RoboEdit: Turning Human Manipulation Videos into Scalable Robot Experience Supplementary Material

Yaowei Guo[1], Zeng Tao[1], Yuxin Jiang[1], Yunuo Chen[1], Zhiyang Dou[2], Yuxiang Ma[2], Yin Yang[3], Demetri Terzopoulos[1†], Ying Jiang[1†], Chenfanfu Jiang[1†]

[1]University of California, Los Angeles
[2]Massachusetts Institute of Technology
[3]University of Utah

A RoboEdit-ADC Details

A.1 Camera Alignment and Depth Regularization

Camera Alignment. VGGT predicts camera-from-world extrinsics $E_t = [R_t^{\mathrm{cw}} \mid t_t^{\mathrm{cw}}]$, camera intrinsics K_t, and depth maps D_t. We convert each extrinsic into camera center $c_t = -R_t^{\mathrm{cw}\top} t_t^{\mathrm{cw}}$ and orientation $R_t^{\mathrm{wc}} = R_t^{\mathrm{cw}\top}$. Because the VGGT reconstruction has an arbitrary scale and world frame, we align it to the hand-object rendering frame with a global similarity transform (s, R_a, t_a). We estimate this transform using standard closed-form SVD alignment of sparse corresponding camera centers between the VGGT and rendering coordinate systems. The aligned camera and depth are $c'_t = sR_a c_t + t_a$, $R'_t = R_a R_t^{\mathrm{wc}}$, and $\widetilde{D}_t = sD_t$, yielding $C_t = (K_t, c'_t, R'_t)$.

Depth Regularization. For depth regularization, we use the aligned depth map to correct the scale and depth of the monocular HaMeR hand reconstruction before retargeting. We select stable wrist and finger-base landmarks from MANO (Romero, Tzionas, and Black 2017) as palm anchors and denote their set by $\mathcal{A}$. For anchor j, we project its HaMeR camera-space position $H_{t,j}$ with K_t, map the resulting pixel $\mathbf{u}_{t,j} = (u_{t,j}, v_{t,j})$ to the resolution of $\widetilde{D}_t$, and take the median positive, finite depth $d_{t,j}$ within a 5×5 neighborhood. We then back-project this observation into a metric 3D anchor:

$$p_{t,j}^{D} = d_{t,j} K_t^{-1} \left[u_{t,j}, v_{t,j}, 1\right]^{\top}.$$

Invalid samples are excluded. From the valid set $\mathcal{A}_t$, we compute $\bar{z}_t^D = \mathrm{median}_{j\in\mathcal{A}_t}(p_{t,j}^D)_z$ and $\bar{z}_t^H = \mathrm{median}_{j\in\mathcal{A}_t}(H_{t,j})_z$, giving $\alpha_t = \bar{z}_t^D / \bar{z}_t^H$. We rescale the complete camera-space hand about the camera center as $\widetilde{H}_t = \alpha_t H_t$, applying the same factor to its global translation while leaving the MANO pose, shape, and relative articulation unchanged. This camera-centered isotropic scaling aligns the metric palm depth while preserving the original image projection.

A.2 Physics-guided Refinement

Refinement Objective. For each retargeted trajectory, we initialize from the kinematic trajectory $q_{1:T}^{e,\mathrm{IK}}$ and optimize the complete temporal window under its kinematics, joint limits, palm geometry, and fingertip correspondences. We retain only trajectories that respect robot joint limits and maintain consistent hand-object contact over time. The main paper defines $\mathcal{L}_{\mathrm{ret}}$; here we expand its four terms. Let $[x]_+ = \max(x, 0)$.

