue curves show the speeds of each end-effector, while the orange bars indicate per-arm grasping flags. The shaded parts indicate periods without movement that are observed by the tracker for progressing the reference index. All grasps and releases signal synchronization events that stop the index from advancing until the corresponding goals are reached.

3.4 Progress Tracker

During training, we need to select the reference index q_t that is used to obtain the current waypoint for the reward function and the reference window c_t supplied to the policy. A simple way to choose q_t is to align the episode and reference time and use the reference index that is closest to the current episode time. We call this open-loop method the **Clock**.

If the policy is not able to keep up with the speed at which the clock switches to new waypoints, the current waypoint might run far ahead of the current state, leading to small rewards and sparse policy supervision, while making the future waypoints in the context window meaningless, as the policy struggles to reach the first one. Figure 3 visualizes this problem. We therefore propose an environment-based **Progress Tracker** that chooses waypoints close to what the policy has achieved so far while ensuring eventual forward progress. The tracker scores a band $\mathcal{B}_t = \{s_{t-1} - 20, \dots, s_{t-1} + 5\}$ of waypoints around the previous reference index s_{t-1} according to the aggregate similarity

$$d_t(k) = \underbrace{\frac{1}{A}\sum_{a=1}^{A}\frac{\|p_t^a - h_k^a\|}{\sigma_h^a}}_{\text{End-Effector}} + \underbrace{\frac{1}{N_o}\sum_{i=1}^{N_o}\frac{\|\hat{o}_t^i - o_k^i\|}{\sigma_o}}_{\text{Object Keypoints}} + \underbrace{\frac{1}{A}\sum_{a=1}^{A}|\hat{g}_t^a - g_k^a|}_{\text{Grasp Flag}} \tag{4}$$

where σ_h^a and σ_o are empirical standard deviations of the reference end-effector paths and object keypoints. To avoid jumps along the reference motion, we apply a distance-based switching cost β, yielding the optimal index

$$k^\star = \arg\min_{k \in \mathcal{B}_t} J_t(k), \quad J_t(k) = d_t(k) + \beta|k - q_{t-1}| \tag{5}$$

To further avoid high-frequency waypoint switching, we only switch to the new index when its similarity improves by more than a hysteresis κ compared to the current index:

$$\tilde{q}_t = \begin{cases} k^\star & \text{if } J_t(k^\star) < J_t(q_{t-1}) - \kappa, \\ q_{t-1} & \text{otherwise.} \end{cases} \tag{6}$$

Directly using $\tilde{q}_t$ as the reference index q_t would allow the policy to hover at the initial state while continuously collecting a tracking reward. We therefore introduce a progress floor ηt that ensures that the waypoints progress no slower than $\eta \leq 1$ times the reference motion:

$$q_t = \max\big(\tilde{q}_t,\ \lfloor \eta t \rfloor\big) \tag{7}$$

Compared to only switching to the next waypoint when the previous one was reached, our tracker allows the policy to track some waypoints with less precision if it allows it to track the reference as a whole more accurately. For example, it might choose a slightly different approach path towards an object if that is necessary for a stable grasp.

For bimanual tasks, both arms share a single reference index. However, because multiple arms need to be coordinated so that one does not progress faster than the other and thereby hurt task performance, every grasp or release by either arm in the reference is treated as a synchronization event. This means that the index may reach the next pending sync event but can be held there until the scene shows the corresponding grasp or release index may reach the next pending sync event but can be held there while the scene does not yet show that grasp or release. This can occur as the consequence that the similarity score Equation (4) estimates that the robot is in a state to move along to next way point before the critical sync has happened (Figure 4). On RoboTwin we set $\eta = 1$, so that the floor is the clock itself, and the tracker then differs from the clock only in that its index can run ahead of the clock when the robot is ahead of the reference and can wait at a sync event that the robot has not completed yet.

3.5 Policy and Training

The policy is a Gaussian MLP trained with proximal policy optimization (PPO) (Schulman et al., 2017). On Meta-World, it outputs an end-effector translation and a gripper action. On RoboTwin, it outputs a small translation residual relative to a reference replay controller which follows the reference's TCP pose and controls the grippers from the grasp flags. See Appendix A.2 for details. For training, we use a dataset of N_T tasks with $N_{D,T}$ reference motions per task. We balance parallel environments across tasks. Each environment cycles through its task's reference motions and restores the simulator state from the start of the reference motion.

4 Experiments

4.1 Setup

Benchmarks and References. We perform single-arm experiments on Meta-World (Yu et al., 2019) and bimanual experiments on RoboTwin 2.0 (Chen et al., 2026). For Meta-World, we use the ten training tasks from ML10 or the 45 training tasks from ML45 and test one-shot generalization on the corresponding five test tasks. For RoboTwin, we train on five tasks and test one-shot generalization on two unseen tasks. Per task, we collect reference motions from successful rollouts of each benchmark's scripted expert. Of these, we hold out 15 references for Meta-World and 5 for RoboTwin, with the rest forming the training dataset. To simulate cross-embodiment transfer, we corrupt references by perturbing end-effector positions, without corrupting object keypoint tracks and grasp flags. See Appendix A.4 for the tasks and details on reference collection and corruption. This setup captures a central difficulty of cross-embodiment transfer, where end-effector positions provide rough guidance but cannot be followed precisely without compromising object and grasp tracking.

Policy Training. We train TRACK on only 10 references per task on Meta-World and 5 on RoboTwin. Only corrupted references are used during training. Note that some baselines use more or different training data (see below). For Meta-World, we use a progress floor of $\eta = 0.5$, allowing the policy to lag behind the reference. For the bimanual RoboTwin setup, the synchronization events already

Table 1: **Results on Meta-World ML10 and ML45.** Success rates on training references, held-out references of training tasks, and five unseen tasks (one-shot). Im2Flow2Act uses exact references and expert actions. The other methods use corrupted end-effector trajectories. We report the mean and standard deviation over five seeds, except for the deterministic reference replay. The best result per block is shaded with for training references and for unseen references. TRACK outperforms reference replay in all settings, but only TRACK (tracker) yields high success rates in the one-shot setting. Imitation baselines overfit to the training tasks.

	ML10			ML45		
Method	Training	Held-out	One-shot	Training	Held-out	One-shot
Exact references (135 per task)						
Im2Flow2Act	0.99 ± 0.00	0.96 ± 0.01	0.22 ± 0.07	0.91 ± 0.00	0.90 ± 0.01	0.18 ± 0.05
Corrupted references (10 per task)						
Reference replay (no learning)	0.83	0.80	0.61	0.72	0.70	0.68
Diffusion Policy	0.52 ± 0.03	0.50 ± 0.04	0.38 ± 0.03	0.38 ± 0.01	0.42 ± 0.02	0.41 ± 0.04
TRACK (clock)	0.99 ± 0.02	0.88 ± 0.05	0.67 ± 0.07	0.88 ± 0.07	0.84 ± 0.07	0.55 ± 0.08
TRACK (tracker)	0.87 ± 0.01	0.82 ± 0.04	0.77 ± 0.07	0.84 ± 0.02	0.81 ± 0.04	0.74 ± 0.15

make the reference wait for the policy, allowing us to use a progress floor of $\eta = 1$, i.e., preventing the policy from running behind the reference outside of synchronization events.

Evaluation Protocol. At the start of each episode, we reset the simulator state to the initial state of the current reference. We report success rates (success-once) of the corresponding task that the reference completes, averaged across five seeds unless stated otherwise. Per seed, we evaluate either on references that were used for training (**Training refs.**, 50 episodes), that are for training tasks but where the specific references are unseen (**Held-out refs.**, 20 episodes), or on references from unseen tasks (**One-shot**, 15 episodes). As tracker-based reference following (Section 3.4) can lead to slower task execution, we score TRACK with the tracker on Meta-World with a 500-step horizon in one-shot trials and on VLM plans (Section 4.4). All other Meta-World results use 200 steps.

