Towards Human-level Dexterous Teleoperation

Puhao Li[1,2,*], **Zeyuan Chen**[2,3,*], **Yingying Wu**[1,2,*], **Pengkun Wei**[2], **Yuyang Li**[2,3], **Tianyu Wang**[2,3], **Jiaxiao Shi**[2], **Mingrui Yu**[1], **Baoxiong Jia**[2], **Song-Chun Zhu**[1,2,3], **Tengyu Liu**[2,†], **Siyuan Huang**[2,†]

[1]Tsinghua University [2]State Key Lab of General Artificial Intelligence, BIGAI
[3]Peking University [*]Equal Contribution [†]Corresponding author

https://bigai-dex.github.io/blog/teledexter

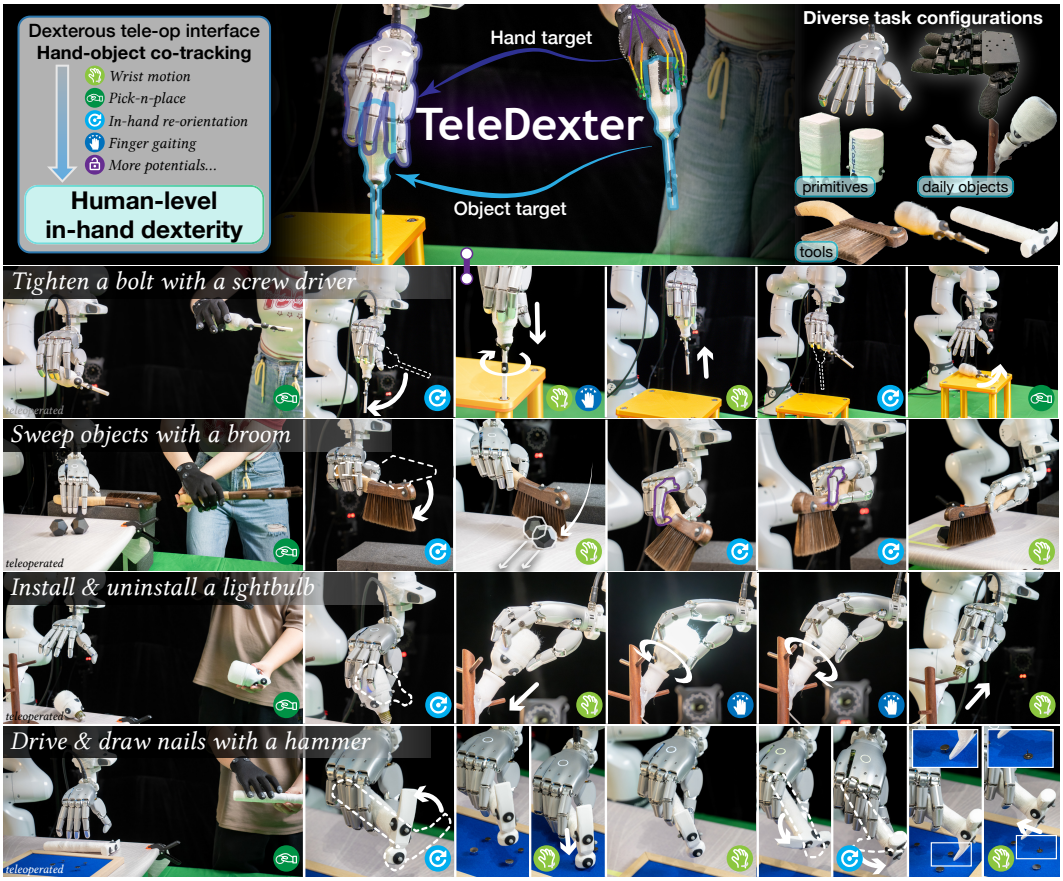


Fig. 1: **TELEDEXTER** learns diverse dexterous in-hand manipulation skills within a single-stage framework, marking a concrete step towards **human-level dexterous teleoperation**.

Abstract: Humans routinely wield tools, swap grasps, and reposition objects within a single hand—seamlessly orchestrating contact transitions that span translation, re-orientation, and finger gaiting. Endowing robot dexterous hands with this level of in-hand dexterity through teleoperation requires precise control of object motion via dynamic hand–object contact, yet current teleoperation systems remain far from this capability. To bridge this gap, we take a major step towards human-level dexterous teleoperation by introducing **TELEDEXTER**, a hand-object co-tracking controller that maps operator intent into learned, low-level contact execution. The controller is trained on consecutive co-tracking subgoals derived from human reference motions, utilizing a hybrid reward that couples sparse subgoal objectives with dense tracking rewards to enable learning across diverse interaction modalities rather than frame-wise trajectory imitation. The entire pipeline requires only single-stage RL and, with random action mask-

ing and domain randomization, transfers zero-shot to the real robot. We evaluate **TELEDEXTER** on seven challenging dexterous teleoperation tasks spanning object reorientation and long-horizon tool use across two dexterous hands, achieving a 75% average success rate where all baselines consistently fail. Furthermore, the collected demonstrations successfully train autonomous policies via behavioral cloning, marking a concrete step towards human-level dexterous teleoperation.

Keywords: Robot Manipulation, Dexterous Teleoperation, Sim-to-Real Transfer

1 Introduction

Everyday manipulation demands human-level dexterity: the ability to dynamically reorient, translate, and regrasp objects within a single hand by continuously coordinating complex finger–object contacts [1, 2]. While such agility is effortless for humans, it remains far beyond the reach of current robots. Teleoperation offers a powerful paradigm for closing this gap by enabling operators to directly teach the robot in the loop [3–5]. *However, achieving human-level in-hand dexterity through dexterous teleoperation remains an open challenge.*

Despite recent progress, existing dexterous teleoperation systems still fall short of human-level in-hand dexterity. One dominant line of work employs kinematic retargeting to map human hand motion directly onto robotic topologies, leveraging vision-based tracking [5–8] or wearable exoskeleton gloves [9–12]. While this paradigm provides an intuitive, low-latency interface that faithfully captures the operator's kinematic intent, it completely ignores hand-object contact forces and object inertia. Consequently, high-acceleration maneuvers, non-prehensile interactions, and continuous finger-gaiting remain highly unstable, frequently resulting in object slippage or drop failures.

An alternative paradigm [13] learns a dexterous action prior in simulation to map coarse teleoperation commands to contact-rich hand actions, improving local contact robustness over pure kinematics. However, these methods typically rely on synthetically generated grasp transitions as training targets, which often lack physical feasibility. Furthermore, encoding the action prior into a generative model introduces cascading trajectory drift and covariate shift during closed-loop execution, severely degrading real-world performance during long-horizon deployment.

To overcome these limitations, we introduce **TELEDEXTER**, *a hand–object co-tracking controller designed for human-level dexterous teleoperation*. Instead of mapping hand joints in isolation, the operator specifies explicit, synchronized geometric targets for both the fingertip positions and the object pose. A low-level controller, trained entirely in simulation with Reinforcement Learning (RL), then handles the complex multi-contact physics necessary to physically realize these dual co-tracking goals in real time. The key technical designs for **TELEDEXTER** are three-fold:

- **Consecutive subgoal co-tracking:** Rather than forcing rigid, frame-by-frame trajectory imitation, we decompose human reference motions into a sequence of synchronized fingertip and object pose subgoals. By training the policy to reach these consecutive targets rather than blindly copying exact motion configurations, the system gains the operational flexibility needed to discover physically feasible contact-switching strategies. This framework is optimized via a single-stage RL pipeline that couples sparse subgoal rewards with dense tracking rewards, eliminating the need for complex, task-specific reward engineering.
- **Co-tracking sequence construction:** We introduce a geometry-aware retargeting pipeline that translates unscripted human hand-object demonstrations into physically grounded reference motions. Going beyond pure kinematic mapping, our approach utilizes a two-stage optimization that incorporates object meshes to enforce contact-surface attraction and penalize mesh interpenetration. This process yields synchronized sequences of fingertip targets and object poses that serve as geometrically feasible subgoals for the low-level controller.
- **Sim-to-real robustness:** To ensure robust zero-shot sim-to-real transfer, we introduce random action masking as a strong action-space regularizer. This technique prevents the policy from

overfitting to perfectly synchronized simulated actuation, which, when combined with systematic domain randomization, allows the controller to deploy directly onto real robots.

We evaluate **TELEDEXTER** on seven challenging dexterous teleoperation tasks across two distinct dexterous hand embodiments. These tasks range from object rearrangement requiring continuous in-hand reorientation to long-horizon tool use with a hammer, screwdriver, brush, and light bulb. **TELEDEXTER achieves an average success rate of 75% across these tasks, whereas baseline models consistently fail.** Furthermore, we demonstrate that the high-quality teleoperation demonstrations collected via **TELEDEXTER** can be directly leveraged to train fully autonomous policies via behavioral cloning, achieving closed-loop manipulation capabilities without a human in the loop during long-term and dexterous task execution. Ablation studies confirm that our consecutive subgoal tracking and reward design are essential for learning diverse in-hand manipulation modalities in a single stage, while random action masking successfully bridges the sim-to-real gap. Broadly, this work provides a scalable foundation for collecting rich, in-hand dexterous manipulation data, unlocking a viable path toward human-level robotic dexterity.

2 Related Work

Dexterous Teleoperation provides a powerful paradigm for transferring human dexterity to robotic hands [3, 14–17]. One dominant line of work is kinematic retargeting, which kinematically translates the human hand configuration to robot joint positions, via vision-based tracking [5–8, 18], wearable or exoskeleton gloves [9–11, 19], or learned neural mappings [12]. This paradigm provides an intuitive, low-latency interface but lacks a dynamics prior, making contact-rich actions such as in-hand reorientation, finger gaiting, and tool use infeasible. To address this limitation, DexGen [13] learns a generative dexterous action prior from simulation rollouts that maps coarse teleoperation commands to fine hand actions, improving contact robustness. However, its reliance on synthetically generated grasp transitions as training goals does not guarantee physical feasibility, and encoding the action prior into a generative model introduces compounding errors that degrade real-world performance. In contrast, we directly train an RL controller guided from human hand–object reference motions, providing physically grounded goals that cover diverse in-hand manipulation modalities. The learned controller directly serves as the robust but agile low-level teleoperation policy, enabling human-level in-hand dexterity including long-horizon tool use.

Learning Dexterous Manipulation via RL in simulation has driven rapid progress, from grasping [20–23], in-hand reorientation [24–30] to complex finger gaiting and dynamic skills [31, 32]. However, these approaches typically learn task-specific policies with dedicated reward engineering for each skill. Recent work mitigates this by using human hand–object interaction data as reference motions to guide RL, enabling diverse manipulation skills without per-task reward design [23, 33–35]. However, these methods primarily learn one policy per trajectory and rarely achieve in-hand dexterous skills such as reorientation or finger gaiting. We attribute this to the dense frame-wise tracking formulation, which is overly restrictive for single-stage policy learning and prevents the RL agent from exploring dynamic contact strategies beyond basic grasping and wrist motion. To address this, we introduce a consecutive subgoal tracking formulation that learns from human hand–object reference motions, enabling diverse and dynamic in-hand skills within a single RL training stage. Combined with the proposed random action masking and systematic domain randomization [28, 36], the learned policy transfers zero-shot to the real robot as a dexterous teleoperation controller.

3 TELEDEXTER

We formulate dexterous teleoperation as hand–object co-tracking. The operator specifies hand and object pose targets, and our learned low-level controller executes the multi-contact dynamics to physically reach these goals. As shown in Fig. 2, we first formulate the co-tracking problem given a set of hand-object reference motions, then present the single-stage RL framework for training the

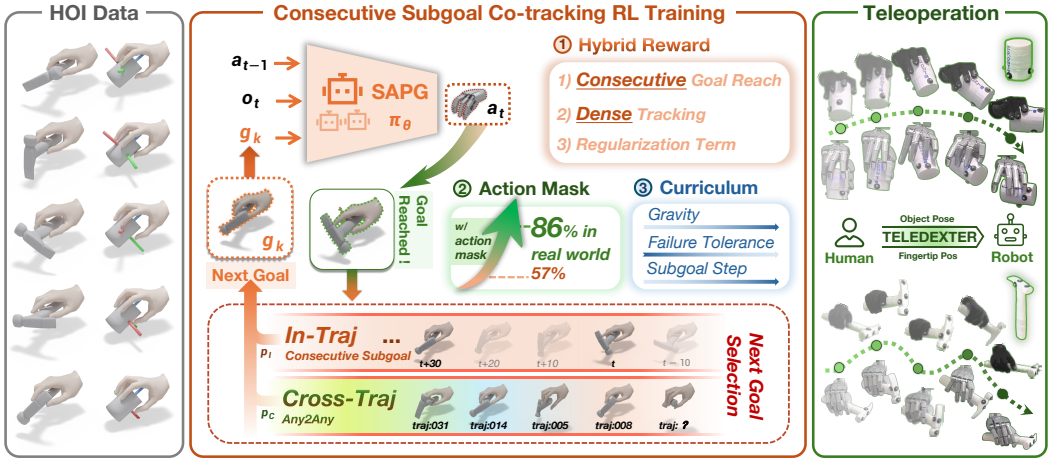


Fig. 2: **Method overview of TELEDEXTER.** Given human hand-object reference motions, we train a co-tracking controller via single-stage RL and deploy it zero-shot to real-world dexterous teleoperation.

co-tracking controller (Sec. 3.1). We then describe how these reference motions are constructed (Sec. 3.2) and deploy the learned policy to the real world as the teleoperation controller (Sec. 3.3).

