POINTWAM: 3D WORLD ACTION MODELING FOR DEXTEROUS ROBOTIC MANIPULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

World action models jointly learn to forecast world dynamics and predict robot actions, such that the learned internal world dynamics guide accurate actions. Existing approaches typically represent the world as RGB frames or latent counterparts while predicting actions as end-effector poses or joint angles, but they often struggle to capture the 3D spatial structure and contact geometry central to dexterous manipulation. We introduce Point World Action Model (PointWAM), a 3D world action model that decomposes the world into a scene (*i.e.*, environment) and hands (*i.e.*, actor), and jointly forecasts both as 3D point trajectories within a shared space-time coordinate frame. This explicit, disentangled representation enables effective pre-training on large-scale human demonstration videos without requiring any task-specific object or keypoint selection. Given a colored point cloud and a language instruction, PointWAM predicts how the scene and hands co-evolve in 3D space over time, then retargets the forecast hand motion to robot actions. On ten DexJoCo tasks, PointWAM surpasses the prior state-of-the-art average multi-task success rate by 11.7 percentage points, with scene-trajectory supervision contributing 10.9 points and human-data pre-training 56.9 points. On a real robot, PointWAM also outperforms GR00T N1.6 and $\pi_{0.5}$ on both single-arm and bimanual tasks, by up to 25 percentage points.

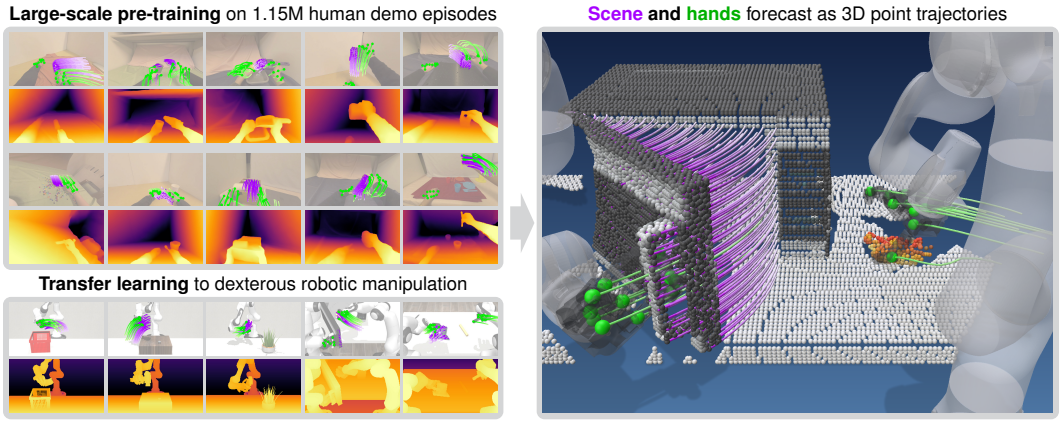


Figure 1: **PointWAM forecasts the scene and hands as 3D point trajectories.** PointWAM pre-trains on human demonstration videos and fine-tunes on robot demonstrations, both expressed as 3D point trajectories (left; depth below RGB). Given a language instruction, its scene and hand forecasts share one space-time coordinate frame (right), and it retargets the hand trajectories to robot actions.

1 INTRODUCTION

Dexterous robotic manipulation requires multiple fingers to precisely coordinate their 3D motion relative to object surfaces (Salisbury & Craig, 1982; Okamura et al., 2000), enabling robots to perform a wide range of tasks from grasping objects of varied shapes to tool use (Bicchi, 2000; An et al., 2026). Conventional Vision-Language-Action models (VLAs) (Black et al., 2025; NVIDIA et al.,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2025) map observations to motor commands without explicitly forecasting how the scene evolves. World Action Models (WAMs) have recently emerged as a promising alternative by treating world and action modeling as deeply intertwined: they jointly learn to forecast how the world will change and what actions cause those changes (Li et al., 2025a).

Many existing WAMs build on video generation backbones and represent the world as RGB frames (Du et al., 2023) or their latent counterparts (Li et al., 2025a; Liao et al., 2026). These representations are limited in their ability to capture key aspects of physical interaction, including 3D spatial structure and causal relations between the actor and the environment. Point-based policies (Haldar & Pinto, 2025; Kim et al., 2026a) address the 3D action prediction problem by forecasting trajectories of 3D points, but do not model how the environment evolves. They also rely on humans or external perception models to select task-relevant objects for their scene inputs. Conversely, action-conditioned 3D world models forecast dense 3D scene trajectories, yet treat robot actions as inputs rather than quantities to be predicted (Huang et al., 2026).

In this work, we introduce *Point World Action Model* (PointWAM), a 3D world action model that decomposes the world into a scene (*i.e.*, environment) and hands (*i.e.*, actor), and represents both as three-dimensional point trajectories within a shared space-time coordinate frame (Fig. 1). This explicit, disentangled world representation enables effective learning of cause-and-effect relations between the environment and the actor by modeling how their points co-evolve in 3D space over time. Given a colored point cloud and a language instruction, PointWAM forecasts trajectories for every observed scene point and hand keypoint in a common coordinate frame. Its action retargeter then converts the forecast hand trajectories into the corresponding robot actions.

The proposed approach offers three additional advantages. First, modeling both the scene and hands in a shared 3D space enables effective pre-training on human demonstration videos from EgoDex (Hoque et al., 2026) and VITRA (Li et al., 2026b). Second, no task-specific objects or scene keypoints need to be selected in advance, so scene forecasting is not restricted to a predefined set of target objects. Third, by decoupling hand-point trajectories from the scene, the forecast hand motion is amenable to effective retargeting onto robot hands for action prediction.