Baselines. Reference replay replays the reference motion at a constant rate, without any separate policy. We control the gripper using a heuristic based on the grasp flag. Reference replay is evaluated on corrupted references, without access to the exact references, to demonstrate the benefit of a separate learning step in this setting. We include two imitation learning baselines: First, we train an off-the-shelf convolutional **Diffusion Policy** (Chi et al., 2023) on reference motions. As robot actions are not part of the reference, the Diffusion Policy imitates reference-derived end-effector displacements and gripper commands, conditioned on proprioception, an achieved-grasp indicator, and a hand/grasp reference window. These inputs differ from TRACK's inputs. We train and evaluate it on corrupted references. We also report its one-shot success after fine-tuning on the single reference of the unseen task. **Im2Flow2Act** (Xu et al., 2024) conditions its policies on 2D flow instead of 3D keypoint trajectories. Unlike Xu et al. (2024), who train on random-exploration play data and track points in rendered images, our variant receives the scripted expert's actions as training targets and ground-truth keypoint flow from the simulator, making it not directly comparable to the other baselines. Since Im2Flow2Act accesses privileged ground-truth keypoint flow and expert actions, it is not affected by reference corruption and therefore only trained and evaluated on clean references. Im2Flow2Act is trained on 135 references per task, and the Diffusion Policy on only the ten references per task that TRACK uses. See Appendix A.6 for additional implementation details.

4.2 Single-Arm Manipulation

Table 1 shows the success rate of TRACK and baselines on both Meta-World splits, using the (corrupted) references obtained from a scripted policy.

TRACK relative to reference replay. The setting-dependent success rates of reference replay are due to its highly task-dependent performance, leading to high variation across the particular tasks in each setting. It also suffers from not having access to a reference gripper state, forcing it to rely on a grasping heuristic derived from the high-level grasp flag and making it fail frequently on tasks that require grasping. TRACK outperforms reference replay on corrupted references in every setting except one-shot on ML45, where the tracker performs similarly to replay, with a lower

Table 2: **Bimanual results on RoboTwin 2.0 with corrupted references.** Success rates on training references, held-out references of training tasks, and two unseen tasks (one-shot). For TRACK, we report mean and standard deviation over five seeds. Reference replay is deterministic. The highest mean of each column is shaded. While TRACK with and without the tracker perform similarly on in-distribution tasks, the tracker shows significant improvements in the one-shot setting.

Method	Training refs.	Held-out refs.	One-shot
Reference replay (no learning)	0.68	0.64	0.57
TRACK (clock)	0.70 ± 0.05	0.73 ± 0.01	0.59 ± 0.05
TRACK (tracker)	0.70 ± 0.06	0.73 ± 0.02	0.69 ± 0.04

success rate achieved only by TRACK (clock). Overall, these results demonstrate the benefits of our learning-based approach for trajectory following when precise end-effector waypoints are unavailable.

TRACK relative to imitation baselines. While Im2Flow2Act performs well on in-distribution tasks, it significantly loses performance in the one-shot setting of out-of-distribution tasks, indicating overfitting on the training tasks. TRACK is able to keep a high success rate across all settings. Even in the in-distribution case, Im2Flow2Act benefits from the privileged keypoint flow and expert actions that are not available to TRACK, making its performance not directly comparable to the other methods. Reducing Im2Flow2Act's training set to the ten references per task that TRACK trains on lowers its success rate across all ML10 evaluation settings (Appendix B.3). The reference-conditioned Diffusion Policy achieves lower success rates than reference replay, as expected given that it suffers from the same gripper information bottleneck as reference replay, but adds an imperfect learning step. Fine-tuning the Diffusion Policy on one reference of each unseen task improves its one-shot success rate on ML10 to 0.52 ± 0.02 but not on ML45 (0.40 ± 0.05), both below the one-shot success rate of TRACK.

Ablations. The clock achieves higher in-distribution success rates than the tracker. However, the tracker improves one-shot success rates. We hypothesize that the tracker helps on unseen tasks because it lets the policy execute more where necessary, for example, when a grasp takes more steps than in the reference. Removing the end-effector path from the reference, including from the reward, more than halves TRACK's success rate (Appendix B.2). This is possibly due to the increased difficulty of learning, as object keypoints do not provide strong guidance for where to move and where to grasp, especially when objects remain static. Similarly, we find that removing the lookahead from the policy's reference window reduces success rates across all evaluation settings (Appendix B.1).

4.3 Bimanual Manipulation

Both variants of TRACK improve upon reference replay in all settings (Table 2). On the in-distribution tasks, these gains come mainly from the tasks that use both arms, where reference replay fails in most episodes. As with the single-arm experiments, we find that the tracker only yields significant benefits in the one-shot setting, where TRACK (clock) performs significantly worse than TRACK (tracker). Overall, these results demonstrate that TRACK transfers well to the bimanual setting, with the same relative performance of the clock and the tracker as observed on the single-arm Meta-World benchmark.

4.4 References from a Vision-Language Model

We also test frozen TRACK policies on interpolated plans from a vision-language model (VLM).

Setup. Following the setup of de Bakker et al. (2025), we use a quantized Qwen3.8-27B (Qwen Team, 2026) vision-language model to generate references. The VLM receives a scene image with object and goal markers, task and motion descriptions, and coordinates. It outputs ten end-effector waypoints with grasp flags, interpolated into a 200-step reference with pauses at grasp flag transitions. Object keypoints move rigidly with the end-effector while the grasp flag is true and remain stationary otherwise. During "one-shot" planning, the prompt also includes a scripted reference from the same task in a different scene. Policies are never trained on VLM plans, only on corrupted scripted

Table 3: **The Meta-World policies of Table 1 on references planned by a VLM.** The TRACK policies were only trained on corrupted scripted references, not on VLM references. Success rates are averaged over five seeds and over all tasks from the respective Meta-World split. The highest success rate per setting is shaded. TRACK (tracker) outperforms reference replay and TRACK (clock) in all settings, highlighting the importance of the tracker in the one-shot setting.

	Training tasks		Unseen tasks	
Method	Zero-shot plan	One-shot plan	Zero-shot plan	One-shot plan
ML10 policies, 10 and 40 tasks				
Reference replay (no learning)	0.27	0.40	0.21	0.35
TRACK (clock)	0.28 ± 0.07	0.50 ± 0.11	0.19 ± 0.03	0.35 ± 0.06
TRACK (tracker)	0.35 ± 0.08	0.60 ± 0.06	0.25 ± 0.05	0.42 ± 0.04
ML45 policies, 45 and 5 tasks				
Reference replay (no learning)	0.23	0.39	0.13	0.08
TRACK (clock)	0.26 ± 0.03	0.48 ± 0.03	0.12 ± 0.05	0.22 ± 0.06
TRACK (tracker)	0.31 ± 0.02	0.52 ± 0.01	0.23 ± 0.08	0.25 ± 0.08

references. We evaluate one episode per reference across five policy seeds, reporting training and unseen tasks separately.

Results. Table 3 shows the success rates on Meta-World with VLM references. TRACK with the tracker consistently outperforms direct replay of the VLM-generated reference, as well as TRACK with the clock. This confirms our previous findings that the tracker is especially important in the one-shot setting, where policies struggle more with following the reference than on known tasks with references of known quality. Overall, success rates are, as expected, substantially lower than with scripted references (cf. Table 1) and improve significantly when the VLM is provided with an example plan (one-shot plan), indicating that the primary limitation is not policy performance, but the VLM planner. In particular, we observe low success rates on pushing and sweeping tasks, most likely due to the fixed object track generation. Conversely, the success rate remains high on reaching tasks and on most pressing tasks (Table 8 in Appendix B.4). Interestingly, on the training tasks, one-shot prompting helps TRACK substantially more than it does reference replay.