$$\mathcal{L}_{\mathrm{track}} = \sum_{t=1}^{T} \left\| q_t^e - q_t^{e,\mathrm{IK}} \right\|_2^2,$$

$$\mathcal{L}_{\mathrm{geo}} = \sum_{t=1}^{T} \sum_{c\in\mathcal{P}_t} [-d_{t,c}]_+^2 + \beta_{\mathrm{geo}} \sum_{t=1}^{T} \sum_{u\in\mathcal{V}_t} [\delta - s_{t,u}]_+^2,$$

$$\mathcal{L}_{\mathrm{contact}} = \sum_{t=1}^{T} \sum_{k\in\mathcal{C}_t} w_{t,k} \left\| p_k^e(q_t^e) - a_{t,k} \right\|_2^2,$$

$$\mathcal{L}_{\mathrm{temp}} = \sum_{t=2}^{T} \left\| q_t^e - q_{t-1}^e \right\|_2^2.$$

Geometry Loss. The first term of $\mathcal{L}_{\mathrm{geo}}$ detects coarse penetration with the simplified MuJoCo collision geometry: $\mathcal{P}_t$ contains collision pairs with signed distance $d_{t,c}$, where $d_{t,c} < 0$ indicates penetration. The second term checks the robot and object visual meshes to capture fine-scale penetration. Here, $\mathcal{V}_t$ contains samples from the robot visual mesh near the object, and $s_{t,u}$ is the signed distance from sample u to the object surface. This term penalizes both negative $s_{t,u}$, which indicates penetration, and positive distances smaller than the clearance threshold $\delta = 0.005\,\mathrm{m}$. The weight β_{geo} balances this fine-scale visual-mesh penalty with the MuJoCo collision penalty.

Contact Loss. We construct the valid-contact set $\mathcal{C}_t$ from the reconstructed human fingertips. At each frame, fingertip k is transformed into the object coordinate frame, and $\rho_{t,k}$ is its Euclidean distance to the nearest point on the reconstructed object surface. Contact starts when $\rho_{t,k} \le \tau_k$. We select $\tau_k \in [0.02\,\mathrm{m}, 0.06\,\mathrm{m}]$ according to the target embodiment and contact geometry. Once active, a contact remains active until $\rho_{t,k} > \tau_k + 0.01\,\mathrm{m}$ where we use a $1\,\mathrm{cm}$ wider exit threshold to prevent contact labels from toggling when the measured distance fluctuates near τ_k. We discard contact

†Corresponding authors.

segments shorter than 10 consecutive frames. For each $k \in \mathcal{C}_t$, $a_{t,k}$ is that nearest object-surface point, $w_{t,k} \geq 0$ is the contact-confidence weight obtained from the temporally filtered contact segment, and $p_k^e(q_t^e)$ is the corresponding robot-fingertip position.

B 3D Robot-State Decoder Details

Decoder Architecture. A shared ResNet-34 (He et al. 2016) feature pyramid extracts local and global features. For each robot hand, a palm-anchor head predicts heatmaps and visibility for eight predefined 2D palm anchors, a spatial fingertip heatmap head localizes up to five fingertips, and an MLP state head predicts palm-frame 3D fingertip coordinates and normalized joint configurations. A shared camera head predicts intrinsics. SQPnP (Terzakis and Lourakis 2020) recovers the camera-space wrist pose from the predicted 2D palm anchors and predefined 3D palm geometry. A temporal Transformer refines the framewise states over all 81 frames, and embodiment-specific forward kinematics produces the complete camera-space robot-hand trajectory.

Loss Functions. The decoder is trained with three objectives:

$$\begin{aligned}\mathcal{L}_{\text{anchor}} &= \lambda^a_{\text{hm}}\mathcal{L}^a_{\text{hm}} + \lambda^a_{\text{uv}}\mathcal{L}^a_{\text{uv}} + \lambda^a_{\text{vis}}\mathcal{L}^a_{\text{vis}},\\ \mathcal{L}_{\text{spa}} &= \lambda_{\text{2D}}\mathcal{L}_{\text{2D}} + \lambda^s_{\text{palm}}\mathcal{L}_{\text{palm}} + \lambda_{\text{cam}}\mathcal{L}_{\text{cam}}\\ &\quad +\lambda^s_{\text{qpos}}\mathcal{L}_{\text{qpos}} + \lambda_K\mathcal{L}_K,\\ \mathcal{L}_{\text{tmp}} &= \lambda^t_{\text{palm}}\mathcal{L}_{\text{palm}} + \lambda^t_{\text{qpos}}\mathcal{L}_{\text{qpos}} + \lambda_{\Delta\text{palm}}\mathcal{L}_{\Delta\text{palm}}\\ &\quad +\lambda_{\Delta\text{qpos}}\mathcal{L}_{\Delta\text{qpos}} + \lambda_{\text{res}}\mathcal{L}_{\text{res}}.\end{aligned}$$