Problem Formulation All quantities below are expressed in the wrist frame. The robot arm tracks the human wrist pose independently via inverse kinematics (IK). Given a set of hand-object reference motions (Sec. 3.2), our goal is to learn a co-tracking policy $\boldsymbol{a}_t = \pi_\theta(\boldsymbol{o}_t, g_t)$ that drives the robot hand and the manipulated object toward target poses prescribed by a co-tracking goal g_t. Here $\boldsymbol{o}_t$ encodes the current robot hand-object state, $\boldsymbol{a}_t \in \mathbb{R}^{n_{\text{dof}}}$ are target joint positions, and the co-tracking goal specifies both target fingertip positions and target object pose: $g_t = \left(\hat{\boldsymbol{p}}_t^{\text{tip}},\ \hat{T}_t^o\right)$, where $\hat{\boldsymbol{p}}_t^{\text{tip}} \in \mathbb{R}^{N_f \times 3}$ and $\hat{T}_t^o = \left(\hat{\boldsymbol{x}}_t^o,\ \hat{R}_t^o\right) \in SE(3)$. During teleoperation, g_t is constructed from real-time captured hand-object poses, casting dexterous teleoperation as a co-tracking problem. This formulation prescribes *what* the hand and object should achieve, while leaving the contact strategy, i.e. *how*, to the learned controller.

3.1 Learning a Co-tracking Controller

Given the reference motions for the manipulated object, we train a hand-object co-tracking controller π_θ in simulation via RL. The co-tracking goals guide the controller to learn a dynamic hand-object action prior through simulated contact, moving beyond kinematic trajectory imitation. The controller input consists of the observation $\boldsymbol{o}_t$ and the co-tracking goal g_t. The observation $\boldsymbol{o}_t$ contains the current hand joint positions $\boldsymbol{q}_t$, the object pose $(\boldsymbol{x}_t^o, R_t^o)$ and the gravity direction in the wrist frame, and the previous action $\boldsymbol{a}_{t-1}$.

Consecutive Subgoal Co-tracking Each reference trajectory is converted into a sequence of co-tracking subgoals sampled at varying intervals. We term this formulation *consecutive subgoal co-tracking*: the policy must reach each hand-object subgoal in order before advancing to the next, while freely discovering its own contact strategy between subgoals. For a subgoal $g_k = (\hat{\boldsymbol{p}}_k^{\text{tip}}, \hat{T}_k^o)$, the per-finger, object position, and object rotation tracking errors are

$$e_{t,k}^f = \left\| \boldsymbol{p}_{t,f}^{\text{tip}} - \hat{\boldsymbol{p}}_{k,f}^{\text{tip}} \right\|_2, \quad e_{t,k}^{\text{pos}} = \left\| \boldsymbol{x}_t^o - \hat{\boldsymbol{x}}_k^o \right\|_2, \quad e_{t,k}^{\text{rot}} = \left\| \text{Log}\left(\hat{R}_k^{o\top} R_t^o \right) \right\|_2. \tag{1}$$

A subgoal is reached when, for N_{stay} consecutive frames, $\max_f e_{t,k}^f < \epsilon_{\text{tip}}$, $e_{t,k}^{\text{pos}} < \epsilon_{\text{pos}}$, and $e_{t,k}^{\text{rot}} < \epsilon_{\text{rot}}$. Once this criterion is satisfied, the policy advances to the next subgoal.

Hybrid Reward Design The dominant learning signal is consecutive goal reaching, augmented with dense tracking reward for early exploration. The reward function combines a *consecutive goal-*

reaching reward with *dense tracking reward*:

$$r_t \;=\; \underbrace{\mathbb{1}_{\text{reach}}(t)\, w_{\text{step}}(t)\, r_{\text{score}}(t)}_{\text{sparse subgoal}} \;+\; \underbrace{\alpha_{\text{dense}}\, r_{\text{dense}}(t)}_{\text{dense tracking}} \;-\; \underbrace{c_{\text{time}}}_{\text{time}}, \tag{2}$$

The indicator $\mathbb{1}_{\text{reach}}(t)$ fires when the active subgoal is reached, and w_{step} weights the reward by the inter-subgoal step size. The score r_{score} measures how well the hand and object match the active subgoal:

$$r_{\text{score}} = \sum_{f=1}^{N_f} w_f \exp\Big(-\beta_f\, e_{t,k}^{f}\Big) + w_{\text{pos}} \exp\Big(-\beta_{\text{pos}}\, e_{t,k}^{\text{pos}}\Big) + w_{\text{rot}} \exp\big(-\beta_{\text{rot}}\, e_{t,k}^{\text{rot}}\big)\,. \tag{3}$$

The dense tracking reward r_{dense} uses the same per-finger and object tracking terms at every timestep, scaled by α_{dense}, providing a small dense signal during early training.

Curriculum Learning We progressively increase three dimensions of difficulty during training. (i) Gravity is reduced initially and annealed to full gravity, easing initial contact establishment. (ii) The subgoal tracking tolerances $(\epsilon_{\text{tip}}, \epsilon_{\text{pos}}, \epsilon_{\text{rot}})$ start permissive and are progressively tightened, enforcing stricter tracking precision over training. (iii) The inter-subgoal step size grows from small to large, so the policy first masters fine-grained local tracking and later handles longer-horizon goal jumps. Episodes are initialized at random frames across the reference motions, and successful traversal of one trajectory resets the environment to another (cross-trajectory reset), enabling continuous learning across the full reference-motion set.

Sim-to-Real Robustness For sim-to-real transfer, we apply domain randomization to tolerate dynamics, sensing, and actuation mismatch between simulation and the real world. Following [28], we randomize the shape and dynamics properties of both the object and the dexterous hand, apply random external forces to the object, and inject observation noise and latency. In addition, we introduce *random action masking* as a strong action-space regularization. Given the policy action $\boldsymbol{a}_t$, we sample a binary mask $\boldsymbol{m}_t \in \{0,1\}^{n_{\text{dof}}}$ and apply $\tilde{\boldsymbol{a}}_t = \boldsymbol{m}_t \odot \boldsymbol{a}_t + (1-\boldsymbol{m}_t) \odot \tilde{\boldsymbol{a}}_{t-1}$, where masked dimensions are frozen at the previous command for a randomly sampled duration. By forcing the policy to succeed even when subsets of joints retain stale commands, this regularization prevents the policy from overfitting to simulation dynamics, which inevitably differ from those of the real world.

Single-stage RL Training Each controller is trained in a single RL stage using large-scale parallel simulation, without staged skill decomposition or task-specific reward engineering. We use Isaac Gym [37], and all reference motions for one object ($\sim$50 minutes in our setting) are loaded simultaneously. We use SAPG [38] to optimize the policy with 4 NVIDIA RTX 5090 GPUs, running $\sim$62,000 parallel environments. Training converges within $\sim 10^{10}$ environment steps. Being reference-driven, the framework scales directly to new objects, hands and more interaction patterns.

3.2 Hand-Object Reference Motion Construction

We now describe how the hand-object reference motions used in training are constructed. Starting from human hand-object interaction trajectories recorded by a motion-capture system, we convert them into robot hand-object reference motions through geometry-aware retargeting. The recorded interactions span three categories: *(i) in-hand translation*, *(ii) in-hand rotation*, and *(iii) free-play* combining arbitrary grasps, finger gaiting, and tool-use motion sequences.

Geometry-aware Retargeting The retargeting proceeds in two stages. The first stage follows vector-based retargeting [5, 6], optimizing robot joint angles $\boldsymbol{q}_{1:T}$ to match human hand geometry via directional and inter-finger vector alignment. Kinematic retargeting alone does not account for object geometry or contact feasibility. The second stage refines the result with a geometry-aware optimization that incorporates the object mesh. The final trajectory is obtained by

$$\boldsymbol{q}^*_{1:T} = \arg\min_{\boldsymbol{q}_{1:T}} \sum_{t=1}^{T} \big(\mathcal{L}^t_{\text{vec}} + \lambda_{\text{surf}}\mathcal{L}^t_{\text{surf}} + \lambda_{\text{pen}}\mathcal{L}^t_{\text{pen}} + \lambda_{\text{col}}\mathcal{L}^t_{\text{col}}\big) + \lambda_{\text{smooth}}\mathcal{L}_{\text{smooth}}, \tag{4}$$

where $\mathcal{L}^t_{\text{vec}}$ is the vector-retargeting loss [5, 6]. Let $\mathcal{H}_t$ and $\mathcal{O}_t$ denote the hand and object meshes at frame t, with $\text{sdf}_{\mathcal{O}_t}(\cdot)$ the differentiable signed distance [39]. The surface term $\mathcal{L}^t_{\text{surf}} = \frac{1}{|\mathcal{S}_t|}\sum_{p\in\mathcal{S}_t}\text{ReLU}(\text{sdf}_{\mathcal{O}_t}(p))$ pulls near-contact hand points ($\mathcal{S}_t = \{p : \text{sdf}_{\mathcal{O}_t}(p) < \tau_{\text{surf}}\}$) onto the object surface. The penetration term $\mathcal{L}^t_{\text{pen}} = \sum_{p\in\mathcal{H}_t}\text{ReLU}\big(-\text{sdf}_{\mathcal{O}_t}(p)\big)$ penalizes interpenetration. The self-collision term $\mathcal{L}^t_{\text{col}} = \frac{1}{2}\sum_{f(i)\neq f(j)}\text{ReLU}\big((r_i + r_j) - \|\boldsymbol{c}_i - \boldsymbol{c}_j\|_2\big)$ prevents inter-finger overlap via collision spheres with radii r_i, r_j and centers $\boldsymbol{c}_i, \boldsymbol{c}_j$. $\mathcal{L}_{\text{smooth}}$ applies the Curobo [40] temporal smoothness energy on $\boldsymbol{q}_{1:T}$ to suppress capture jitter. The output is a set of robot hand-object reference trajectories $\{(\boldsymbol{q}^*_t, T^o_t)\}_{t=1}^T$ in the wrist frame, from which the co-tracking goals used in Sec. 3.1 are constructed as fingertip targets $\hat{\boldsymbol{p}}^{\text{tip}}_t = \text{FK}_{\text{tip}}(\boldsymbol{q}^*_t)$ and object targets $\hat{T}^o_t = T^o_t$.

3.3 Real-world Teleoperation Deployment

The learned co-tracking controller deploys zero-shot as the teleoperation controller. A real-time system captures the operator's wrist, fingertip, and object poses; the arm tracks the wrist via IK, while the fingertip and object poses form the co-tracking goal for the policy. For contact initialization, kinematic retargeting [6] handles pre-grasp positioning; once stable contact is established, the operator switches to the co-tracking controller for dexterous manipulation.

4 Experiments

We conduct a systematic real-world evaluation of **TELEDEXTER** for dexterous hand teleoperation. In experiments, we describe the experimental setup (Sec. 4.1), compare **TELEDEXTER** against representative baselines on seven dexterous tasks across two hand embodiments (Sec. 4.2), demonstrate autonomous policy learning from collected teleoperation data (Sec. 4.3), and conduct ablation studies on consecutive subgoal tracking and random action masking (Sec. 4.4).

4.1 Experimental Setup

Robot Platforms All real-world experiments are conducted on a Franka FR3 arm equipped with a dexterous robot hand. We evaluate two hand embodiments: LeapHand [41], a four-finger hand with 16 DoFs, and SharpaWave, a five-finger human-like hand with 22 DoFs, to validate that our framework can generalize across different robotic hand morphologies and actuation spaces.

Teleoperation Interface We use a NOKOV MoCap system to track the operator's hand pose and the manipulated object's 6D pose in real time. For all methods, the operator's wrist pose is converted into Franka arm commands via IK. The methods differ in how they map the captured hand and object references to dexterous hand actions. All teleoperation evaluations run at 30 Hz.

Baselines *DexRT* [5, 6] directly maps the operator's hand motion to the robot hand through kinematic retargeting. *GeoRT* [12] learns a neural retargeting function using geometric objectives. *DexGen* [13] uses a learned generative action prior to convert teleoperation commands into contact-rich hand actions. *SimToolReal* [42] is an object-centric sim-to-real tool manipulation method that requires human reference motions at inference time to guide execution. Although not a teleoperation approach, it provides a strong reference for learned dexterous tool use.

4.2 Dexterous Teleoperation Evaluation

Overview As shown in Fig. 3, we evaluate **TELEDEXTER** on seven real-world dexterous tasks spanning two categories. The three reorientation tasks (`CylinderReorient`, `CuboidReorient`, `BunnyReorient`) test precise in-hand pose control over symmetric, edge/corner, and irregular geometries. The four tool-use tasks (`HammerUse`, `BrushSweep`, `ScrewdriverUse`, `BulbReplace`) require long-horizon multi-stage manipulation involving in-hand reorientation to transition between functional grasps, finger gaiting, and tool application. Together, these tasks provide a comprehensive evaluation of whether a teleoperation system can robustly execute diverse dexterous manipulation.