5 Conclusion

In this paper, we introduce TRACK, a framework for trajectory-conditioned robot learning that connects geometric task specifications to learned manipulation policies. We train a single policy across multiple tasks using a shared task-independent tracking reward. A state-based progress tracker aligns the policy's observed reference windows with execution actual execution speed. The trained, frozen policies are then deployed on unseen tasks. In simulated single-arm and bimanual tasks, TRACK improves over direct replay in most settings with corrupted references, as well as over imitation learning baselines. Furthermore, we show that TRACK can execute VLM-generated task specifications. These results support our claim that we can use task references for task specification and robot control.

Limitations and Future Work. All experiments are in simulation, so perception and keypoint tracking under occlusion and demonstrations from humans are not evaluated. While corrupted end-effector paths provide a proxy for cross-embodiment generalization, using, e.g., references from robotic hands remains to be evaluated. However, the VLM experiments evaluate a similar type of real cross-embodiment transfer.

In future work, we plan to evaluate TRACK on a real robot. Given the proposed representation, one could also use it for transferring policies trained in simulation to the real robot. We also plan to use human demonstrations as task specifications for unseen tasks. Moreover, our method could be adapted to a reference that was recorded in a different scene. Additionally, TRACK could be extended beyond grasp flags to more complex interaction types.

AI USE STATEMENT

Generative AI tools were used substantially to write and refactor the experimental code, inspect result artifacts, and draft and edit this manuscript under author direction. The authors reviewed the implementation, citations, claims and numbers.

ETHICS STATEMENT

This work uses simulated manipulation benchmarks and no human-subject or personal data.

REPRODUCIBILITY STATEMENT

Section 3 describes the method, and Appendix A.1 lists the hyperparameters of the TRACK runs behind Tables 1 and 2. Appendix A.3 details the progress tracker for two arms. Appendix A.4 lists the tasks and describes how references are collected and corrupted and how methods are evaluated, and Appendix A.6 describes the baselines. Appendix B.5 gives the prompts of the vision-language model verbatim. We plan to make the code available during the discussion period and will release it upon publication.

REFERENCES

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, Weiliang Deng, Yubin Guo, Tian Nian, Xuanbing Xie, Qiangyu Chen, Kailun Su, Tianling Xu, Guodong Liu, Mengkang Hu, Huan-ang Gao, Kaixuan Wang, Zhixuan Liang, Yusen Qin, Xiaokang Yang, Ping Luo, and Yao Mu. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. In *International Conference on Machine Learning (ICML)*, 2026.

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

Christopher R. Dance, Julien Perez, and Théo Cachet. Demonstration-conditioned reinforcement learning for few-shot imitation. In *International Conference on Machine Learning (ICML)*, volume 139 of *PMLR*, pp. 2376–2387, 2021.

Vincent de Bakker, Joey Hejna, Tyler Ga Wei Lum, Onur Celik, Aleksandar Taranovic, Denis Blessing, Gerhard Neumann, Jeannette Bohg, and Dorsa Sadigh. Scaffolding dexterous manipulation with vision-language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Siddhant Haldar, Vaibhav Mathur, Denis Yarats, and Lerrel Pinto. Watch and match: Supercharging imitation with regularized optimal transport. In *Conference on Robot Learning (CoRL)*, volume 205 of *PMLR*, pp. 32–43, 2022.

William Huey, Huaxiaoyue Wang, Anne Wu, Yoav Artzi, and Sanjiban Choudhury. Imitation learning from a single temporally misaligned video. In *International Conference on Machine Learning (ICML)*, volume 267 of *PMLR*, pp. 26283–26310, 2025.

Kushal Kedia, Tyler Ga Wei Lum, Jeannette Bohg, and C. Karen Liu. SimToolReal: An object-centric policy for zero-shot dexterous tool manipulation. In *Robotics: Science and Systems (RSS)*, 2026.

Yuxuan Kuang, Sungjae Park, Katerina Fragkiadaki, and Shubham Tulsiani. Dex4D: Task-agnostic point track policy for sim-to-real dexterous manipulation. *arXiv preprint arXiv:2602.15828*, 2026.

Kailin Li, Puhao Li, Tengyu Liu, Yuyang Li, and Siyuan Huang. ManipTrans: Efficient dexterous bimanual manipulation transfer via residual learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6991–7003, 2025.

Qiayuan Liao, Takara E. Truong, Xiaoyu Huang, Yuman Gao, Guy Tevet, Koushil Sreenath, and C. Karen Liu. BeyondMimic: From motion tracking to versatile humanoid control via guided diffusion. *Science Robotics*, 11(117):eadx8924, 2026. doi: 10.1126/scirobotics.adx8924.

Xueyi Liu, Jianibieke Adalibieke, Qianwei Han, Yuzhe Qin, and Li Yi. DexTrack: Towards generalizable neural tracking control for dexterous manipulation from human references. In *International Conference on Learning Representations (ICLR)*, 2025.

Tyler Ga Wei Lum, Olivia Y. Lee, C. Karen Liu, and Jeannette Bohg. Crossing the human-robot embodiment gap with sim-to-real RL using one human demonstration. In *Conference on Robot Learning (CoRL)*, volume 305 of *PMLR*, pp. 4418–4441, 2025.

Zhengyi Luo, Ye Yuan, Tingwu Wang, Chenran Li, Fernando Castañeda, Sirui Chen, Zi-Ang Cao, Jiefeng Li, David Minor, Qingwei Ben, Jinhyung Park, David Sami, Zi Wang, Xingye Da, Runyu Ding, Cyrus Hogg, Lina Song, Edy Lim, Eugene Jeong, Tairan He, Haoru Xue, Wenli Xiao, Simon Yuen, Jan Kautz, Yan Chang, Umar Iqbal, Linxi "Jim" Fan, and Yuke Zhu. SONIC: Supersizing motion tracking for natural humanoid whole-body control. *Science Robotics*, 11(117):eaed4592, 2026. doi: 10.1126/scirobotics.aed4592.

Zhao Mandi, Yifan Hou, Dieter Fox, Yashraj Narang, Ajay Mandlekar, and Shuran Song. DexMachina: Functional retargeting for bimanual dexterous manipulation. In *International Conference on Machine Learning (ICML)*, 2026.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics*, 37(4):143:1–143:14, 2018. doi: 10.1145/3197517.3201311.

Qwen Team. Qwen3.8-27B. Model card, `https://huggingface.co/Qwen/Qwen3.8-27B`, 2026.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978. doi: 10.1109/TASSP.1978.1163055.

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

Mengda Xu, Zhenjia Xu, Yinghao Xu, Cheng Chi, Gordon Wetzstein, Manuela Veloso, and Shuran Song. Flow as the cross-domain manipulation interface. In *Conference on Robot Learning (CoRL)*, volume 270 of *PMLR*, pp. 2475–2499, 2024.

Sirui Xu, Yu-Wei Chao, Liuyu Bian, Arsalan Mousavian, Yu-Xiong Wang, Liang-Yan Gui, and Wei Yang. Dexplore: Scalable neural control for dexterous manipulation from reference-scoped exploration. In *Conference on Robot Learning (CoRL)*, volume 305 of *PMLR*, pp. 2184–2199, 2025.

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 (CoRL)*, volume 100 of *PMLR*, pp. 1094–1100, 2019.

A Implementation Details

A.1 Hyperparameters

Table 4 lists the settings of the TRACK runs behind Tables 1 and 2. Within a benchmark, the clock and tracker arms differ only in how the reference index q_t is chosen, and the ML10 and ML45 runs differ in the task pool, the number of parallel environments, the rollout length and the training budget. The clock uses $q_t = \min(t,\, T-1)$. The tracker is initialized at $s_0 = 0$ and updated once per environment step from the state after the step, and σ_h^a and σ_o are the root-mean-square distances of a reference's end-effector path and keypoint tracks from their own means, computed per reference. On Meta-World the achieved grasp $\hat{g}_t$ is one when the gripper is closed further than 90% of its opening and the contact force between gripper and object exceeds 1 N, and F_t is the summed contact force between gripper and object. The policy is a diagonal Gaussian with a state-independent standard deviation, initialized to one on Meta-World and to 0.125 on RoboTwin, and it acts with its mean during evaluation.