Here, each λ is a scalar weight for its supervision term; a, s, and t distinguish anchor, spatial, and temporal supervision, respectively. The individual terms are defined as follows:
Anchor Heatmap Loss: $\mathcal{L}^a_{\text{hm}}$ uses a heatmap-distribution loss to supervise the rigid palm-anchor heatmaps.
Anchor Coordinate Loss: $\mathcal{L}^a_{\text{uv}}$ uses Smooth-L1 loss on the 2D image coordinates of the palm anchors.
Anchor Visibility Loss: $\mathcal{L}^a_{\text{vis}}$ uses binary cross-entropy to supervise anchor visibility.
2D Fingertip Loss: $\mathcal{L}_{\text{2D}}$ combines a heatmap-distribution loss with Smooth-L1 loss on the 2D fingertip coordinates.
3D Geometry Losses: $\mathcal{L}_{\text{palm}}$ and $\mathcal{L}_{\text{cam}}$ use Smooth-L1 losses on 3D fingertip coordinates in the palm and camera frames, respectively.
Joint-State Loss: $\mathcal{L}_{\text{qpos}}$ uses Smooth-L1 loss on normalized robot joint states.
Camera-Intrinsics Loss: $\mathcal{L}_K$ uses Smooth-L1 loss on the predicted camera intrinsics.
Temporal Motion Losses: $\mathcal{L}_{\Delta\text{palm}}$ and $\mathcal{L}_{\Delta\text{qpos}}$ use Smooth-L1 losses to match adjacent-frame fingertip and joint-state motion.
Temporal Residual Loss: $\mathcal{L}_{\text{res}}$ applies an L2 penalty to the temporal corrections.

Unavailable fingertips, joints, and frames are excluded using validity masks. $\mathcal{L}_{\text{anchor}}$, $\mathcal{L}_{\text{spa}}$, and $\mathcal{L}_{\text{tmp}}$ are the weighted combinations shown above.

C RoboEdit-14M Statistics

Tables 1 and 2 summarize RoboEdit-14M by source and target embodiment, while Fig. 2 shows representative synthetic human/robot pairs.

Source dataset	Paired clips	Paired frames	Duration (h)
DexYCB	67,200	5,443,200	50.40
GigaHands	11,715	948,915	8.79
H2O	7,955	644,355	5.97
HOT3D	35,591	2,882,871	26.69
TACO	52,086	4,218,966	39.06
Total	174,547	14,138,307	130.91

Table 1: RoboEdit-14M source distribution, with duration computed at 30 FPS.

Embodiment	Real	Synthetic	Total
Ability	24,197	4,839	29,036
Allegro	24,197	4,839	29,036
Unitree Dex3	12,237	2,447	14,684
Inspire	24,197	4,839	29,036
Panda gripper	12,237	2,447	14,684
SCHUNK SVH	24,197	4,838	29,035
XHand	24,197	4,839	29,036
Total	145,459	29,088	174,547

Table 2: RoboEdit-14M distribution by target embodiment.

D Training Details

Computing Infrastructure. Experiments were conducted on Ubuntu 22.04.5 workstations equipped with two AMD EPYC 9354 CPUs, 1.5 TiB of host memory, and eight NVIDIA H100 NVL GPUs, each with 94 GiB of memory.

Backbone and Cross-Embodiment Adaptation. We first train the video-editing backbone for approximately 10K steps, updating 283.4M parameters with AdamW at a learning rate of 5×10^{-5} and an effective batch size of 12 across eight GPUs. We then freeze the backbone and jointly train LoRA adapters on selected attention and feed-forward projections together with residual bottleneck adapters in the final ten transformer blocks, using an effective batch size of 16 across four GPUs. The LoRA and residual-adapter branches contain 2.5M and 7.9M trainable parameters, respectively.