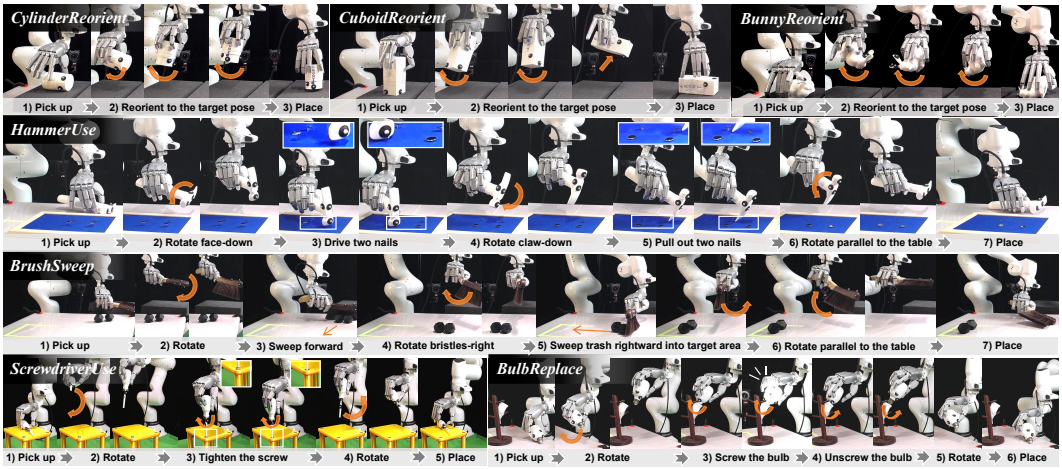


Fig. 3: **Task descriptions.** Seven dexterous tasks across two categories: three reorientation tasks over diverse geometries and four long-horizon tool-use tasks. Each task is decomposed into well-defined stages.

Tab. 1: **Dexterous teleoperation results on SharpaWave.** Each cell: **SR** / **TP** (%; higher is better). *SimToolReal* is not a teleoperation method († = category-specific, ‡ = all categories) averaged over three tasks only.

Task	*DexRT*	*GeoRT*	*DexGen*	*SimToolReal*†	*SimToolReal*‡	**TELEDEXTER**
`CylinderReorient`	6.7 / 37.8	0.0 / 24.4	0.0 / 31.1	—	—	**80.0 / 86.7**
`CuboidReorient`	26.7 / 51.1	0.0 / 33.3	0.0 / 28.9	—	—	**80.0 / 86.7**
`BunnyReorient`	0.0 / 35.6	0.0 / 31.1	0.0 / 26.7	—	—	**66.7 / 77.8**
`HammerUse`	0.0 / 26.7	0.0 / 30.5	0.0 / 26.7	0.0 / 27.6	20.0 / 36.2	**66.7 / 86.7**
`BrushSweep`	0.0 / 39.0	0.0 / 29.5	0.0 / 8.6	26.7 / 41.9	0.0 / 5.7	**73.3 / 89.5**
`ScrewdriverUse`	6.7 / 37.3	0.0 / 25.3	0.0 / 33.3	0.0 / 17.3	0.0 / 20.0	**73.3 / 86.7**
`BulbReplace`	0.0 / 35.6	0.0 / 25.6	0.0 / 20.0	—	—	**86.7 / 95.6**
Average	5.7 / 37.6	0.0 / 28.5	0.0 / 25.0	8.9 / 28.9	6.7 / 20.6	**75.2 / 87.1**

Tab. 2: **Teleoperation on LeapHand.**

Task	**TELEDEXTER**
`CylinderReorient`	60.0 / 73.3
`CuboidReorient`	73.3 / 82.2

Protocol and Metrics For each task, we conduct 15 trials per method. Each task is decomposed into a sequence of well-defined stages (shown in Fig. 3). We report two metrics. Success rate (**SR**) is the percentage of trials completing all stages. Task progress (**TP**) is the average percentage of stages completed per trial, where each stage contributes equally. A trial is terminated when an unrecoverable grasp loss or object drop occurs during execution.

Results and Analysis As shown in Tab. 1, **TELEDEXTER** achieves 75.2% average **SR** and 87.1% average **TP** across all seven tasks, while all baselines near-uniformly fail. On the reorientation tasks, **TELEDEXTER** achieves 66.7–80.0% **SR** by executing dynamic in-hand reorientation through learned contact strategies. In contrast, kinematic retargeting methods (*DexRT*, *GeoRT*) rarely progress beyond the initial pick-up stage, as they lack the dynamics prior needed for contact-rich in-hand manipulation. *DexGen*, despite its learned action prior, similarly fails due to compounding errors in its generative model that degrade real-world contact execution. The gap widens on tool-use tasks, which demand long-horizon coordination of functional grasp transitions, finger gaiting, and tool application. **TELEDEXTER** achieves 66.7–86.7% **SR** on these tasks, while all teleoperation baselines achieve near-zero **SR**. The **TP** metric reveals where failures occur: **TELEDEXTER**'s narrow **SR**-to-**TP** gap (75.2% vs. 87.1%) indicates that most failures happen at late task stages, whereas baselines consistently collapse at the first stage requiring in-hand reorientation or finger gaiting. On `BulbReplace`, **TELEDEXTER** completes both the screw-in and unscrew rotation stages without losing a single trial (15/15 through stage 4 of 6), with the only failures at final placement (13/15). Similarly, on `ScrewdriverUse`, 13 of 15 trials sustain continuous finger gaiting through the tightening stage, a contact mode that no baseline can execute. *SimToolReal*, an object-centric policy trained specifically for tool manipulation, achieves at most

Tab. 3: **Autonomous policy stage-wise success.** `BulbInstall`: pick up → reorient → align → install; `HammerDriver`: pick up → rotate → drive nails; `BrushForward`: pick up → rotate → sweep forward.

Task	Stage 1	Stage 2	Stage 3	Stage 4	**SR**
`BulbInstall`	13/15	12/13	8/12	7/8	46.7%
`HammerDriver`	15/15	15/15	11/15	—	73.3%
`BrushForward`	7/15	7/7	6/7	—	40.0%

26.7% **SR** on a single task and fails on the others, indicating that even dedicated tool-use policies struggle to generalize across the diverse contact transitions required for long-horizon manipulation. As shown in Tab. 2, applying the same training pipeline to LeapHand yields a strong controller with minimal embodiment-specific tuning. Notably, both controllers are trained from the same human hand-object interaction reference motions; only the geometry-aware retargeting stage adapts to the target morphology (4-finger, 16-DoF LeapHand vs. 5-finger, 22-DoF SharpaWave). Despite this substantial morphological gap, LeapHand achieves 60.0–73.3% **SR** on the reorientation tasks, confirming that the framework generalizes across embodiments without re-collecting human reference motions.

4.3 From Teleoperation to Autonomy

Overview A key advantage of **TELEDEXTER** is its ability to collect dexterous manipulation data beyond the reach of existing teleoperation systems. While the teleoperation evaluation (Sec. 4.2) demonstrates that **TELEDEXTER** enables human-level in-hand dexterity, the collected trajectories also serve as high-quality expert demonstrations for training autonomous policies. To validate this, we train Diffusion Policies [43] on three dexterous tasks: `BulbInstall`, `HammerDriver`, and `BrushForward`. Each task retains the most contact-intensive stages of its teleoperation counterpart while removing the return-and-place phases, isolating the core dexterous manipulation skills.

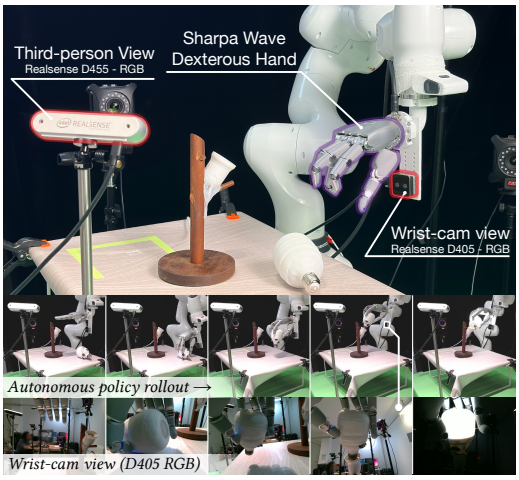


Fig. 4: **Autonomous policy setup and rollout.**

Protocol and Metrics We adopt the Conv-UNet Diffusion Policy architecture [43], conditioned on RGB observations from third-person and wrist-mounted cameras, as shown in Fig. 4. For each task, we collect 50 expert demonstrations and evaluate each policy over 15 real-world trials. Each task is decomposed into well-defined stages following the same protocol as the teleoperation evaluation; we report stage-wise success (the number of trials surviving past each stage) and overall **SR**.

Results and Analysis As shown in Tab. 3, all three tasks achieve non-trivial success rates from only 50 demonstrations, confirming that **TELEDEXTER** captures sufficiently rich contact-mode coverage for behavioral cloning across diverse dexterous manipulation skills. The stage-wise breakdown reveals a distinct bottleneck per task. `HammerDriver` achieves the highest **SR** (73.3%) with perfect grasping and reorientation (15/15 through stage 2). The only failures occur at the nail-driving stage, where the policy must sustain repeated contact force against the foam target. `BulbInstall` (46.7% **SR**) shows a similar pattern: grasping and reorientation succeed reliably, but the precision alignment stage (8/12) is the primary bottleneck. Once aligned, installation succeeds in 7/8 trials, indicating that the screw-in skill transfers well from demonstrations. `BrushForward` (40.0% **SR**) presents the opposite profile: grasping is the dominant failure point (7/15), due to the thin and irregular brush handle requiring precise finger placement that is difficult to resolve from RGB alone. Trials that survive the grasp nearly all complete the subsequent rotation and sweep (6/7).

Tab. 4: **Sparse subgoal vs. dense tracking** (sim). EpLen: episode length (↑). Goals: consecutive subgoals reach (↑).

	Dense eval: EpLen		Sparse eval: Goals	
Object	Ours	Dense	Ours	Dense
Cuboid	**378.6**	115.8	**32.6**	2.6
Hammer	**376.9**	131.2	**186.6**	2.7
Screwdriver	**373.2**	88.7	**178.5**	2.7

Tab. 5: **Random action masking** (real). Each cell reports **SR/ TP** (%).

Task	w/ AM	w/o AM
`HammerUse`	**66.7 / 86.7**	33.3 / 57.1
`ScrewdriverUse`	**73.3 / 86.7**	0.0 / 36.0
`CuboidReorient`	**80.0 / 86.7**	26.7 / 51.1

Notably, no baseline teleoperation system evaluated in Tab. 1 can reliably complete any of these three tasks, making it infeasible to collect comparable demonstration data with existing methods. These results validate **TELEDEXTER** as both a teleoperation interface and a scalable data collection pipeline for autonomous dexterous manipulation.

4.4 Ablation Studies

Consecutive Subgoal Tracking vs. Dense Tracking We compare our consecutive subgoal tracking formulation against standard dense frame-wise tracking in simulation on held-out reference motions for three objects (Tab. 4). We evaluate in two modes: dense, where the reference advances every control step, and sparse, where the target advances only after the current subgoal is reached. Even under dense evaluation, which favors the dense tracking baseline, sparse subgoal tracking achieves significantly longer episode lengths. Dense frame-wise tracking forces the policy to replicate reference trajectories step by step, leaving insufficient tolerance to discover physically feasible contact strategies and causing early termination. In contrast, sparse subgoal tracking only requires stable goal completion, allowing the policy to find feasible contact sequences through simulation rollout. This advantage is amplified in sparse evaluation, where dense tracking policies stall after only a few subgoals while sparse subgoal tracking reaches orders of magnitude more.

Random Action Masking We ablate random action masking on three real-world teleoperation tasks (Tab. 5). Removing action masking causes substantial degradation across all evaluated tasks. Random action masking serves as a strong action-space regularization that prevents the policy from overfitting to simulation dynamics, which inevitably differ from real-world dynamics. Without this regularization, the policy exploits simulation-specific dynamics patterns that do not transfer, confirming that random action masking is critical for zero-shot sim-to-real deployment.

5 Conclusion

We introduce **TELEDEXTER**, a hand-object co-tracking controller that takes a concrete step toward human-level dexterous teleoperation. Built on our proposed *consecutive subgoal tracking* with *hybrid reward design*, and *random action masking*, **TELEDEXTER** learns diverse in-hand skills in single-stage RL with no task-specific reward, and transfers to real robots. Across seven long-horizon and challenging dexterous tasks, **TELEDEXTER** achieves a strong success rate where baselines uniformly fail. Furthermore, the collected demonstrations successfully train policies to autonomously execute long-horizon dexterous tasks, establishing **a scalable foundation for collecting rich in-hand dexterous manipulation data and a viable path toward human-level robotic dexterity**.

6 Limitations

TELEDEXTER currently learns an object-specific controller. Adapting to a new object requires collecting human hand-object interaction data and training a dedicated policy. Scaling to a unified, object-conditioned controller that generalizes across object categories without per-object data collection and training is a promising direction for future work.