Table 4: **Hyperparameters of TRACK.** Where a row has one value, it holds for both benchmarks.

	Meta-World	RoboTwin 2.0
Reference and episode length T	200	300
Training references per task	10	5
Keypoints per object K	1	4
Training budget	6M steps (ML10), 27M steps (ML45)	3M steps
Window stride δ, lookahead L	4, 5	
Reward weights $(\lambda_h, \lambda_g, \lambda_o, \lambda_a)$	$(3,\ 2,\ 1,\ 10^{-3})$	$(1,\ 1,\ 3,\ 10^{-3})$
Shaping α	$20\,\mathrm{m}^{-1}$	$20\,\mathrm{m}^{-1}$ (end-effector), $5\,\mathrm{m}^{-1}$ (object keypoints)
Tolerance ε	2 cm (end-effector term only)	
Tracker band $\mathcal{B}_t$	$\{s_{t-1}-20, \ldots, s_{t-1}+5\}$ (on RoboTwin over merged indices)	
Tracker switching cost β, hysteresis κ	0.02, 0.05	0.005, 0.01
Progress floor η	0.5	1
PPO implementation	Stable-Baselines3 (Raffin et al., 2021)	
Rollout steps per environment	256 (ML10), 512 (ML45)	256
Minibatch size, epochs per update	512, 10	
Learning rate	3×10^{-4}, constant	
Discount γ, GAE λ, clip range	0.99, 0.95, 0.2	
Entropy coef., value coef., max. gradient norm	0, 0.5, 0.5	
Observation normalization	running mean and variance of x_t, clipped at 10; reference inputs raw	
Reward normalization	off	on
State encoder	MLP 128–128, ELU	
Reference encoder	MLP 256–256–128, ELU after the hidden layers	
Policy and value heads	MLP 256–256, tanh, separate encoders for actor and critic	
Checkpoint	best periodic evaluation*	highest held-out success†
Seeds	5	

*Every 200k environment steps, five deterministic episodes per training task on the training references. †Every 0.5M environment steps, 20 episodes per training task on the held-out references; the unseen tasks are not used.

A.2 RoboTwin Controller

For each arm a, the policy outputs a normalized translation residual $a_t^a \in [-1, 1]^3$. Given the current commanded tool-centre position u_{t-1}^a, its Cartesian target before inverse kinematics and joint-limit enforcement is

$$u_t^{a,\mathrm{target}} = u_{t-1}^a + \mathrm{clip}\big(h_{q_t}^a - u_{t-1}^a,\ -b,\ b\big) + b\, a_t^a, \qquad b = 2\,\mathrm{cm}, \tag{8}$$

where clipping is per axis. The residual is applied afresh at each step, rather than accumulated as a persistent offset. The wrist follows the orientation recorded in the reference, and the gripper controller closes or opens according to its grasp flag. If the gripper remains closed without holding an object for 15 steps while a grasp is requested, the controller opens it for eight steps before retrying. Reference replay uses this same controller with zero residual and the clock index.

A.3 Progress Tracker for Two Arms

On RoboTwin, the tracker keeps a single index s_t for both arms instead of one tracker per arm, which could let the arms drift apart in time. The waypoints of both arms in the window and in the reward

therefore always come from the same step of the reference, and the arms keep the relative timing of the reference. Equation 4 averages the end-effector and grasp terms over the $A = 2$ arms. Compared to the single-arm tracker, the index is additionally gated by sync events, and the matching runs over merged reference indices.

Sync events. Because the cost averages over both arms and all three terms, a grasp that one arm misses raises it only partially, and the matcher, or at the latest the progress floor, could move the index on into a part of the reference in which that arm carries an object that it does not hold. The window and the reward would then describe a motion that can no longer happen. Every change of an arm's reference grasp flag is therefore a sync event e with index k_e, a grasp where the flag of arm a_e switches from zero to one and a release where it switches back. A hold that lasts until the end of the reference has no release event. The events of both arms are sorted by index into one list, and $e^\star$ denotes the first event that has not been completed. It is completed once $s_{t-1} \geq k_{e^\star} - 1$ and the achieved grasp of its arm shows it, that is $\hat{g}_t^{a_{e^\star}} = 1$ for a grasp and $\hat{g}_t^{a_{e^\star}} = 0$ for a release. On RoboTwin, $\hat{g}_t^a$ is one when the normalized opening of the gripper of arm a is below 0.8 and its finger links are in contact with one of the task's movable objects, so pressing the closed gripper onto the table does not count. The floored index of Equation 7 is capped at the pending event,

$$q_t = s_t = \min\Big(\max\big(\tilde{s}_t,\ \lfloor \eta t \rfloor\big),\ k_{e^\star} \Big), \tag{9}$$

so the floor can push the index up to a pending event but not past it, and while the index waits at the event, the waypoints of both arms wait with it. After the last event, the cap is removed.

Filtering the events. The reference grasp flag is a threshold on the contact force and can flicker, and a spurious release inside a carry would become an event that a correct execution never shows. Before the events are read off, gaps of fewer than ten indices between two holds of an arm are therefore closed, and holds of fewer than ten indices are then removed. The filter applies only to the event list, while Equation 4 and the reward use the unfiltered flag.

Merged indices. The references contain segments in which neither arm nor any object moves. On such a segment, every candidate in the band is the same pose, so the matcher cannot tell how far into the segment the robot is, and once the segment is longer than the forward part of the band, it cannot find a later index that fits better. Equations 4 and 5 are therefore evaluated on merged states. Consecutive indices between which no end-effector and no object keypoint moves by more than 5 mm and no grasp flag changes form one state, and the band $\mathcal{B}_t$ and the switching cost β count merged states. While the matcher stays in a merged state, the index advances by one per step through the indices of that state, as the clock would, and when the matcher moves to another state, the index jumps to that state's first index. The floor and the cap of Equation 9 act on this index, and the window and the reward read the original reference at it, so merging does not discard any part of the reference.

A.4 Tasks and References

Meta-World. The ML10 training tasks, as defined in the Meta-World code, are reach, push, pick-place, door-open, drawer-close, button-press-topdown, peg-insert-side, window-open, sweep and basketball, and the ML10 test tasks are door-close, drawer-open, lever-pull, shelf-place and sweep-into. The ML45 training tasks are the 45 training tasks of the ML45 split of Yu et al. (2019), and its test tasks are bin-picking, box-close, door-lock, door-unlock and hand-insert. References are rollouts of the benchmark's scripted expert policies over 200 steps. A rollout is kept if the task succeeds at any step, and 150 are kept per task. The grasp flag is the expert's achieved grasp $\hat{g}_t$. Of the 150 references of a task, 15 are held out and the first ten of the remaining ones are used for training.

RoboTwin 2.0. The training tasks are place-cans-plasticbox, place-can-basket, place-container-plate, place-a2b-left and stack-bowls-two, and the unseen tasks are beat-block-hammer and stack-blocks-two. References are executions of each task's scripted planner program in RoboTwin 2.0 (Chen et al., 2026) under new scene seeds, kept if planning and the task's success check both succeed. Each records the end-effector position of both arms, a grasp flag per arm with the definition of the achieved grasp in Appendix A.3 (normalized opening below 0.8 and finger links in contact with a movable object), and the object keypoints. Only objects that move in the references carry keypoints,

so static objects such as the plastic box, the plate and the block that the hammer strikes are left out. We keep 50 references per task, resampled to $T = 300$ steps, the positions by linear interpolation and the grasp flags by nearest index. The same five indices are held out on every task, and the first five of the remaining ones are used for training.