3D Robot-State Decoder. We first train the palm detector and framewise spatial estimator on rendered and composited robot frames. We then freeze both components and train temporal refinement on complete 81-frame sequences. We train the spatial and temporal stages for approximately 8.9K and 1.5K optimization steps, respectively; both stages use AdamW with a learning rate of 2×10^{-5}.

Qwen-Image-Edit. We fine-tune Qwen-Image-Edit for approximately 4.9K optimization steps on aligned image pairs sampled from RoboEdit-14M. At inference, it generates robot references at indices $\{0, 10, \ldots, 80\}$, after which RoboEdit-Trans produces an 81-frame video.

Randomness Control. For reproducibility, we fix the random seeds used for data sampling, decoder training, and

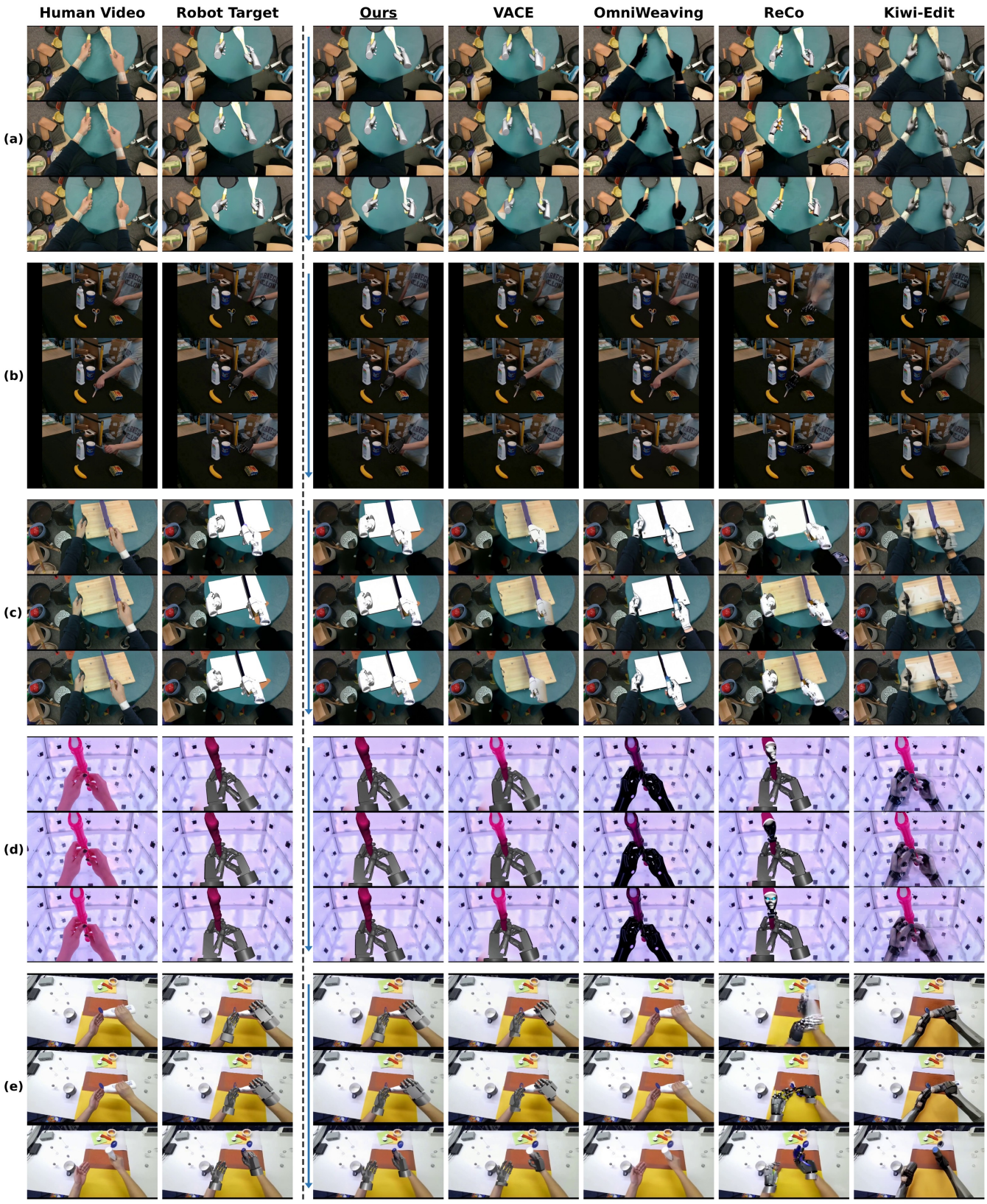

Figure 1: Additional qualitative comparisons on five human-object interactions.

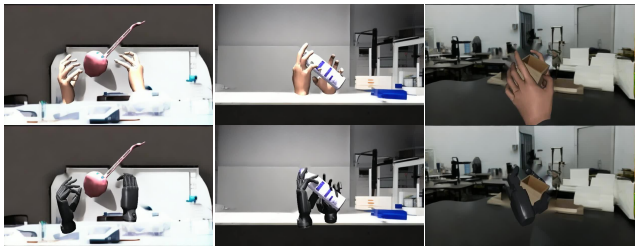

Figure 2: Synthetic paired-video examples. Each column shows an aligned synthetic human frame and its robot target under the same scene, object state, and camera view.

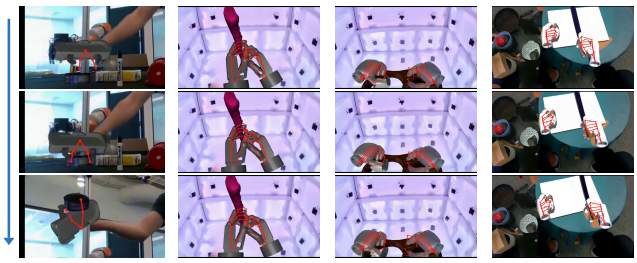