Our real-world deployment relies on a motion-capture system for real-time hand and object pose estimation. Replacing this with a markerless, vision-based tracking system would significantly lower the barrier to deployment and broaden the practical applicability of the framework.

References

[1] A. Billard and D. Kragic. Trends and challenges in robot manipulation. *Science*, 364(6446): eaat8414, 2019.

[2] I. M. Bullock, R. R. Ma, and A. M. Dollar. A hand-centric classification of human and robot dexterous manipulation. *IEEE Transactions on Haptics*, 6(2):129–144, 2013. doi:10.1109/TOH.2012.53.

[3] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 5628–5635. IEEE, 2018.

[4] A. Mandlekar, D. Xu, R. Martín-Martín, Y. Zhu, L. Fei-Fei, and S. Savarese. Human-in-the-loop imitation learning using remote teleoperation. *arXiv preprint arXiv:2012.06733*, 2020.

[5] Y. Qin, W. Yang, B. Huang, K. Van Wyk, H. Su, X. Wang, Y.-W. Chao, and D. Fox. Anyteleop: A general vision-based dexterous robot arm-hand teleoperation system. *arXiv preprint arXiv:2307.04577*, 2023.

[6] A. Handa, K. Van Wyk, W. Yang, J. Liang, Y.-W. Chao, Q. Wan, S. Birchfield, N. Ratliff, and D. Fox. Dexpilot: Vision-based teleoperation of dexterous robotic hand-arm system. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9164–9170. IEEE, 2020.

[7] R. Ding, Y. Qin, J. Zhu, C. Jia, S. Yang, R. Yang, X. Qi, and X. Wang. Bunny-visionpro: Real-time bimanual dexterous teleoperation for imitation learning. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12248–12255. IEEE, 2025.

[8] X. Cheng, J. Li, S. Yang, G. Yang, and X. Wang. Open-television: Teleoperation with immersive active visual feedback. In *8th Annual Conference on Robot Learning*, 2024.

[9] H. Zhang, S. Hu, Z. Yuan, and H. Xu. Doglove: Dexterous manipulation with a low-cost open-source haptic force feedback glove. *arXiv preprint arXiv:2502.07730*, 2025.

[10] H.-S. Fang, B. Romero, Y. Xie, A. Hu, B.-R. Huang, J. Alvarez, M. Kim, G. Margolis, K. Anbarasu, M. Tomizuka, et al. Dexop: A device for robotic transfer of dexterous human manipulation. *arXiv preprint arXiv:2509.04441*, 2025.

[11] A. Zhu, M. Zhu, B. J. Kim, J. V. S. Ramos, Y. Shi, Y. Wu, R. Dhar, F. Yang, R. Hou, H. Fang, et al. Dexexo: A wearability-first dexterous exoskeleton for operator-agnostic demonstration and learning. *arXiv preprint arXiv:2603.17323*, 2026.

[12] Z.-H. Yin, C. Wang, L. Pineda, K. Bodduluri, T. Wu, P. Abbeel, and M. Mukadam. Geometric retargeting: A principled, ultrafast neural hand retargeting algorithm. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 17376–17382. IEEE, 2025.

[13] Z.-H. Yin, C. Wang, L. Pineda, F. Hogan, K. Bodduluri, A. Sharma, P. Lancaster, I. Prasad, M. Kalakrishnan, J. Malik, et al. Dexteritygen: Foundation controller for unprecedented dexterity. *arXiv preprint arXiv:2502.04307*, 2025.

[14] G. Niemeyer, C. Preusche, S. Stramigioli, and D. Lee. Telerobotics. In *Springer handbook of robotics*, pages 1085–1108. Springer, 2016.

[15] H. Hedayati, M. Walker, and D. Szafir. Improving collocated robot teleoperation with augmented reality. In *Proceedings of the 2018 ACM/IEEE international conference on human-robot interaction*, pages 78–86, 2018.

[16] G. Du, P. Zhang, J. Mai, and Z. Li. Markerless kinect-based hand tracking for robot teleoperation. *International Journal of Advanced Robotic Systems*, 9(2):36, 2012.

[17] J. Kofman, S. Verma, and X. Wu. Robot-manipulator teleoperation by markerless vision-based hand-arm tracking. *International Journal of Optomechatronics*, 1(3):331–357, 2007.

[18] S. Yang, M. Liu, Y. Qin, R. Ding, J. Li, X. Cheng, R. Yang, S. Yi, and X. Wang. Ace: A cross-platform and visual-exoskeletons system for low-cost dexterous teleoperation. In *8th Annual Conference on Robot Learning*, 2024.

[19] C. Wang, H. Shi, W. Wang, R. Zhang, L. Fei-Fei, and C. K. Liu. Dexcap: Scalable and portable mocap data collection system for dexterous manipulation. *arXiv preprint arXiv:2403.07788*, 2024.

[20] P. Li, T. Liu, Y. Li, Y. Geng, Y. Zhu, Y. Yang, and S. Huang. Gendexgrasp: Generalizable dexterous grasping. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8068–8074. IEEE, 2023.

[21] Y. Xu, W. Wan, J. Zhang, H. Liu, Z. Shan, H. Shen, R. Wang, H. Geng, Y. Weng, J. Chen, et al. Unidexgrasp: Universal robotic dexterous grasping via learning diverse proposal generation and goal-conditioned policy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4737–4746, 2023.

[22] Y. Li, B. Liu, Y. Geng, P. Li, Y. Yang, Y. Zhu, T. Liu, and S. Huang. Grasp multiple objects with one hand. *IEEE Robotics and Automation Letters*, 9(5):4027–4034, 2024.

[23] K. Li, P. Li, T. Liu, Y. Li, and S. Huang. Maniptrans: Efficient dexterous bimanual manipulation transfer via residual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6991–7003, 2025.

[24] M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1), 2020. doi:10.1177/0278364919887447.

[25] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[26] A. Handa, A. Allshire, V. Makoviychuk, A. Petrenko, R. Singh, J. Liu, D. Makoviichuk, K. Van Wyk, A. Zhurkevich, B. Sundaralingam, et al. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5977–5984. IEEE, 2023.

[27] H. Qi, A. Kumar, R. Calandra, Y. Ma, and J. Malik. In-hand object rotation via rapid motor adaptation. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 1722–1732. PMLR, 2023. URL https://proceedings.mlr.press/v205/qi23a.html.

[28] T. Chen, M. Tippur, S. Wu, V. Kumar, E. Adelson, and P. Agrawal. Visual dexterity: In-hand reorientation of novel and complex object shapes. *Science Robotics*, 8(84):eadc9244, 2023.

[29] M. Yang, A. Church, Y. Lin, C. J. Ford, H. Li, E. Psomopoulou, D. A. Barton, N. F. Lepora, et al. Anyrotate: Gravity-invariant in-hand object rotation with sim-to-real touch. In *8th Annual Conference on Robot Learning*, 2024.

[30] X. Liu, H. Wang, and L. Yi. Dexndm: Closing the reality gap for dexterous in-hand rotation via joint-wise neural dynamics model. *arXiv preprint arXiv:2510.08556*, 2025.

[31] H. Qi, B. Yi, M. Lambeta, Y. Ma, R. Calandra, and J. Malik. From simple to complex skills: The case of in-hand object reorientation. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14291–14298. IEEE, 2025.

[32] J. Wang, Y. Yuan, H. Che, H. Qi, Y. Ma, J. Malik, and X. Wang. Lessons from learning to spin" pens". *arXiv preprint arXiv:2407.18902*, 2024.

[33] X. Liu, K. Lyu, J. Zhang, T. Du, and L. Yi. Parameterized quasi-physical simulators for dexterous manipulations transfer. In *European Conference on Computer Vision*, pages 164–182. Springer, 2024.

[34] Y. Chen, C. Wang, Y. Yang, and K. Liu. Object-centric dexterous manipulation from human motion data. In *8th Annual Conference on Robot Learning*, 2024.

[35] X. Liu, J. Adalibieke, Q. Han, Y. Qin, and L. Yi. Dextrack: Towards generalizable neural tracking control for dexterous manipulation from human references. *arXiv preprint arXiv:2502.09614*, 2025.

[36] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[37] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[38] J. Singla, A. Agarwal, and D. Pathak. Sapg: split and aggregate policy gradients. In *Proceedings of the 41st International Conference on Machine Learning*, pages 45759–45772, 2024.

[39] C. Fuji Tsang, M. Shugrina, J. F. Lafleche, O. Perel, C. Loop, T. Takikawa, V. Modi, A. Zook, J. Wang, W. Chen, T. Shen, J. Gao, K. M. Jatavallabhula, E. Smith, A. Rozantsev, S. Fidler, G. State, J. Gorski, T. Xiang, J. Li, M. Li, and R. Lebaredian. Kaolin: A pytorch library for accelerating 3d deep learning research, 2024. URL https://github.com/NVIDIAGameWorks/kaolin.

[40] B. Sundaralingam, A. Murali, and S. Birchfield. curobov2: Dynamics-aware motion generation with depth-fused distance fields for high-dof robots, 2026.

[41] K. Shaw, A. Agarwal, and D. Pathak. Leap hand: Low-cost, efficient, and anthropomorphic hand for robot learning. *arXiv preprint arXiv:2309.06440*, 2023.

[42] K. Kedia, T. G. W. Lum, J. Bohg, and C. K. Liu. Simtoolreal: An object-centric policy for zero-shot dexterous tool manipulation. *arXiv preprint arXiv:2602.16863*, 2026.

[43] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Supplementary Materials of
Towards Human-level Dexterous Teleoperation

This appendix complements the main paper with extended results and full implementation details. Sec. A provides extended results and analysis, including any-to-any reposition tests, additional ablation studies, stage-wise teleoperation analysis, and failure modes. Sec. B details the task definitions, hardware setup, and teleoperation interface. Sec. C details the complete implementation of **TELEDEXTER**. Sec. D details each baseline implementation, and Sec. E details the autonomous policy architecture and training.

A Extended Results and Analysis

A.1 Any-to-Any Reposition Evaluation

Overview This evaluation directly tests whether our co-tracking controller, deployed zero-shot to the real robot, can handle arbitrary in-hand hand-object tracking goal transitions. During teleoperation the operator's goal can jump to any feasible hand-object configuration at any time; the any-to-any reposition test isolates this capability by streaming a sequence of randomly sampled targets and measuring how many the controller reaches consecutively before failure.

Protocol and Metrics We evaluate the co-tracking controller on `Cylinder` and `Cuboid` on LeapHand. We choose LeapHand for this stress test. Despite LeapHand's more limited dexterity (4 fingers, 16 DoFs vs. 5 fingers, 22 DoFs), the controller trained by our framework still demonstrates non-trivial in-hand repositioning capability and robustness, as shown below.

For each object, we construct a hand-object targets pool consisting of all frames from our HOI reference set (Sec. C.6), which covers the full feasible workspace of in-hand configurations. We run 15 trials per object. Each trial begins by sampling an initial grasp pose from the pool; the tester places the object into the hand accordingly and starts the controller. The controller is then queried with a stream of targets, each freshly sampled from the same pool as soon as the previous one is reached. A target is counted as reached when the object position error falls below $2\,\text{cm}$ and the object rotation error falls below 20° simultaneously. A trial terminates when (i) the object slips out of the hand, or (ii) the active target has not been reached for $20\,\text{s}$. We report two metrics: consecutive successes, the number of targets reached in sequence before failure (averaged over trials), and per-target success rate, the fraction of all attempted targets that are successfully reached.

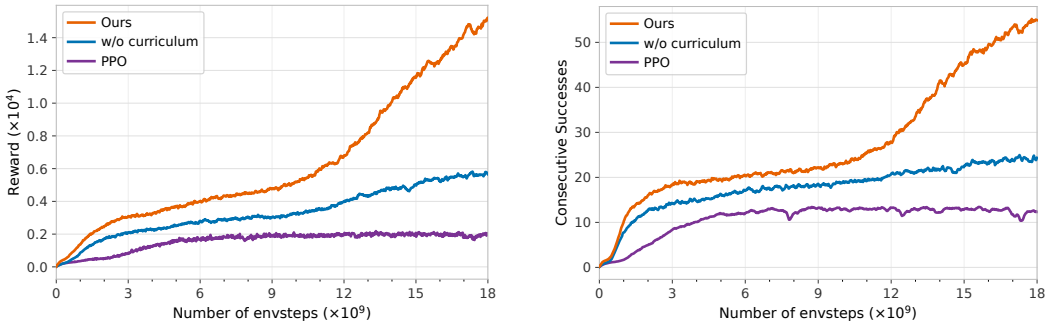
Results and Analysis As shown in Tab. 6, the controller sustains long sequences of arbitrary goal transitions on both objects, confirming that it generalizes well beyond the training reference trajectories. On `Cylinder` the controller reaches 41.1 consecutive targets on average with a per-target success rate of 97.6%, substantially outperforming `Cuboid` (12.1 consecutive successes, 92.3%). We attribute this gap to object geometry: the cylinder's continuous surface allows fingers to slide and roll the object fluidly during transitions without encountering abrupt geometric changes, and its rotational symmetry reduces the effective distance between arbitrary target poses. The cuboid's edges and corners, by contrast, can obstruct finger motion during rapid regrasps — fingers must navigate around these features to reach certain target orientations, increasing the chance of contact transition jams and failed repositioning attempts.