Corrupted references. The corruption is drawn once per reference and per arm and is then fixed, with a random generator seeded per task. Each axis of the end-effector path receives a smooth warp $A \sin(2\pi f k/(T-1) + \varphi)$ with $A \sim \mathcal{U}[0, 2\,\mathrm{cm}]$, $f \sim \mathcal{U}[0.5, 1.5]$ cycles per reference and $\varphi \sim \mathcal{U}[0, 2\pi)$, plus independent Gaussian jitter with a standard deviation of 3 mm at every index. Object keypoint tracks, grasp flags and initial scene states are not changed. The corrupted references are used for training and for all three evaluation settings.

Evaluation. An episode is a success if the benchmark's success criterion holds at any step, and episodes are not terminated early. Episodes on training and held-out references draw a reference of the respective split at random at every reset. For the one-shot setting, 15 references of each unseen task are drawn once from all its references (150 on Meta-World, 50 on RoboTwin), and every seed is scored on the same 15, with one episode each.

A.5 RoboTwin 2.0 Tasks

The bimanual experiments of Section 4.3 use RoboTwin 2.0 (Chen et al., 2026) with its `aloha-agilex` robot, which has two arms with six joints and a gripper each, and the task configuration `demo_clean`, in which the table is not cluttered and the background, lighting and table height are not randomized. A single policy is trained on the five training tasks of Table 5, which place objects with one arm, with both arms at once or with both arms one after the other, and stack bowls. The two unseen tasks, tool use with a hammer and stacking blocks, are used neither for training nor for checkpoint selection and are only evaluated one-shot, with each episode conditioned on a single reference of the task (Appendix A.4). Figure 5 shows one frame of each task.

Table 5: **RoboTwin 2.0 tasks.** The regime is how the task's scripted planner uses the two arms. Where one arm acts, the scene decides whether it is the left or the right arm. In the stacking tasks, each object is moved by the arm on its side of the table, so one arm moves both objects or the two arms move one each in turn.

Task	Regime	Split	Description
place-cans-plasticbox	two arms, simultaneous	training	Both arms grasp and lift a can each and place it into a plastic box.
place-can-basket	two arms, sequential	training	One arm places a can into a basket, then the other arm lifts the basket.
place-container-plate	one arm	training	One arm places a container onto a plate.
place-a2b-left	one arm	training	One arm places object A to the left of object B.
stack-bowls-two	one or two arms, sequential	training	Stack one bowl into another.
beat-block-hammer	one arm	unseen (one-shot)	One arm grasps a hammer and strikes a block with it.
stack-blocks-two	one or two arms, sequential	unseen (one-shot)	Stack one block onto another.

A.6 Baselines

Reference replay. At step t, reference replay commands the end-effector displacement $\mathrm{clip}\big((h_t - p_t)/1\,\mathrm{cm},\, -1,\, 1\big)$, that is, it moves toward the current waypoint at up to the action bound of 1 cm per axis. It closes the gripper if the grasp flag is set at index t or $t + 4$ and opens it otherwise. The lookahead of four indices accounts for the grasp flag marking an established grasp, which appears several steps after the gripper starts to close. The offset between the controlled hand body and the

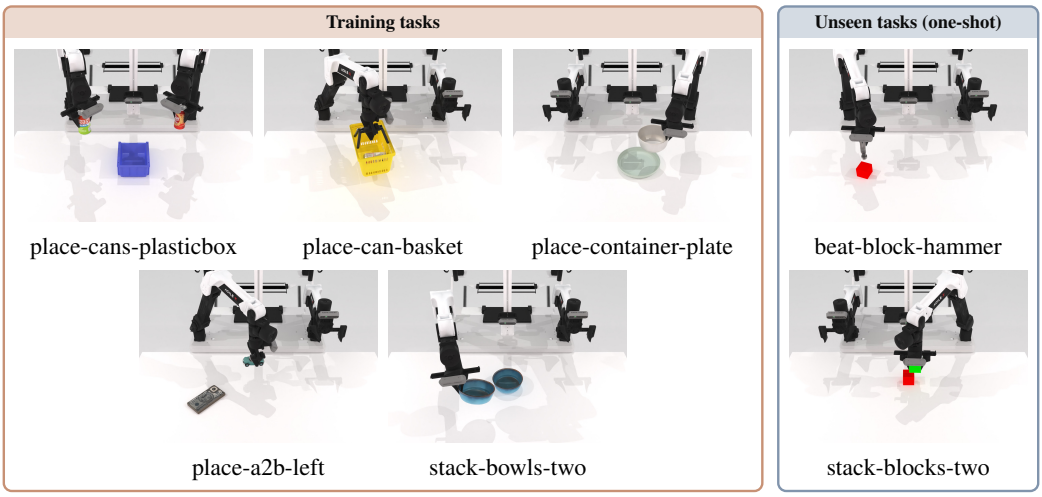


Figure 5: **RoboTwin 2.0 tasks.** Expert demonstrations of the five training tasks (left) and the two unseen tasks (right), not rollouts of TRACK. For each task, the scene of its first training reference is rebuilt from that reference's seed, the task's scripted planner program is run again in it, and the frames are rendered with RoboTwin's path tracer from a fixed camera facing the robot across the table, without overlays and with the empty foreground cropped. Each image shows the frame in which an object is lifted highest, except for place-can-basket, which shows the final frame, in which the second arm lifts the basket with the can inside. As the camera faces the robot, its left arm is on the right of each image.

tool center is read once from the first observation of an episode. Replay reads nothing else from the environment and is deterministic.

Diffusion Policy. We use the one-dimensional convolutional U-Net of Chi et al. (2023) with channel widths 256, 512 and 1024, kernel size 5 and FiLM conditioning. It sees two observation steps, predicts sixteen steps and executes eight before predicting again. Training uses 100 diffusion steps with a squared-cosine noise schedule and inference 16 DDIM steps. We train for 10^6 gradient steps with AdamW (learning rate 10^{-4}, weight decay 10^{-6}, 500 warm-up steps and a cosine schedule), batch size 256 and an exponential moving average of the weights. Every 10^5 steps the averaged weights are evaluated on the training references for ten episodes per training task, and the best evaluation is kept. We train on the ten corrupted training references per task. For the one-shot fine-tuning, we start from these weights and train for 3000 gradient steps (learning rate 10^{-4}, 100 warm-up steps, batch size 256) on the first training reference of the unseen task, with half of every batch drawn from the original training data, and score the final weights. The fine-tuning reference is not among the 15 one-shot references.

Im2Flow2Act. We reuse the flow-conditioned diffusion policy of the public Im2Flow2Act implementation (Xu et al., 2024), i.e. its flow encoder, its transformers and its U-Net, and replace the data pipeline, training loop and evaluation to run on Meta-World. The flow is not generated from video. It consists of the ground-truth positions of 128 simulator points projected into a 224 by 224 image of the corner camera, half on the object and half on the robot, preferring points that move, whereas the original uses points on the object only. Each point also carries its initial 3D position, read from the simulator, where the original computes it from a depth image. The plan spans 32 frames, the action is the 4-dimensional Meta-World action, the proprioception is the tool-center position and the gripper opening, and the training targets are the scripted expert's actions. The transformers are 384-dimensional with eight heads, with depth 4 for the flow-state and plan encoders and depth 8 for the alignment transformer, and the U-Net has channel widths 256, 512 and 1024. The policy sees one observation step, predicts sixteen steps and executes eight. Training uses 50 diffusion steps with a squared-cosine noise schedule and inference 16 DDIM steps, AdamW (learning rate 10^{-4}, weight decay 0.01, 500 warm-up steps and a cosine schedule), batch size 16 and an exponential moving average of the weights, which is evaluated. On ML10 we train for 4×10^5 steps (24 epochs over 135

references per task) and select epoch 23 on every seed. With ten references per task we train for 320 epochs and select one of epochs 80, 160, 240 and 319 per seed. On ML45 we train for five epochs (379,690 steps) and use epoch 4 on every seed.