Figure 3: Additional 3D Robot-State Decoder results on RoboEdit-Trans edited videos. Red overlays denote predicted camera-space hand states.

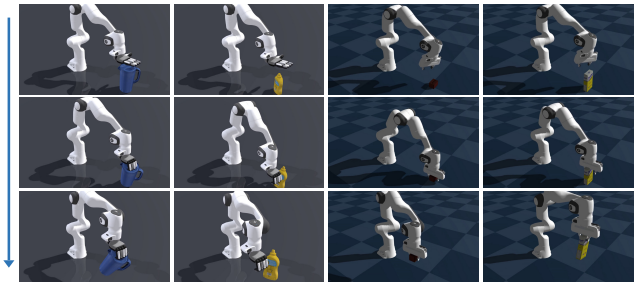

Figure 4: Simulation rollouts of the trajectory-conditioned controller across four YCB-object manipulation tasks.

model inference. RoboEdit-Trans data sampling initializes its NumPy generator with 20260531 plus the epoch index. Decoder training initializes the Python, NumPy, and PyTorch generators with 20260705 plus the distributed rank. RoboEdit-Trans, Qwen-Image-Edit, and baseline inference use deterministic per-sample PyTorch seeds derived from 42, 20260629, and 20260624, respectively.

E Trajectory-Conditioned Control and Real-Robot Deployment

We evaluate whether the robot-hand trajectories recovered by the 3D Robot-State Decoder are physically executable and can supervise downstream control. We first train a trajectory-conditioned controller in simulation and then execute simulation-validated trajectories on a physical Franka Panda.

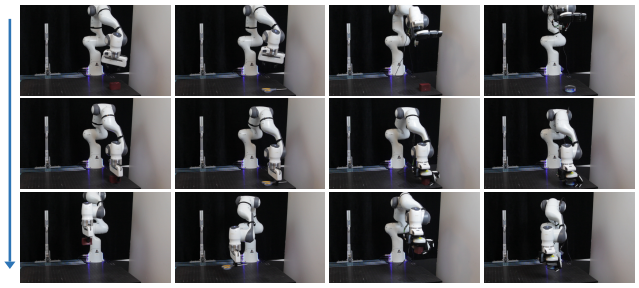

Figure 5: Additional real-robot deployment results across four YCB-object tasks.