Tab. 6: **Any-to-any results on LeapHand.** Each object is evaluated over 15 trials.

Object	Consecutive Successes (↑)	Per-target SR (↑)
`Cylinder`	41.1	97.6%
`Cuboid`	12.1	92.3%

A.2 Additional Ablation Studies

We ablate two training recipe choices on `Hammer`: the curriculum schedule (Sec. C.3) and the policy optimizer (SAPG vs. PPO). All other training settings and reference motions are shared. We report training curves of reward and consecutive subgoals reached in Fig. 5.



(a) Reward. (b) Consecutive successes.

Fig. 5: **Training-recipe ablation curves on `Hammer`.** (a) Reward and (b) consecutive subgoals reached vs. environment steps for the full method, without curriculum, and with PPO replacing SAPG.

Curriculum Schedule As shown in Fig. 5, disabling the curriculum (training at full deployment difficulty throughout) produces faster initial progress but plateaus at a lower final performance. The curriculum completes its annealing within the first 2×10^9 environment steps, yet the advantage it provides continues to grow well beyond that point. We attribute this to the quality of the early-training foundation: the curriculum first exposes the policy to small subgoal steps, permissive tolerances, and reduced gravity, allowing it to discover a diverse repertoire of stable contact primitives before difficulty ramps up. Without this scaffolding, the policy must simultaneously learn basic contact strategies and cope with full-difficulty dynamics, converging to a narrower set of behaviors that limits its ability to compose longer manipulation sequences later in training.

PPO vs. SAPG As shown in Fig. 5, replacing SAPG [38] with vanilla PPO causes a substantial drop in both reward and consecutive successes. SAPG maintains multiple independently exploring policy blocks whose gradients are aggregated, promoting diverse strategy discovery within a single training run. This exploration diversity is critical for co-tracking, where the policy must master qualitatively different contact modes — translation, continuous rotation, finger gaiting, and regrasping — within a single network. PPO's unimodal gradient updates tend to commit early to a limited strategy set, under-covering the full spectrum of in-hand manipulation modalities in our reference motions.

A.3 Stage-Wise Teleoperation Success Analysis

For each of the four long-horizon tool-use tasks, we plot the number of trials (out of 15) that survive past each task stage for **TELEDEXTER** and all baselines (Fig. 6). The main paper highlights key stage-wise numbers for **TELEDEXTER**; here we provide the complete per-stage breakdown across all methods and analyze where each baseline breaks down. We additionally include *SimToolReal* [42], an object-centric sim-to-real tool-use policy that is not a teleoperation method but provides a strong reference for learned dexterous tool manipulation.

Across all four tasks, a consistent pattern emerges. **TELEDEXTER** retains 13–15 out of 15 trials through the demanding mid-task stages and only drops modestly at the final placement stages, whereas every baseline suffers a sharp collapse at or shortly after the first stage that requires in-hand reorientation or functional grasp transition.

In `HammerUse`, all methods pick up the hammer (stage 1), but baselines diverge sharply at stage 2 (rotate face-down): *DexRT* drops from 15 to 6 and *DexGen* from 14 to 10. By stage 4 (rotate claw-down), no teleoperation baseline retains any trial. *SimToolReal*† (category-specific) leverages task-specific training to reach $14/15$ at stage 2 but collapses entirely at stage 3 (drive nails), unable

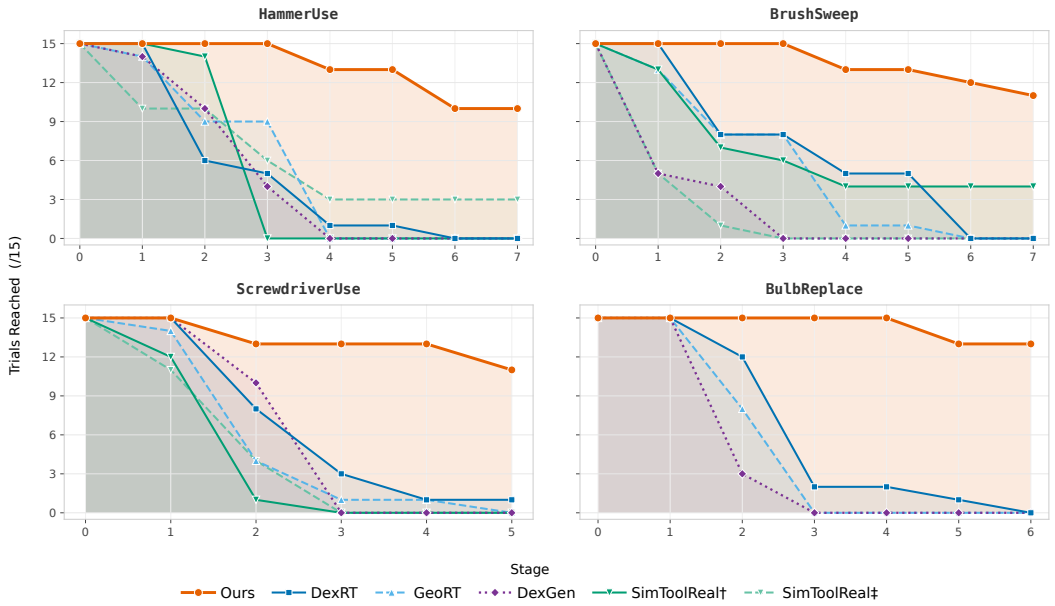


Fig. 6: **Stage-wise success on four long-horizon tool-use tasks.** For each task, the horizontal axis indexes the task stage and the vertical axis is the number of trials (out of 15) that reach that stage. Each curve corresponds to one method (**TELEDEXTER** and baselines from Tab. 1 in the main paper). *SimToolReal* is not a teleoperation method but is included as a strong baseline for learned tool manipulation.

to sustain the repeated contact force required for hammering. *SimToolReal*‡ (all-category) maintains $3/15$ through completion at the cost of weaker early-stage performance. **TELEDEXTER** retains all 15 trials through stage 3 and 10 through final placement.

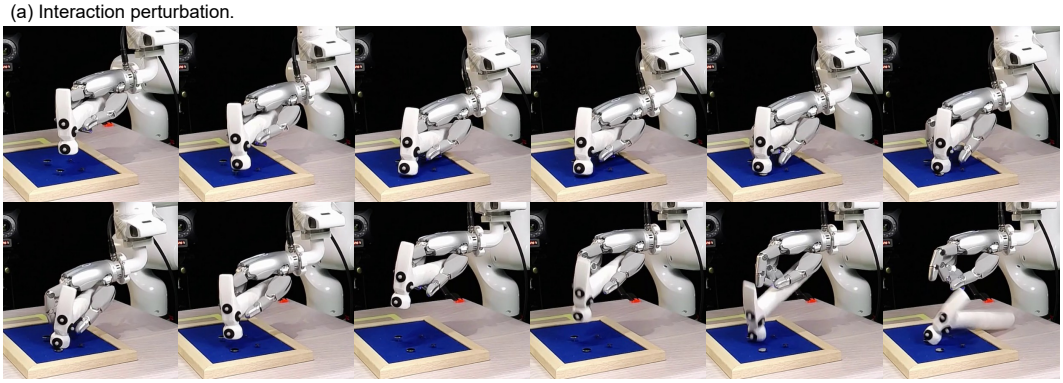
In `BrushSweep`, a similar bottleneck emerges at grasp-transition stages. *DexGen* collapses earliest, losing most trials at pick-up ($5/15$) due to compounding generative-model errors. *DexRT* and *GeoRT* survive the initial sweep (stage 3, $8/15$ each) but fail at stage 4 (rotate bristles-right), which demands controlled in-hand reorientation while maintaining grasp. *SimToolReal*† is the strongest baseline on this task with $4/15$ completions, while **TELEDEXTER** reaches $11/15$.

In `ScrewdriverUse`, *DexGen* maintains $10/15$ through stage 2 (rotate to align) but drops to 0 at stage 3 (tighten the screw), as its action prior cannot sustain continuous axial rotation. Both *SimToolReal* variants also fail entirely at stage 3. *DexRT* retains $1/15$ to completion, the only teleoperation baseline trial to finish this task. **TELEDEXTER** maintains $13/15$ through tightening and finishes with $11/15$.

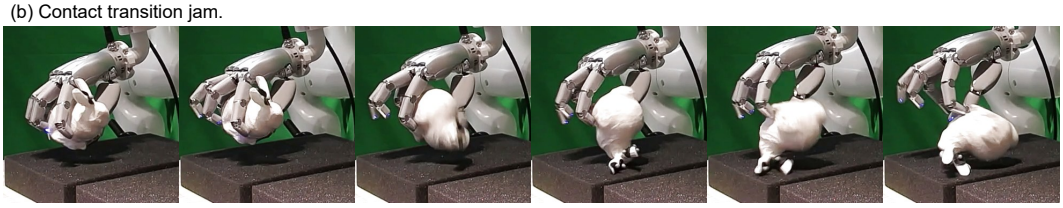
In `BulbReplace`, evaluated without *SimToolReal* as the task falls outside its object-centric formulation, all methods pick up the bulb ($15/15$), but the critical drop occurs at stage 3 (screw in), where *DexRT* falls from 12 to 2 and both *GeoRT* and *DexGen* reach 0. These stages require precise bidirectional rotation about the bulb axis under sustained contact, which kinematic retargeting cannot achieve. **TELEDEXTER** completes both rotational stages without trial loss ($15/15$ through stage 4) and finishes with $13/15$.

Taken together, these results show that all baselines fail not at grasping or gross positioning, but at contact-rich in-hand manipulation: reorientation, finger gaiting, and sustained tool application. Kinematic methods (*DexRT*, *GeoRT*) lack a dynamics prior and collapse at the first contact-intensive stage. *DexGen* suffers compounding trajectory drift that causes abrupt failure at contact transitions. *SimToolReal* achieves partial success on trained tool categories but cannot generalize across the diverse contact modes within a single long-horizon task. **TELEDEXTER** sustains high trial survival precisely at these stages, confirming that the co-tracking controller provides the in-hand dexterity needed for long-horizon tool use.

(a) Interaction perturbation.



(b) Contact transition jam.



(c) Tracking stall.

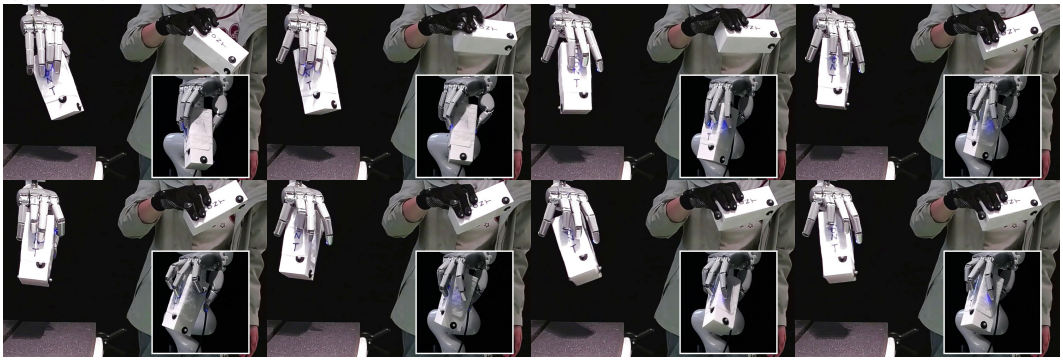


Fig. 7: **Real-world failure cases.** Each panel shows a snapshot at the point of failure together with the representative failure mode: (a) interaction perturbation, (b) contact transition jam, (c) tracking stall.

A.4 TELEDEXTER Failure Analysis

We identify three dominant failure modes of our teleoperation controller, illustrated in Fig. 7.

- **Interaction perturbation (Fig. 7a).** Forceful tool–environment contact, such as striking a nail during `HammerUse`, generates impulsive reaction forces that shift the object's in-hand pose by a large, instantaneous amount. The controller, trained exclusively on free-space hand–object interaction, has never encountered such impact dynamics and cannot recover from the resulting out-of-distribution grasp configuration. This mode does not appear in reorientation-only tasks.
- **Contact transition jam (Fig. 7b).** During finger gaiting or in-hand reorientation, a finger that should lift away occasionally remains wedged against the object due to actuator compliance or geometric interlocking. As the remaining fingers continue to move, the jam generates unbalanced forces that eject the object from the hand. This failure is most frequent on irregular objects (`BunnyReorient`) where concavities increase the chance of interlocking.
- **Tracking stall (Fig. 7c).** The controller attempts a regrasp but fails to establish contacts that can move the object toward the target pose. Unlike the contact transition jam, the object is not lost—the hand simply cannot make progress. We attribute this to the absence of tactile observations (Tab. 7): the policy cannot distinguish between a finger pressing against the object and one sliding past it, and thus cannot adapt when a regrasp attempt fails.

The three modes point to distinct limitation of our current system. Interaction perturbation reflects a *training distribution* gap, as tool–environment impact dynamics are absent from the simulation training. Contact transition jams reveal an *actuation compliance* mismatch between rigid-body simulation and real direct-drive fingers with passive compliance. Tracking stalls expose the lack of *tactile observation*, without which the policy cannot close the loop on contact state during regrasping. Addressing these limits through interaction-aware training, compliant-contact simulation, and tactile-rich observation spaces is a promising direction for future work.