B Additional Results

B.1 Ablation: Lookahead

Table 6: **Lookahead ablation on ML10.** Success rate of TRACK (clock) with window lookahead $L = 0$, 5 and 10 at stride $\delta = 4$. Mean $\pm$ standard deviation over five seeds, 200-step episodes, trained and evaluated on corrupted references. One-shot is the mean over the five unseen tasks. Grasp and carry is the mean over the tasks that lift and carry an object while grasped: pick-place, peg-insert-side and basketball on their training references, and shelf-place one-shot.

L	Steps	Training	Held-out	One-shot	Grasp and carry
0	0	0.80 ± 0.12	0.76 ± 0.11	0.57 ± 0.06	0.48 ± 0.26
5 (used)	20	0.93 ± 0.04	0.88 ± 0.03	0.67 ± 0.10	0.73 ± 0.05
10	40	0.92 ± 0.04	0.79 ± 0.09	0.66 ± 0.12	0.74 ± 0.05

The reference window of Equation 2 contains the current goal and L future goals. Table 6 varies L on ML10, with $L = 5$ being the one used in the main results. Without lookahead ($L = 0$), the mean success rate is consistently lower. The difference is largest on the grasp-and-carry tasks. Doubling the lookahead to $L = 10$ does not improve on $L = 5$ on any split.

B.2 Ablation: End-Effector Reference

Table 7: **TRACK without the end-effector reference.** Success rate of policies that do not observe or get rewards for the reference end-effector path. Mean and sample standard deviation over five seeds. †Uses a 500-step horizon. All other cells use 200 steps.

	ML10			ML45		
Object only	Training	Held-out	One-shot	Training	Held-out	One-shot
Clock	0.36 ± 0.12	0.33 ± 0.13	0.16 ± 0.18	0.42 ± 0.08	0.38 ± 0.07	0.18 ± 0.04
Tracker	0.31 ± 0.05	0.27 ± 0.05	$0.18 \pm 0.12^{\dagger}$	0.40 ± 0.06	0.34 ± 0.07	$0.21 \pm 0.10^{\dagger}$

Table 7 removes the end-effector reference from TRACK, from both the conditioning and the reward, so that the policy follows only the object keypoint tracks and the grasp flags. Compared with the TRACK rows of Table 1, success rates consistently decrease by about half or more, both under the clock and under the tracker. We specifically find that unseen tasks that involve grasping fail almost entirely. The (corrupted) end-effector path is therefore an important signal for the policy, which the object keypoint tracks cannot replace.

B.3 Im2Flow2Act with Ten References

Trained on only the ten references per task that TRACK uses instead of 135, Im2Flow2Act reaches 0.75 ± 0.01 on the training references of ML10, 0.62 ± 0.01 on held-out references and 0.10 ± 0.02 one-shot, as mean and sample standard deviation over five seeds (training details in Appendix A.6). With 135 references it reaches 0.99 ± 0.00, 0.96 ± 0.01 and 0.22 ± 0.07 (Table 1).

B.4 VLM Plans

Table 8 lists every task of Section 4.4 on one-shot plans. Replay of a plan scores zero or close to zero on most tasks that lift and carry an object and on the push tasks, and the policies score on the former but not on the latter.

B.5 Prompts of the Vision-Language Model

Every plan of Section 4.4 is the answer to one request to Qwen3.8-27B (Qwen Team, 2026), loaded with 4-bit weights, with thinking disabled, the checkpoint's default sampling settings and at most 2048 new tokens. The request holds one image and one text. The image is a 480 by 480 render of the scene from the benchmark's corner camera, with a labeled marker on every listed item and the frame axes drawn at the manipulated object (Figure 6, left). The text is the template below with the task line, the motion sequence, the item names and the starting positions of that task and scene filled in; the task lines and motion sequences of all fifty tasks are listed at the end of this section. The answer is parsed as JSON. A plan is rejected, and the request repeated at most twice, if it has fewer than two distinct waypoints, travels less than 2 cm in total, or has to be clipped by more than 15 cm to stay inside the workspace.

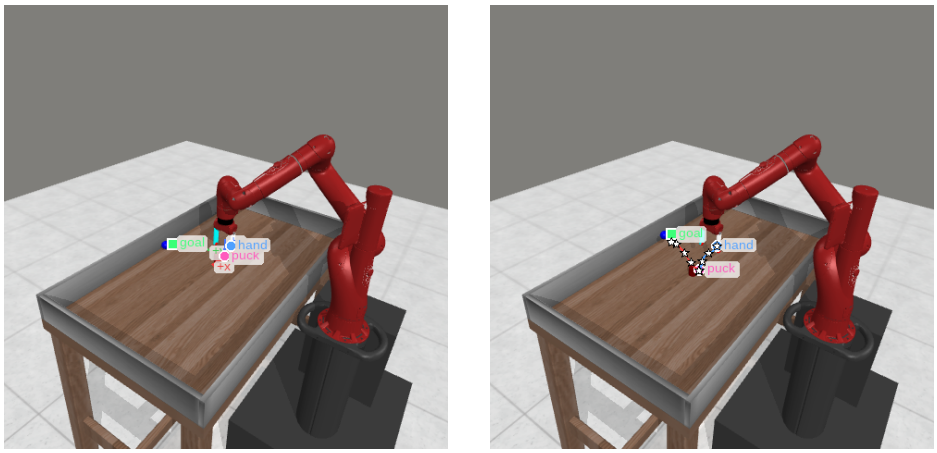


Figure 6: **The images in the prompt.** Left: the scene to plan for, here the first planned scene of pick-place, with a labeled marker on the puck, the goal and the hand and the frame axes at the puck. Right: the worked example of the one-shot prompt, another scene of the same task with the expert's end-effector path drawn, blue while the gripper is open and red while it holds, and a star at every keyframe listed in the prompt.

Zero-shot prompt. The text sent for the scene on the left of Figure 6, verbatim.

```
You are planning the motion of a robot hand to pick up the puck and
place it at the goal position.

MOTION SEQUENCE TO FOLLOW:
Move above the puck, descend onto it and close the gripper. Lift the
puck
clear of the table, carry it to the goal marker and release it there.

COORDINATE FRAME (metres):
- x: sideways across the table; +x is to the robot's right.
- y: forward, away from the robot's base (the robot stands near y = 0,
the far edge of the table is near y = 1.0).
- z: up; the table top is at z = 0. The hand cannot go below z = 0.05
or above z = 0.5.
All positions below are relative to the puck, which sits at [0.000,
0.000, 0.000].
The image shows the scene from a fixed camera; each listed item is
marked with a labelled dot, and the coloured arrows at the origin show
the +x (red), +y (green) and +z (blue) directions.

STARTING POSITIONS:
- puck: [0.000, 0.000, 0.000] (the origin)
- goal: [0.169, 0.197, 0.080] (target location)
```

```
- hand: [0.076, -0.079, 0.130] (the robot hand's current position; the
trajectory starts here)

TASK: output EXACTLY 10 waypoints of the hand (the point between the
fingertips) that carry out the motion sequence, in order.
- Waypoint 1 is the hand's current position. Every later waypoint moves
the hand by at least 0.02 m from the previous one; a plan that stays in
place is wrong.
- Each waypoint is a distinct phase of the motion: approach, descend,
grasp, lift, carry, place, release ... as the sequence requires.
- "grasp" is true at every waypoint where the fingers are closed on an
object, false otherwise. Close only once the hand is AT the object
(same position as the object). Tasks that push or press never close.
- Objects rest on the table: approach them from above, lift before
carrying, and end with the hand at the final position of the task.

OUTPUT FORMAT --- a JSON array of 10 entries, numbers only, no prose:
'''json
[{"hand": [x, y, z], "grasp": false},
 {"hand": [x, y, z], "grasp": false},
 ...]
'''
```