Trajectory-conditioned residual control. For each interaction, the 3D Robot-State Decoder provides a decoded robot-hand trajectory $\tau^h = \{h_t^\star\}_{t=1}^T$, which serves as the control reference; $h_t^\star$ denotes the reference palm or end-effector state. We represent the trajectory relative to its first frame and resample it to the control horizon. Given the current task state s_t and the robot-hand tracking error $h_t^\star - p_t$, the residual policy π_ϕ predicts a bounded correction to a nominal joint command computed by inverse kinematics (IK):

$$a_t = \pi_\phi(s_t, h_t^\star - p_t)\,,$$
$$q_t^{\mathrm{cmd}} = \mathrm{clip}\big(q_t^{\mathrm{IK}} + \rho_{\mathrm{act}} a_t, q_{\min}, q_{\max}\big)\,.$$

Here, π_ϕ is the residual policy with parameters ϕ, s_t is the current robot and object task state, and p_t is the current palm or end-effector state. q_t^{IK} is the nominal arm configuration computed by IK, and q_t^{cmd} is the resulting command sent to the low-level arm controller; $a_t \in [-1, 1]^7$ is the residual arm action, and $\rho_{\mathrm{act}} = 0.02\,\mathrm{rad}$ is its scale. $q_{\min}$ and $q_{\max}$ denote the Franka arm joint limits. The operator clip enforces these limits elementwise. The low-level controller executes q_t^{cmd}, whose outcome is evaluated through palm tracking and task success.

Control objective. The reward supervises tracking of the decoded robot-hand trajectory while encouraging successful and smooth task execution:

$$r_t = w_{\mathrm{hand}} \exp(-\beta \left\| p_t - h_t^\star \right\|_2) + r_t^{\mathrm{task}} - w_{\mathrm{act}} \left\| a_t \right\|_2^2 .$$

Here, r_t is the total reward; w_{hand} and w_{act} weight trajectory tracking and action regularization, respectively; β controls sensitivity to the Euclidean tracking error; and r_t^{task} contains rewards for grasp maintenance, object-motion consistency, and lift completion. The decoder trajectory therefore conditions both the policy action and its training objective.

Simulation setup. We train the residual policy using PPO (Schulman et al. 2017) in Genesis (Genesis Authors 2024), with 512 parallel environments spanning 18 YCB objects. A single policy is trained across all objects, without per-object fitting. During training, the initial object position and yaw are randomized by up to 4 cm and 23°, respectively.

A rollout succeeds if the object remains grasped and its terminal position is within 8 cm of the demonstrated target. Across the randomized environments, the controller achieves

trajectory-reproduction success rates of 71% for the Panda gripper and 62% for XHand. Representative simulation rollouts are shown in Fig. 4.

Real-robot deployment. We deploy the controller on a 7-DoF Franka Panda, using the decoded robot-hand trajectories as control references for YCB-object manipulation.

F Additional Qualitative Results

Figure 1 extends the main-paper comparison with five additional interaction examples. We again use midpoint non-keyframes at indices 5, 35, and 65. Figure 3 shows additional decoded robot states from edited videos, and Figure 5 presents additional real-robot results on YCB-object manipulation with both end effectors.

Generative AI Disclosure. Generative AI tools were used only for language editing. All manuscript text, code, figures, references, and experimental outputs were reviewed and verified by the authors.

References

Genesis Authors. 2024. Genesis: A Generative and Universal Physics Engine for Robotics and Beyond.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Romero, J.; Tzionas, D.; and Black, M. J. 2017. Embodied Hands: Modeling and Capturing Hands and Bodies Together. *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)*, 36(6).

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347.

Terzakis, G.; and Lourakis, M. 2020. A Consistently Fast and Globally Optimal Solution to the Perspective-n-Point Problem. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I*, 478–494. Berlin, Heidelberg: Springer-Verlag. ISBN 978-3-030-58451-1.