B Experimental Setup Details

B.1 Teleoperation Task Definitions

We provide detailed descriptions of the seven real-world tasks introduced in the main paper (Fig. 3), together with their stage decompositions. Each task is decomposed into N well-defined stages that the trial must reach in order. The task progress ($\mathbf{TP}$) reported in the main paper is the average k/N of the furthest stage k reached across trials, and the success rate ($\mathbf{SR}$) is the fraction of trials that reach all N stages. For all tasks, stage $0/N$ denotes failure to pick up the object. We wrap the tested objects with medical bandage tape to increase surface friction.

`CylinderReorient` / `CuboidReorient` / `BunnyReorient` **(3 stages each).** The three reorientation tasks share an identical stage decomposition and differ only in object geometry, testing continuous in-hand pose control on rotationally symmetric (`CylinderReorient`), corner-rich (`CuboidReorient`), and irregular freeform (`BunnyReorient`) geometries. *Stages:*

- $1/3$: picked up the object.
- $2/3$: reoriented to the target pose in-hand.
- $3/3$: placed back on the table.

`HammerUse` **(7 stages).** A long-horizon task that exercises bidirectional functional grasp transitions (face-down vs. claw-down) and repeated striking and pulling. *Stages:*

- $1/7$: picked up the hammer.
- $2/7$: rotated face-down for hammering.
- $3/7$: drove the two nails into the board.
- $4/7$: rotated claw-down.
- $5/7$: pulled out the two nails with the claw.
- $6/7$: rotated parallel to the table.
- $7/7$: placed back on the table.

`BrushSweep` **(7 stages).** A long-horizon task requiring transitions between two functional brush orientations while sweeping across an extended workspace. *Stages:*

- $1/7$: picked up the brush.
- $2/7$: rotated for forward sweeping.
- $3/7$: swept the debris forward.
- $4/7$: rotated so the bristles face rightward.
- $5/7$: swept the debris rightward into the target area.
- $6/7$: rotated parallel to the table.
- $7/7$: placed back on the table.

`ScrewdriverUse` **(5 stages).** A precision task that tests axial alignment with the screw and sustained in-hand rotation about the tool axis. *Stages:*

- $1/5$: picked up the screwdriver.
- $2/5$: rotate the screwdriver tip downward..
- $3/5$: tightened the screw until fully seated.
- $4/5$: rotated the screwdriver for placement.
- $5/5$: placed back on the table.

`BulbReplace` (6 **stages).** A precision insertion task that combines functional in-hand reorientation with bidirectional rotation about the bulb axis. *Stages:*

- $1/6$: picked up the bulb.
- $2/6$: rotated to align with the socket.
- $3/6$: screwed the bulb in until it lit up.
- $4/6$: unscrewed it until fully removed.
- $5/6$: rotated for placement.
- $6/6$: placed back on the table.

B.2 Autonomous Policy Task Definitions

The autonomous Diffusion Policies are evaluated on simplified subsets of three teleoperation tasks, each retaining the most dexterous stages while removing the return-and-place phases. Stage decompositions follow the same protocol as the teleoperation tasks (Sec. B.1); a trial is terminated when an unrecoverable failure occurs.

`BulbInstall` (4 **stages).** A subset of `BulbReplace` that covers bulb installation only (no unscrewing or return). *Stages:*

- $1/4$: picked up the bulb.
- $2/4$: reoriented to the installation pose.
- $3/4$: aligned with the socket.
- $4/4$: screwed in until the bulb lit up.

`HammerDriver` (3 **stages).** A simplified variant of `HammerUse` with larger nails and a foam target, retaining the core in-hand rotation to a functional hammering grasp. *Stages:*

- $1/3$: picked up the hammer.
- $2/3$: rotated face-down for hammering.
- $3/3$: drove the nails into foam.

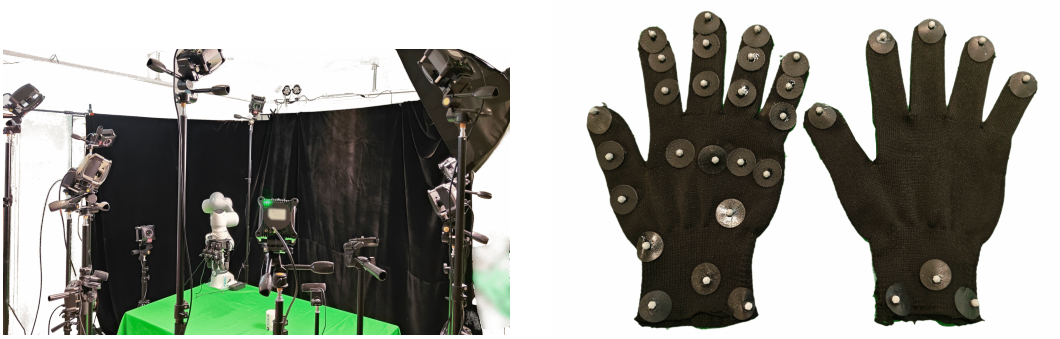
`BrushForward` (3 **stages).** A subset of `BrushSweep` that covers a single forward sweep only (no bristle-reorientation or return sweep). *Stages:*

- $1/3$: picked up the brush.
- $2/3$: rotated for forward sweeping.
- $3/3$: swept forward across the workspace.

B.3 Hardware and Teleoperation System

B.3.1 Motion Capture System

We use a NOKOV optical motion capture system for both offline reference-motion collection and real-time teleoperation, with a different glove configuration for each setting. For offline collection (Fig. 8a), the operator stands in a dedicated capture volume equipped with tripod-mounted infrared cameras and wears a full-coverage glove instrumented with retro-reflective markers on the wrist, palm, and all finger joints (Fig. 8b, left). This dense marker layout enables the system to output, per frame, (i) 24 hand joint 3-D positions, (ii) 21 joint pose matrices, (iii) the wrist $\mathrm{SE}(3)$ pose, and (iv) the object's 6-D pose. For real-time teleoperation, the operator wears a compact glove with markers placed only on the wrist and fingertips (Fig. 8b, right) at the deployment workstation. This lightweight configuration provides the wrist pose and fingertip positions needed by the control loop. The MoCap stream is fed directly into the teleoperation loop at 30 Hz. In both settings, each rigid object is tagged with a marker rigid body that yields its 6-D pose.



(a) Motion capture setup. (b) Marker-instrumented gloves.

Fig. 8: **Hand–object motion capture setup.** (a) The dedicated capture volume equipped with tripod-mounted NOKOV infrared cameras. (b) Two glove configurations: the left glove, used for offline reference-motion collection, has dense markers on the wrist, palm, and all finger joints; the right glove, used for real-time teleoperation, has markers only on the wrist and fingertips.

B.3.2 Teleoperation Interface

At runtime, the NOKOV system (Sec. B.3.1) streams the operator's wrist pose and fingertip positions together with the manipulated object's 6-D pose at 30 Hz. Each frame triggers two parallel control paths that together close the teleoperation loop at the same rate:

- **Arm.** The operator's wrist pose (world frame) is mapped to a 7-DoF Franka FR3 joint target via inverse kinematics.
- **Hand.** The operator's fingertip positions and the object's 6-D pose are converted to the wrist frame and assembled into the co-tracking goal g_t. The learned policy then maps the current hand–object state and g_t to joint position targets sent to the dexterous-hand SDK via position control.

The co-tracking policy is trained on in-hand HOI references (Sec. C.6) and does not cover the pre-grasp approach. We therefore split each trial into two phases. In the *reaching and grasping phase*, the hand is driven by kinematic vector retargeting [5, 6], which maps the operator's fingertip positions to robot joints via vector alignment and handles the pre-grasp approach and initial contact acquisition. Once the hand reaches an approximate grasp pose near the object, the operator switches to the *in-hand phase*, where the co-tracking policy drives all subsequent in-hand reorientation, re-grasping, and finger gaiting.

C TELEDEXTER Implementation Details

This section provides the full implementation details of **TELEDEXTER**. We follow the same notation as Sec. 3 unless stated otherwise.

C.1 Observation & Action Space

Observation Space The observation $\boldsymbol{o}_t$ concatenates the elements listed in Tab. 7 (SharpaWave: $N_f = 5$, $n_{\text{dof}} = 22$; LeapHand: $N_f = 4$, $n_{\text{dof}} = 16$). Two design choices are worth highlighting: (i) since every other entry is expressed in $\mathcal{F}_{\text{wrist}}$, the gravity direction $\hat{\boldsymbol{g}}_{\text{wrist}}$ is the *only* signal that anchors the policy to the world frame and tells it the absolute orientation of the wrist; and (ii) we deliberately exclude joint velocities and contact forces, since both are noisy or unavailable on hardware and the policy can implicitly recover them from the kinematic state, previous action, and gravity cue.

Action Space Extending the residual joint-target parameterization of HORA [27] with a soft deadzone, the policy output $\boldsymbol{a}_t \in [-1, 1]^{n_{\text{dof}}}$ is converted into a joint command by

$$\boldsymbol{a}_t^{\text{cmd}} = \text{clip}_{\text{joint}}\Big(\boldsymbol{a}_{t-1}^{\text{cmd}} + \alpha_a \cdot \text{deadzone}_\tau(\boldsymbol{a}_t)\Big), \quad \text{deadzone}_\tau(a) = \text{sign}(a)\ \max\big(|a| - \tau,\, 0\big),$$

Tab. 7: **Observation o_t: per-element dimensions.** $\hat{\cdot}$ denotes a target quantity; Δ denotes target $-$ current.

Element	Description	Formula	SharpaWave	LeapHand
$\boldsymbol{q}_t$	joint positions	n_{dof}	22	16
$\cos \boldsymbol{q}_t$	cosine of joint positions	n_{dof}	22	16
$\sin \boldsymbol{q}_t$	sine of joint positions	n_{dof}	22	16
$\boldsymbol{x}_t^o$	object position in $\mathcal{F}_{\text{wrist}}$	3	3	3
$\boldsymbol{q}_t^o$	object quaternion in $\mathcal{F}_{\text{wrist}}$	4	4	4
$\hat{\boldsymbol{g}}_{\text{wrist}}$	gravity direction in $\mathcal{F}_{\text{wrist}}$	3	3	3
$\hat{\boldsymbol{p}}_t^{\text{tip}}$	target fingertip positions	$3N_f$	15	12
$\Delta\hat{\boldsymbol{p}}_t^{\text{tip}}$	fingertip target $-$ current	$3N_f$	15	12
$\hat{\boldsymbol{x}}_t^o$	target object position	3	3	3
$\Delta\hat{\boldsymbol{x}}_t^o$	object-position target $-$ current	3	3	3
$\hat{\boldsymbol{q}}_t^o$	target object quaternion	4	4	4
$\Delta\hat{\boldsymbol{q}}_t^o$	object-quaternion target $-$ current	4	4	4
$\tilde{\boldsymbol{a}}_{t-1}$	previous low-level command	n_{dof}	22	16
Total $\lvert\boldsymbol{o}_t\rvert$			**142**	**112**

with action scale $\alpha_a = 0.1$, deadzone threshold $\tau = 0.1$. Under this residual formulation, exactly outputting zero is hard for a Gaussian policy; the deadzone gives an explicit "hold-still" region in action space (any $\boldsymbol{a}_t$ within $\pm\tau$ produces zero delta), letting the policy actively choose to keep the current command rather than having to emit exactly zero.

C.2 Complete Reward Design

We give the complete form of the reward used to train the co-tracking controller. The reward couples a sparse *consecutive subgoal-reaching* term, a dense *tracking* term, and a small time penalty:

$$r_t \;=\; \underbrace{\mathbb{1}_{\text{reach}}(t)\, w_{\text{step}}(t)\, r_{\text{score}}(t)}_{\text{sparse subgoal}} \;+\; \underbrace{\alpha_{\text{dense}}\, r_{\text{dense}}(t)}_{\text{dense tracking}} \;-\; \underbrace{c_{\text{time}}}_{\text{time}}, \tag{5}$$

Subgoal Indicator $\mathbb{1}_{\text{reach}}(t)$ At each step, the active subgoal g_k is considered *instantaneously reached* when all tracking errors fall below their tolerances simultaneously: $e_{t,k}^{\text{pos}} < \epsilon_{\text{pos}}$, $e_{t,k}^{f} < \epsilon_{\text{tip}}$ for every fingertip $f \in \{\text{thumb}, \text{index}, \text{middle}, \text{ring}, \text{pinky}\}$, and $e_{t,k}^{\text{rot}} < \epsilon_{\text{rot}}$. The indicator $\mathbb{1}_{\text{reach}}(t)$ fires only when this condition is held for N_{stay} consecutive frames, where N_{stay} is resampled after every successful subgoal hit.

Step Weighting $w_{\text{step}}(t)$ After a subgoal g_k is hit, the next subgoal is drawn from the same reference trajectory at index $k' = k + \Delta k$, where the jump $\Delta k \in \mathbb{Z}$ is drawn from a uniform distribution whose range expands over training according to the curriculum (Sec. C.3); $|\Delta k|$ is the number of reference frames skipped between two consecutive subgoals. The step-weighting factor w_{step} scales the sparse bonus by this temporal gap so that larger jumps yield proportionally larger rewards and the policy is not biased toward exploiting trivially close subgoals. When the environment performs a cross-trajectory switch (Sec. C.4), w_{step} is set to a much larger fixed value for the reward computation at that step instead. Both values are listed in Tab. 8.