One-shot prompt. The one-shot variant sends the example scene's render before the scene to plan for, replaces the sentence that describes the image by the first line below, and inserts the remaining lines before the line that starts with TASK. The example is an expert demonstration of another scene of the same task, never the scene being planned, thinned to at most ten keyframes, namely the ends, the frames before and after every change of the grasp flag, and the rest uniform in arc length, and written in the example scene's own frame.

```
Images: first the example scene(s) with the expert's path drawn (blue
while the gripper is open, red while it holds), LAST the scene you must
plan for, with labelled dots on each listed item and coloured arrows at
the origin for +x (red), +y (green), +z (blue).

WORKED EXAMPLE --- a DIFFERENT scene of the same task, positions
relative to that scene's puck:
- puck: [0.000, 0.000, 0.000]
- goal: [0.026, 0.155, 0.153]
- hand: [-0.051, -0.054, 0.130]
A correct plan for that scene (9 waypoints):
'''json
[{"hand": [-0.051, -0.054, 0.130], "grasp": false},
 {"hand": [-0.031, -0.025, 0.088], "grasp": false},
 {"hand": [-0.009, -0.004, 0.046], "grasp": false},
 {"hand": [-0.001, 0.004, -0.008], "grasp": false},
 {"hand": [-0.000, 0.006, -0.007], "grasp": true},
 {"hand": [0.011, 0.045, 0.028], "grasp": true},
 {"hand": [0.014, 0.084, 0.070], "grasp": true},
 {"hand": [0.021, 0.128, 0.105], "grasp": true},
 {"hand": [0.026, 0.155, 0.108], "grasp": true}]
'''
```

The answer. The model's answer to the zero-shot prompt above. The waypoints are expanded into the reference of $T = 200$ steps described in Section 4.4, and the policy then executes it.

```
'''json
[
 {"hand": [0.076, -0.079, 0.13], "grasp": false},
 {"hand": [0.036, -0.04, 0.22], "grasp": false},
 {"hand": [0.0, 0.0, 0.28], "grasp": false},
 {"hand": [0.0, 0.0, 0.11], "grasp": false},
 {"hand": [0.0, 0.0, 0.035], "grasp": true},
```

```
 {"hand": [0.0, 0.0, 0.14], "grasp": true},
 {"hand": [0.085, 0.099, 0.26], "grasp": true},
 {"hand": [0.169, 0.197, 0.32], "grasp": true},
 {"hand": [0.169, 0.197, 0.13], "grasp": true},
 {"hand": [0.169, 0.197, 0.09], "grasp": false}
]
```

Task lines and motion sequences. The first line of the prompt and its motion sequence for every task, with only the line breaks inside a motion sequence removed. The item names in the starting positions follow the benchmark's objects, and the goal marker is listed for every task that has one.

```
assembly: pick up a wrench and put its ring over a peg
    Move above the wrench handle, descend and close the gripper. Lift
  the wrench, carry it so its ring is above the peg at the goal marker,
  and lower it onto the peg.

basketball: pick up the basketball and drop it into the hoop
    Move above the ball, descend and close the gripper on it. Lift the
  ball high, carry it above the hoop at the goal marker and release it.

bin-picking: pick the object out of one bin and place it in the other
    Descend into the bin onto the object and close the gripper. Lift it
  clear of the bin walls, carry it over to the goal marker and release
  it.

box-close: put a lid on a box
    Move above the lid's handle, descend and close the gripper. Lift
  the lid, carry it above the box at the goal marker and lower it onto
  the box.

button-press-topdown: press a button from above
    Move directly above the button, then descend and push it down until
  it reaches the goal marker (fully pressed). The gripper stays open.

button-press-topdown-wall: press a button from above, reaching past a
  wall
    A wall stands in the way. Go over it, move directly above the
  button, then descend and push it down until it reaches the goal
  marker. The gripper stays open.

button-press: press a button that faces the robot
    Move in front of the button at its height. Push the button
  horizontally until it reaches the goal marker (fully pressed). The
  gripper stays open.

button-press-wall: press a button that faces the robot, reaching past a
  wall
    A wall stands in the way. Go around it, move in front of the button
  at its height and push it until it reaches the goal marker. The
  gripper stays open.

coffee-button: press the button on a coffee machine
    Move in front of the coffee machine's button at its height and push
  it until it reaches the goal marker. The gripper stays open.

coffee-pull: pull a mug away from the coffee machine
    Move to the mug and close the gripper on it. Drag it along the
  table until it sits on the goal marker, then release.

coffee-push: push a mug to the coffee machine
    Lower the hand behind the mug (opposite the goal). Push the mug
  along the table until it sits on the goal marker. The gripper stays
  open.
```

```
dial-turn: turn a dial
    Move to the dial's edge. Push it along an arc about the dial's
  centre until it reaches the goal marker. The gripper stays open.

disassemble: lift a wrench off a peg
    Move to the wrench resting on the peg and close the gripper on its
  handle. Lift straight up until the ring clears the peg, ending at the
  goal marker.

door-close: close an open door by pushing its handle
    Move to the door handle. Push it along an arc (the door swings
  about its hinge) until it reaches the goal marker. Grasping is
  optional.

door-lock: lock a door by turning its lock
    Move to the lock and press against it. Move the lock to the goal
  marker (the lock rotates about its axis).

door-open: open a door by pulling its handle
    Move to the door handle and close the gripper on it. Pull the
  handle along an arc (the door swings about its hinge) until it
  reaches the goal marker.

door-unlock: unlock a door by turning its lock
    Move to the lock and press against it. Move the lock to the goal
  marker (the lock rotates about its axis).

drawer-close: close an open drawer by pushing its handle
    Move to the drawer handle. Push the handle straight in until it
  reaches the goal marker. The gripper stays open.

drawer-open: open a drawer by pulling its handle
    Move to the drawer handle and close the gripper on it. Pull the
  handle straight out until it reaches the goal marker.

faucet-close: close a faucet by rotating its handle
    Move to the faucet handle. Push it along an arc about the faucet's
  axis until it reaches the goal marker. The gripper stays open.

faucet-open: open a faucet by rotating its handle
    Move to the faucet handle. Push it along an arc about the faucet's
  axis until it reaches the goal marker. The gripper stays open.

hammer: pick up a hammer and drive a nail with it
    Move above the hammer handle, descend and close the gripper. Lift
  the hammer, bring its head to the nail and push the nail in until the
  hammer reaches the goal marker.

hand-insert: push a puck into a hole
    Lower the hand behind the puck (opposite the goal). Push the puck
  along the table until it drops into the hole at the goal marker. The
  gripper stays open.

handle-press-side: press a handle down, approaching from the side
    Move above the handle from the side, then push it down until it
  reaches the goal marker. The gripper stays open.

handle-press: press a handle down
    Move directly above the handle, then push it down until it reaches
  the goal marker. The gripper stays open.

handle-pull-side: pull a handle up, approaching from the side
    Move to the handle from the side and close the gripper on it. Pull
  it up until it reaches the goal marker.
```

```
handle-pull: pull a handle up
    Move to the handle and close the gripper on it. Pull it up until it
  reaches the goal marker.

lever-pull: pull a lever up
    Move under the lever's end. Push it upward along its arc until it
  reaches the goal marker. The gripper stays open.

peg-insert-side: pick up a peg and insert it sideways into a hole
    Move above the peg, descend and close the gripper. Lift the peg,
  align it with the hole at the goal marker, and slide it in sideways
  until it sits on the goal marker.

peg-unplug-side: pull a peg sideways out of a hole
    Move to the exposed end of the peg and close the gripper on it.
  Pull the peg straight out of the hole until it reaches the goal
  marker.

pick-out-of-hole: pick the puck out of a hole and place it at the goal
  position
    Descend into the hole onto the puck and close the gripper. Lift the
  puck well clear of the hole, carry it to the goal marker and release
  it.

pick-place: pick up the puck and place it at the goal position
    Move above the puck, descend onto it and close the gripper. Lift
  the puck clear of the table, carry it to the goal marker and release
  it there.

pick-place-wall: pick up the puck and place it at the goal position,
  over a wall
    Move above the puck, descend onto it and close the gripper. Lift
  high enough to clear the wall between the puck and the goal, carry
  the puck to the goal marker and release it there.

plate-slide-back-side: slide a puck sideways back out of a goal
    Lower the hand beside the puck (opposite the goal). Push the puck
  along the surface until it reaches the goal marker. The gripper stays
  open.

plate-slide-back: slide a puck back out of a goal
    Lower the hand behind the puck (opposite the goal). Push the puck
  along the surface until it reaches the goal marker. The gripper stays
  open.

plate-slide-side: slide a puck sideways into a goal
    Lower the hand beside the puck (opposite the goal). Push the puck
  along the surface until it reaches the goal marker. The gripper stays
  open.

plate-slide: slide a puck into a goal
    Lower the hand behind the puck (opposite the goal). Push the puck
  along the surface until it reaches the goal marker. The gripper stays
  open.

push-back: push the puck back to the goal position
    Lower the hand to table height on the side of the puck opposite the
  goal. Push the puck along the table until it sits on the goal marker.
  The gripper stays open; the puck is never lifted.

push: push the puck to the goal position
    Lower the hand to table height on the side of the puck opposite the
  goal. Push the puck along the table until it sits on the goal marker.
  The gripper stays open; the puck is never lifted.
```

```
push-wall: push the puck to the goal position around a wall
    A wall lies between the puck and the goal. Lower the hand behind
  the puck and push it along a path that goes around the wall until it
  sits on the goal marker. The gripper stays open.

reach: move the robot hand to the goal position
    Move the hand directly to the goal marker and stop there. Nothing
  is grasped.

reach-wall: move the robot hand to the goal position, going around a
  wall
    A wall stands between the hand and the goal. Go up and over or
  around it, then move to the goal marker and stop there. Nothing is
  grasped.

shelf-place: pick up the object and place it on the shelf
    Move above the object, descend and close the gripper. Lift to shelf
  height, carry the object to the goal marker on the shelf and release
  it.

soccer: push a soccer ball into the goal
    Lower the hand behind the ball (opposite the goal). Push the ball
  along the table until it sits on the goal marker. The gripper stays
  open.

stick-pull: pick up a stick and use it to pull an object to the goal
    Move above the stick, descend and close the gripper on it. Lift it
  slightly, hook its tip behind the object and pull the object along
  the table until it reaches the goal marker.

stick-push: pick up a stick and use it to push an object to the goal
    Move above the stick, descend and close the gripper on it. Lift it
  slightly, bring its tip against the object and push the object along
  the table until it reaches the goal marker.

sweep-into: sweep a puck into a hole
    Lower the hand beside the puck (opposite the goal). Sweep the puck
  along the table until it drops into the hole at the goal marker. The
  gripper stays open.

sweep: sweep a puck off the table to the goal
    Lower the hand beside the puck (opposite the goal). Sweep the puck
  along the table until it reaches the goal marker. The gripper stays
  open.

window-close: close a sliding window by its handle
    Move to the window handle. Push the handle sideways along the
  window rail until it reaches the goal marker. Grasping is optional.

window-open: open a sliding window by its handle
    Move to the window handle. Push the handle sideways along the
  window rail until it reaches the goal marker. Grasping is optional.
```