Subgoal Score $r_{\text{score}}(t)$ The score blends per-fingertip and object tracking terms with fixed weights:

$$r_{\text{score}} = \alpha_s \Big[\; w_{\text{tip}}^s \sum_{f \in \mathcal{F}} \rho_f \;+\; w_{\text{obj}}^s \left(\rho_{\text{pos}} + \rho_{\text{rot}}\right) \Big], \tag{6}$$

where the fingertip set $\mathcal{F}$ consists of thumb, index, middle, ring, and pinky for SharpaWave, with the ring finger omitted for LeapHand. Each $\rho_\cdot$ is an exponential kernel applied to the corresponding tracking error—the per-fingertip distance $e_{t,k}^{f}$, the object position error $e_{t,k}^{\text{pos}}$, and the object rotation error $e_{t,k}^{\text{rot}}$ in radians:

$$\rho_f = \exp\!\big(-\beta_f\, e_{t,k}^{f}\big), \quad \rho_{\text{pos}} = \exp\!\big(-\beta_{\text{pos}}\, e_{t,k}^{\text{pos}}\big), \quad \rho_{\text{rot}} = \exp\!\big(-\beta_{\text{rot}}\, |e_{t,k}^{\text{rot}}|\big). \tag{7}$$

Outer scale $\alpha_s = 1.5$. The remaining blend weights and decay rates β (following Li et al. [23]) are listed in Tab. 8.

Dense Tracking $r_{\text{dense}}(t)$ Following Li et al. [23], a small dense signal shapes early exploration before any subgoal is reached:

$$r_{\text{dense}} = \sum_{i \in \mathcal{I}} w_i^d \, \rho_i + (1 - \sigma_t) \left(w_{\text{pos}}^d \, \rho_{\text{pos}} + w_{\text{rot}}^d \, \rho_{\text{rot}} \right), \tag{8}$$

where $\mathcal{I} = \{\text{thumb, index, middle, ring, pinky, lvl1, lvl2}\}$, with *lvl1* the MCP (root) knuckles and *lvl2* the medial (proximal) knuckles, each averaged across fingers, sharing the same exponential kernel form as the per-finger ρ_f. The $(1 - \sigma_t)$ factor turns the dense object signal off early in training and ramps it in as the curriculum hardens (Sec. C.3). Weight values w_i^d and decay rates β are listed in Tab. 8; the overall dense-reward scale in Eq. (5) is $\alpha_{\text{dense}} = 0.1$.

Tab. 8: **Reward parameters.** Full set of constants used in the reward function (Eq. (5)), grouped by reward component.

Parameter	Value
Subgoal indicator	
ϵ_{pos} (object pos. tolerance)	1 cm
ϵ_{tip} (fingertip tolerance)	3 cm
ϵ_{rot} (object rot. tolerance)	10°
N_{stay} (dwell duration)	$\mathcal{U}\{5, 15\}$
Step weighting w_{step}	
in-traj	$\lvert\Delta k\rvert + 5$
cross-traj	100
Subgoal score r_{score}	
α_s (outer scale)	1.5
w_{tip}^s (per-fingertip)	0.5
w_{obj}^s (pos / rot)	2.0
Time penalty	
c_{time}	0.1

Parameter	Value
Kernel decay rates β	
β_{thumb}	100
$\beta_{\text{index}} = \beta_{\text{middle}} = \beta_{\text{ring}} = \beta_{\text{pinky}}$	90
β_{lvl1}	50
β_{lvl2}	40
β_{pos}	80
β_{rot} (rad)	3
Dense tracking r_{dense}	
w_{thumb}^d	1.0
$w_{\text{index}}^d = w_{\text{middle}}^d = w_{\text{ring}}^d = w_{\text{pinky}}^d$	0.8
w_{lvl1}^d	0.6
w_{lvl2}^d	0.4
$w_{\text{pos}}^d = w_{\text{rot}}^d$	1.5
α_{dense} (outer scale)	0.1

Time Penalty A constant per-step cost $c_{\text{time}} = 0.1$ pressures the policy to complete subgoals quickly and prevents it from lingering in locally stable but unproductive configurations.

C.3 Curriculum Schedule

Following the curriculum design of Li et al. [23], we progressively harden four difficulty knobs over training. Three of them are jointly controlled by a scalar progress factor σ_t that decays from 1 to $\sigma_{\min} = 0.7$, and simulator gravity follows its own schedule. The four knobs are:

- **Inter-subgoal step size** (driven by σ_t). After each successful subgoal hit (Sec. C.4), the next subgoal is drawn $|\Delta k|$ reference frames ahead, with the upper bound on $|\Delta k|$ growing from 40 frames ($\sim$ 0.67 s at the 60 Hz reference rate) to $\sim$ 80 frames ($\sim$ 1.33 s) over training. Closer subgoals are easier to reach, so the policy first learns to reach nearby subgoals and only later is asked to reach farther ones.
- **Action-masking duration** (driven by σ_t). The freeze duration of the random action mask (Sec. C.8) is sampled uniformly from $\{1, \ldots, d_t^{\max}\}$, with the upper bound $d_t^{\max}$ growing from 1 to 10 frames over training. Shorter freezes are easier to tolerate, so the policy first faces brief freezes and only later is exposed to longer ones.
- **Dense object reward** (driven by σ_t). The object-tracking term $(\rho_{\text{pos}} + \rho_{\text{rot}})$ in r_{dense} (Eq. (8)) is scaled by $(1 - \sigma_t)$, growing from zero early on to its full value late, so the policy first learns to track the fingertips and only later learns to track the object pose.

- **Simulator gravity** (independent schedule). Gravity is annealed linearly from 0 to $-9.8\,\mathrm{m/s^2}$ over the first 32 K environment frames, so the policy first learns to manipulate under reduced object weight and only later has to support the full deployment load.

Schedules The progress factor σ_t decays linearly over an annealing window of $T_\sigma = 25{,}600$ environment frames (counted per env), where t is the per-env frame counter:

$$\sigma_t = 1 - (1 - \sigma_{\min}) \cdot \min\big(t/T_\sigma,\, 1\big).$$

Each σ_t-driven upper bound expands cubically toward its saturation value $u_{\max}$ from its initial value $u_{\min}$,

$$u_t^{\max} = u_{\min} + \frac{u_{\max} - u_{\min}}{1 - \sigma_{\min}^3}\left(1 - \sigma_t^3\right),$$

instantiated with $(u_{\min}, u_{\max}) = (40, 80)$ for the inter-subgoal step bound $k_t^{\max}$ and $(1, 10)$ for the action-mask duration bound $d_t^{\max}$.

C.4 Reset Conditions

Goal Reset Whenever a subgoal g_k is hit, the next reference trajectory τ_{next} and subgoal index k_{next} are sampled as

$$(\tau_{\text{next}},\, k_{\text{next}}) \;=\; \begin{cases} (\tau,\; k + \Delta k) & \text{with probability } 0.9, \\ (\tau' \sim \mathrm{Unif}(\mathcal{D}_o),\; k) & \text{with probability } 0.1 \quad (\textit{cross-trajectory switch}), \end{cases} \tag{9}$$

where τ is the current trajectory, $\mathcal{D}_o$ is the reference set for the current object o, and Δk is drawn from a uniform distribution whose range expands over training according to the curriculum (Sec. C.3). The next subgoal is then read as the k_{next}-th frame of τ_{next}. A cross-trajectory switch swaps the trajectory while preserving the frame index. Its purpose is to train the policy to handle transitions between different demonstrations rather than overfitting to any single one. This generality is required at deployment, where the operator's motion will not stay within any recorded trajectory.

Episode Reset An episode ends if (i) any joint velocity or object linear or angular velocity exceeds its safety bound, (ii) the object position error exceeds 15 cm, or (iii) the policy accumulates too many out-of-tolerance frames before reaching the next subgoal:

$$n_{\text{fail}} \leftarrow \begin{cases} n_{\text{fail}} + 1, & \text{frame out of tolerance}, \\ 0, & \text{subgoal hit}, \end{cases} \qquad \text{terminate if } n_{\text{fail}} > n_{\text{fail}}^{\max} = \eta_{\text{fail}}\,|\Delta k|,$$

where "out of tolerance" means *any* of the five fingertip, object position, or object rotation errors is above threshold, and η_{fail} is the failure-tolerance scale, meaning the policy is allowed η_{fail} out-of-tolerance frames per unit of subgoal step before the episode terminates. We use $\eta_{\text{fail}} = 1.5$. Immediately after a cross-trajectory switch, $n_{\text{fail}}^{\max}$ is overridden by a flat 300 frames to accommodate the longer regrasp transition. On termination, the environment is re-initialized via *reference state initialization* (RSI) [23]: a random frame in the first 90% of the assigned trajectory sets hand DoFs from the retargeted $\boldsymbol{q}^*$ and the object pose from the reference.

C.5 Geometry-Aware Retargeting Details

We define each loss term in Eq. (4) and specify the two-stage optimization procedure and loss weights below. The main paper provides compact inline definitions of each loss term; here we give the full formulation together with the optimization procedure and hyperparameters. Let $\mathcal{H}_t$ denote the robot hand model at frame t with joint configuration $\boldsymbol{q}_t$ and wrist pose $(\boldsymbol{p}_t^w, R_t^w)$, and $\mathcal{O}_t$ the manipulated object mesh transformed by the recorded object pose T_t^o. Each link of $\mathcal{H}_t$ carries a precomputed surface point cloud and, for self-collision, a set of body-attached collision spheres. All signed distances $\mathrm{sdf}_{\mathcal{O}_t}(\cdot)$ from a point on the hand surface to the object (positive outside) are computed with NVIDIA Kaolin [39], which provides a differentiable point-to-mesh SDF used by both $\mathcal{L}_{\text{surf}}^t$ and $\mathcal{L}_{\text{pen}}^t$ below.

Vector Alignment $\mathcal{L}^t_{\text{vec}}$ We use the standard vector-retargeting loss of Qin et al. [5], Handa et al. [6] without modification: a weighted Huber on per-vector errors between the captured operator hand keypoints (Sec. C.6) and the corresponding robot vectors. We refer readers to the original papers for the exact loss form, keypoint vector set, and per-vector weights.

Surface Attraction $\mathcal{L}^t_{\text{surf}}$ In the second stage we incorporate the object mesh: points on the hand surface that fall within a threshold τ_{surf} of the object are pulled onto the surface,

$$\mathcal{L}^t_{\text{surf}} = \frac{1}{|\mathcal{S}_t|} \sum_{p \in \mathcal{S}_t} \text{ReLU}(\text{sdf}_{\mathcal{O}_t}(p)), \quad \mathcal{S}_t = \{p : \text{sdf}_{\mathcal{O}_t}(p) < \tau_{\text{surf}}\}.$$

This glues the contact-side of the hand to the object without forcing non-contacting links onto it.

Mesh Interpenetration $\mathcal{L}^t_{\text{pen}}$ A symmetric penetration penalty pulls any hand surface point that has entered the object back out:

$$\mathcal{L}^t_{\text{pen}} = \sum_{p \in \mathcal{H}_t} \text{ReLU}\big(-\text{sdf}_{\mathcal{O}_t}(p)\big).$$

The weights λ_{pen} and λ_{surf} are listed in the optimization paragraph below.

Self-Collision $\mathcal{L}^t_{\text{col}}$ Self-collision is computed between collision spheres on *different* fingers: for every pair of spheres (i, j) with finger indices $f(i) \neq f(j)$, radii r_i, r_j and centers $\boldsymbol{c}_i, \boldsymbol{c}_j$,

$$\mathcal{L}^t_{\text{col}} = \tfrac{1}{2} \sum_{f(i) \neq f(j)} \text{ReLU}\big((r_i + r_j) - \|\boldsymbol{c}_i - \boldsymbol{c}_j\|_2\big).$$

Intra-finger sphere pairs are ignored because adjacent links are designed to touch.

Trajectory Smoothness $\mathcal{L}_{\text{smooth}}$ We adopt the temporal smoothness energy of Curobo [40] without modification: a log-cosh penalty on the velocity, a squared ℓ_2 penalty on the acceleration, and a squared ℓ_2 penalty on the jerk of the per-frame joint configuration $\boldsymbol{q}_t$, summed across the trajectory with internal coefficients fixed to the Curobo defaults.

Optimization The two stages are run sequentially per trajectory:

- **Stage 1 (vector retargeting).** We minimize $\sum_t \mathcal{L}^t_{\text{vec}}$ jointly over all frames $\boldsymbol{q}_{1:T}$ with Adam, using a GPU-batched FK implementation that follows the dex-retargeting design [5]. The learning rate is 10^{-3} and we run 6,000 iterations. This gives a strong but contact-blind initialization.
- **Stage 2 (geometry-aware post-optimization).** From the Stage 1 solution we minimize the remaining loss terms of Eq. (4) with Adam. The learning rate is 3×10^{-3}, we run 160 iterations, and the gradient norm is clipped to 1.0. The surface-attraction mask $\mathcal{S}_t$ is built once at the start of Stage 2 with $\tau_{\text{surf}} = 2$ and frozen for all iterations.

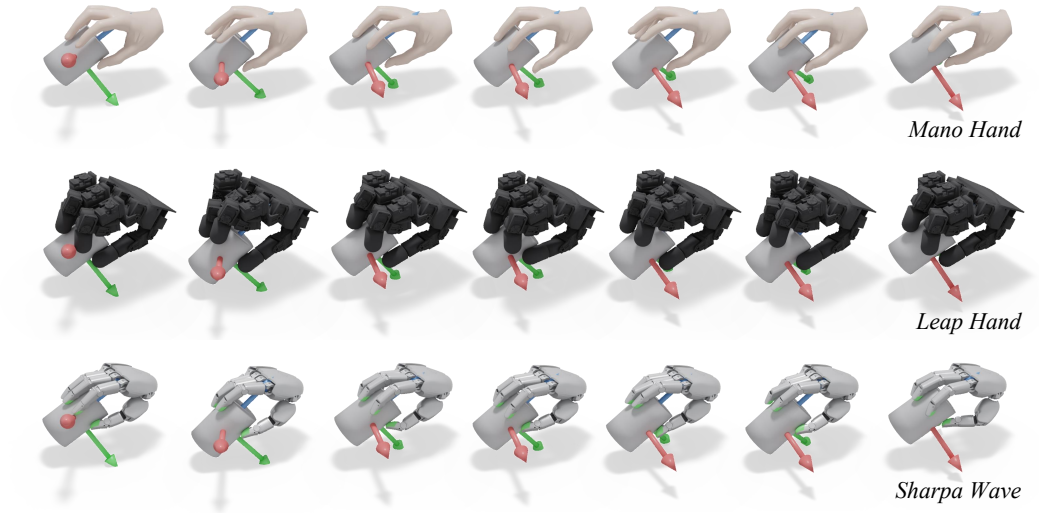
We set $\lambda_{\text{surf}} = 10$, $\lambda_{\text{pen}} = 2$, $\lambda_{\text{col}} = 10$, $\lambda_{\text{smooth}} = 0.1$ for both SharpaWave and LeapHand. The final per-frame robot configuration $\boldsymbol{q}^*_t$, together with the recorded object pose T^o_t, forms the reference motion consumed by RL.

C.6 Hand–Object Reference Motion Construction

For each object, we record 150 reference trajectories of unscripted hand–object interactions using the NOKOV MoCap system (Sec. B.3.1) at 30 Hz. Each trajectory lasts 20 s (600 frames), giving a total of $\sim$ 50 minutes of interaction data per object. The interactions span three categories: (1) in-hand translation, (2) in-hand rotation, and (3) free-play combining arbitrary grasps, finger gaiting, and tool-use sequences. This diversity ensures the reference set covers the full range of contact modes the controller may encounter at deployment.

Fig. 9 visualizes short reference-motion clips for `Cylinder` and `Cuboid`, each showing the source MANO hand alongside the retargeted LeapHand and SharpaWave configurations. The retargeted robot hands accurately reproduce the operator's grasp poses and contact transitions across both embodiments, confirming that the two-stage retargeting pipeline (Sec. C.5) transfers contact-rich interaction structure despite the significant kinematic differences between the human hand and the two robot platforms.

(a) *Cylinder*



(b) *Cuboid*

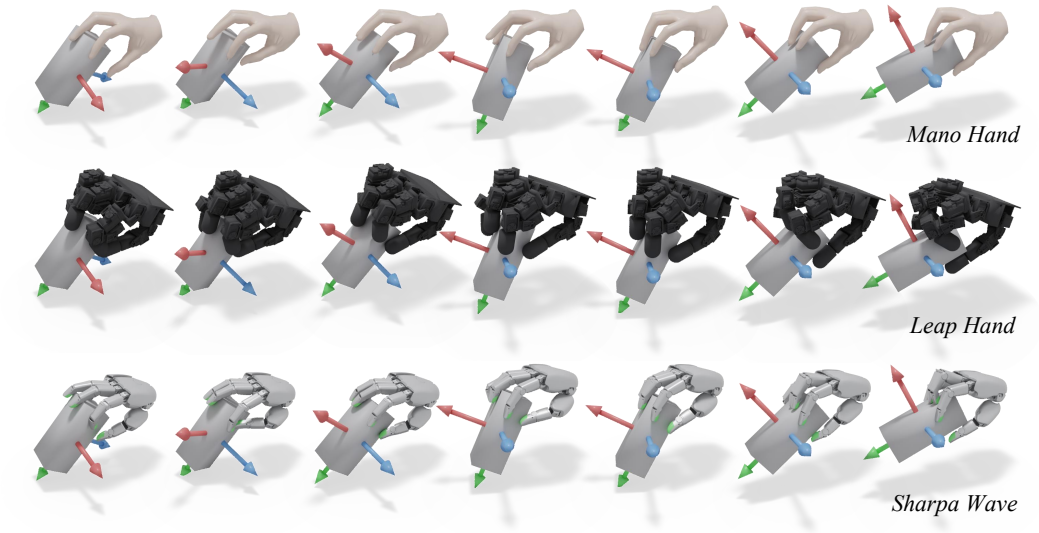


Fig. 9: **Hand–object reference motion visualization.** Retargeted motion clips for (a) `Cylinder` and (b) `Cuboid`. Each clip shows the source MANO hand and the corresponding retargeted LeapHand and SharpaWave sequences. Coordinate axes indicate the object 6-D pose.

C.7 Domain Randomization

We randomize hand and object dynamics, external perturbations, sensing noise, and observation latency to tolerate the sim-to-real gap. Each parameter is sampled independently at the start of every episode and held fixed throughout. The implementation of random force perturbation follows VisualDexterity [28]. The full set of ranges is listed in Tab. 9.

C.8 Random Action Masking Details

Random action masking is the strong action-space regularizer introduced in Sec. 3.1. It prevents the policy from overfitting to the perfectly synchronized actuation of simulation, which is unrealistic on hardware where actuator compliance, backlash, and PD response vary across DoFs.

Tab. 9: **Domain randomization ranges.**

Group	Parameter	Operation	Range
Hand dynamics	body mass	scaling	$\mathcal{U}[0.9,\ 1.2]$
	shape friction	absolute	$\mathcal{U}[1.0,\ 4.0]$
	DoF stiffness K_p	scaling	$\mathcal{U}[0.8,\ 1.2]$
	DoF damping K_d	scaling	$\mathcal{U}[0.8,\ 1.2]$
Object physics	body mass	scaling	$\mathcal{U}[0.5,\ 2.0]$
	surface friction	absolute	$\mathcal{U}[0.5,\ 4.0]$
	rolling / torsional friction	absolute	$\mathcal{U}[0,\ 0.05]$
	restitution	additive	$\mathcal{U}[0,\ 1.0]$
	mesh scale	scaling	$\mathcal{U}[0.95,\ 1.05]$
External force on object	trigger probability per step	—	$\mathcal{U}[0.01,\ 0.25]$
	force scale	constant	1.0
	exponential decay	constant	0.99
Sensing noise	joint position $\boldsymbol{q}_t$	additive Gaussian	$\sigma = 0.1$
	fingertip position	additive uniform	± 5 mm
	object position	additive uniform	± 5 mm
	object orientation	additive uniform	$\pm 2^\circ$
Observation latency	queue size	constant	2 frames
	queue sampling probability	constant	0.5
Initial state	wrist orientation	additive	$\pm 30^\circ$
	reference frame index	uniform	first 90% of clip

Mechanism At each environment step, with probability $p_{\text{mask}} = 0.15$ *and only when no mask is currently active*, we sample a fresh mask: $n_m = 3$ DoF indices are drawn uniformly without replacement, and a freeze duration $d \sim \text{Uniform}\{1, d_t^{\max}\}$ is drawn (where $d_t^{\max}$ ramps from 1 to 10 following the curriculum schedule in Sec. C.3). For the next d control steps, the executed action $\tilde{\boldsymbol{a}}_t$ on the masked DoFs is overwritten with the previously commanded action, while the unmasked DoFs receive the current policy output:

$$\tilde{\boldsymbol{a}}_t[j] = \begin{cases} \tilde{\boldsymbol{a}}_{t-1}[j] & j \in \mathcal{M}_t \\ \boldsymbol{a}_t[j] & \text{otherwise} \end{cases}.$$

After d steps the mask deactivates, and a new mask can be sampled on the next step. Because the mask refreshes asynchronously across the ~ 62 K parallel environments, training sees a wide spectrum of partially-stale joint commands.

Sim-to-Real Effect Random action masking effectively augments the training distribution with desynchronized, partially stale joint commands. The policy is therefore forced to recover useful contact configurations even when some joints respond late or not at all, which closely matches the dominant failure modes we observe on the real hardware (motor lag, backlash, occasional missed commands on the SharpaWave SDK). Empirically, masking is the single most impactful sim-to-real intervention we tested (Tab. 5 in the main paper; further analysis in Sec. A.4).

C.9 RL Training Hyperparameters and Compute

Tab. 10 provides the full set of network, PPO, and SAPG-specific hyperparameters used to train the controller. All our controllers are trained on 4 NVIDIA RTX 5090 GPUs running 15,600 parallel environments per GPU (62,400 environments in total). Each controller is trained for $\sim 10^{10}$ environment steps in a single RL stage, which takes approximately 1 day on this setup.

Tab. 10: **Hyperparameters of SAPG.**

Hyperparameter	Value
Network	
LSTM hidden units	512
LSTM Layer Normalization	enabled
LSTM sequence length	4
MLP hidden sizes	[512, 1024, 1024, 512, 512]
Activation	ELU
PPO	
Learning rate	2×10^{-4}
LR schedule	adaptive
KL threshold	0.008
Num opt-epochs	4
Minibatch size (per GPU)	31,200
Horizon length	32
Discount (γ)	0.99
GAE λ	0.95
Clip range (ϵ)	0.2
Max grad norm	1.0
Bounds-loss coef.	10^{-4}
Parallel envs (per GPU)	15,600
SAPG	
Num blocks	6
Entropy Bonus Scale	0.005
Off-policy ratio	1.0
Mix ratio	0.5

D Baseline Implementation Details

This section describes how each baseline in Tab. 1 is implemented and what we change relative to the original release. All baselines are deployed on the same hardware platform and share the same teleoperation interface (Sec. B.3).

DexRT [5, 6] We use the open-source dex-retargeting codebase[1] with vector-alignment retargeting. The key parameter is the fingertip scaling factor, set to 1.0 for SharpaWave and 1.2 for LeapHand.

GeoRT [12] We follow the original paper and official implementation[2] to train and deploy the neural retargeter. The fingertip workspace data used for training is collected with our own inference glove to match the operator's hand kinematics at deployment.

DexGen [13] No official implementation is available for DexGen. We re-implement its foundation dexterity controller following the training and deployment recipe described in the original paper. Because the key intermediate step (the AnyGrasp-to-AnyGrasp RL policy) lacks sufficient detail for faithful reproduction, we substitute it with our co-tracking controller to generate the simulation rollouts, matching the data scale reported in the original paper. All subsequent stages (diffusion-based action prior training and deployment) follow the original design.

SimToolReal [42] SimToolReal is not a teleoperation method but an object-centric sim-to-real tool-use policy; we include it as a strong reference for learned dexterous manipulation. We follow the official implementation[3] and re-train on our hardware (Franka FR3 + SharpaWave right hand), adapting the simulation workspace and robot embodiment to match our deployment setup. The three tool categories (hammer, brush, screwdriver) and all other training details follow the original paper. We

[1] https://github.com/dexsuite/dex-retargeting
[2] https://github.com/facebookresearch/GeoRT
[3] https://github.com/tylerlum/simtoolreal

evaluate both the category-specific variant (SimToolReal†, one policy per tool) and the all-categories variant (SimToolReal‡, a single policy across tools), as reported in Tab. 1.

E Autonomous Policy Details

We adopt the Conv-UNet Diffusion Policy of Chi et al. [43] (DDPM noise predictor with 1-D temporal convolutions). The policy is conditioned on RGB observations from one third-person and one wrist-mounted camera, each encoded by a separate frozen DINOv2 ViT-S/14 encoder. At each step the policy observes the last $T_o = 2$ frames and predicts an action chunk of length $T_p = 16$. Architecture and training hyperparameters are summarised in Tab. 11.

Tab. 11: **Diffusion Policy hyperparameters.**

Hyperparameter	Value
Visual encoder	DINOv2 ViT-S/14, pretrained, frozen
Encoder sharing	Separate encoders per camera
Image resolution	240×320; random crop 210×280
Observation horizon T_o	2
Action horizon T_p	16
Conv-UNet channels	$[512,\ 1024,\ 2048]$
Diffusion embedding dim	128
Diffusion steps (train)	100
Diffusion steps (inference)	16 (DDIM)
Optimizer	AdamW ($\beta_1 = 0.95$, $\beta_2 = 0.999$, wd 10^{-6})
Learning rate	10^{-4}, cosine schedule, 500-step warmup
Batch size	16
Training epochs	300