Table 8: **Success on one-shot plans, per task.** Success-once of reference replay and of the ML10 and ML45 policies of Table 3 on the one-shot plans of every task, one trial per plan (fifteen per task, ten on nine tasks) and a mean over five seeds for the policies. The second column lists the splits whose training tasks include the task, so the ML10 policies never trained on a task marked ML45 and no policy trained on a task marked –. Within the ML10 and within the ML45 policies the highest value of a row is shaded when it is not zero. †Scored with a 500-step horizon, and the other columns use 200 steps.

		ML10 policies			ML45 policies		
Task	Training task in	Replay	Clock	Tracker†	Replay	Clock	Tracker†
assembly	ML45	0.00	0.00	0.00	0.00	0.00	0.00
basketball	ML10, ML45	0.00	0.37	0.57	0.00	0.36	0.69
bin-picking	–	0.07	0.35	0.69	0.07	0.21	0.17
box-close	–	0.13	0.01	0.09	0.13	0.01	0.15
button-press-topdown	ML10, ML45	1.00	1.00	0.98	1.00	1.00	1.00
button-press-topdown-wall	ML45	1.00	0.97	0.95	1.00	0.97	1.00
button-press	ML45	0.47	0.15	0.27	0.47	0.28	0.48
button-press-wall	ML45	0.73	0.81	0.72	0.73	0.45	0.99
coffee-button	ML45	1.00	0.77	1.00	1.00	1.00	1.00
coffee-pull	ML45	0.73	0.52	0.69	0.73	0.80	0.91
coffee-push	ML45	0.33	0.32	0.31	0.33	0.40	0.40
dial-turn	ML45	0.20	0.28	0.48	0.20	0.15	0.39
disassemble	ML45	0.00	0.08	0.01	0.00	0.20	0.27
door-close	ML45	1.00	0.63	0.79	1.00	0.76	0.67
door-lock	–	0.20	0.61	0.63	0.20	0.55	0.59
door-open	ML10, ML45	0.10	0.32	0.24	0.10	0.42	0.26
door-unlock	–	0.00	0.11	0.28	0.00	0.09	0.19
drawer-close	ML10, ML45	1.00	0.84	0.90	1.00	0.94	0.94
drawer-open	ML45	0.80	0.87	0.93	0.80	0.97	0.92
faucet-close	ML45	0.47	0.48	0.41	0.47	0.33	0.39
faucet-open	ML45	0.53	0.49	0.39	0.53	0.43	0.44
hammer	ML45	0.67	0.23	0.43	0.67	0.56	0.72
hand-insert	–	0.00	0.05	0.09	0.00	0.24	0.16
handle-press-side	ML45	1.00	1.00	0.92	1.00	0.91	0.97
handle-press	ML45	1.00	0.89	0.95	1.00	0.91	0.97
handle-pull-side	ML45	0.00	0.00	0.00	0.00	0.00	0.16
handle-pull	ML45	0.00	0.00	0.00	0.00	0.00	0.00
lever-pull	ML45	0.30	0.30	0.24	0.30	0.38	0.18
peg-insert-side	ML10, ML45	0.00	0.07	0.40	0.00	0.19	0.32
peg-unplug-side	ML45	0.53	0.25	0.36	0.53	0.28	0.23
pick-out-of-hole	ML45	0.00	0.00	0.40	0.00	0.24	0.16
pick-place	ML10, ML45	0.00	0.40	0.94	0.00	0.74	0.88
pick-place-wall	ML45	0.00	0.08	0.65	0.00	0.71	0.69
plate-slide-back-side	ML45	0.13	0.19	0.31	0.13	0.40	0.40
plate-slide-back	ML45	0.00	0.39	0.59	0.00	0.61	0.35
plate-slide-side	ML45	0.00	0.09	0.15	0.00	0.23	0.29
plate-slide	ML45	0.00	0.06	0.04	0.00	0.08	0.12
push-back	ML45	0.00	0.09	0.09	0.00	0.11	0.01
push	ML10, ML45	0.00	0.04	0.10	0.00	0.00	0.08
push-wall	ML45	0.07	0.03	0.07	0.07	0.05	0.08
reach	ML10, ML45	0.90	0.90	0.90	0.90	0.90	0.90
reach-wall	ML45	1.00	1.00	0.98	1.00	1.00	1.00
shelf-place	ML45	0.00	0.31	0.56	0.00	0.33	0.69
soccer	ML45	0.13	0.17	0.29	0.13	0.20	0.25
stick-pull	ML45	0.00	0.00	0.00	0.00	0.21	0.08
stick-push	ML45	0.00	0.16	0.09	0.00	0.81	0.83
sweep-into	ML45	0.47	0.17	0.09	0.47	0.08	0.32
sweep	ML10, ML45	0.00	0.03	0.05	0.00	0.07	0.04
window-close	ML45	1.00	0.95	0.93	1.00	1.00	0.99
window-open	ML10, ML45	1.00	1.00	0.96	1.00	1.00	0.96