Our main contributions can be summarized as follows.

- We introduce PointWAM, which jointly forecasts the 3D trajectories of scene points and hand keypoints in a shared space-time coordinate frame, unifying world and action modeling.
- We demonstrate that 3D world modeling benefits dexterous manipulation. Scene-trajectory supervision improves average DexJoCo success by 10.9 percentage points over the same model without it, and our 3D encoder outperforms lifted image features by 5.4 percentage points.
- We show that pre-training on human demonstrations from EgoDex and VITRA improves average DexJoCo success by 56.9 percentage points over training from scratch. With this pre-training, PointWAM surpasses the prior best multi-task success rate on DexJoCo by 11.7 percentage points and outperforms robot-pre-trained VLAs on a real robot by up to 25 percentage points.

2 Related Work

We discuss the closest work here and provide a broader review in Appendix A.

World action models (WAMs). WAMs unify predictive world modeling with action prediction; unlike conventional VLAs, which map a visual observation straight to a motor command, a WAM is trained to forecast how the world will change and predict what actions the robot must perform (Du et al., 2023; Hu et al., 2025; Liao et al., 2026; Kim et al., 2026d; Li et al., 2025a). Existing WAMs typically forecast future visual representations in the latent space of pretrained video models while predicting robot actions as end-effector poses. These models must therefore learn how changes in latent visual features correspond to robot motion, rather than explicitly modeling both scene and hand motion as 3D trajectories. Closest to ours, EgoWAM (Li et al., 2026a), PointWorld (Huang et al., 2026), and PointZero (Duisterhof et al., 2026) forecast dense 3D scene motion. EgoWAM jointly learns scene forecasting and action prediction but maps human hand motion to the actions of parallel-jaw grippers, a 6-DoF pose and a gripper state per hand, which lacks individual finger motions. Pri4R (Kim et al., 2026c) likewise targets grippers, adding a training-only 3D point-trajectory

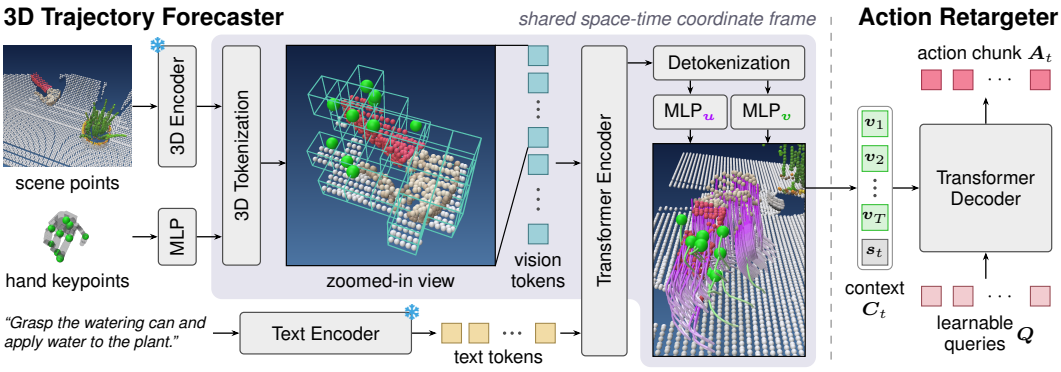


Figure 2: **PointWAM architecture.** The 3D trajectory forecaster (left) encodes scene points, hand keypoints, and a language instruction, and jointly forecasts scene $\boldsymbol{u}$ and hand $\boldsymbol{v}$ trajectories in a shared space-time coordinate frame. The action retargeter (right) maps $\boldsymbol{v}$ and the robot state $\boldsymbol{s}_t$ to an action chunk $\boldsymbol{A}_t$. MLP denotes a multilayer perceptron, and ❄ marks frozen modules.

head to a VLA fine-tuned on robot demonstrations. PointWorld forecasts scene motion with a 3D point-cloud network (Wu et al., 2024) conditioned on future robot trajectories, so it needs a separate planner that selects them with a task cost. PointZero pre-trains a similar world model without robot data and, like PointWorld, predicts no actions. PointWAM instead pre-trains on human demonstration videos for dexterous hands, jointly forecasts scene and hand trajectories in a shared space-time coordinate frame, and retargets the hand trajectories to robot actions without a separate planner.

Point trajectories from human demonstration videos. Human demonstration videos show hand and scene motion from which trajectory supervision can be extracted without recorded robot actions. Point-based policies can learn to forecast hand keypoint trajectories from these videos and retarget the forecasts to robot actions. Point Policy (Haldar & Pinto, 2025) and Dexterous Point Policy (Kim et al., 2026a), however, rely on humans or external perception models to select the objects in the policy's scene input, object keypoints specified by a human for Point Policy and object point clouds segmented by external models for Dexterous Point Policy. Neither performs explicit world modeling through scene-motion forecasting. 3PoinTr (Hung et al., 2026) does forecast dense 3D scene trajectories from human demonstration videos. Extracted video tracks supervise its forecasts, and a separate policy trained on robot demonstrations predicts actions from the forecast tracks. PointWAM instead forecasts trajectories for every observed scene point and hand keypoint with one backbone, and a retargeter converts the hand trajectories into robot actions without a separate policy.

3 Point World Action Model

We first formulate the task and outline our approach (Section 3.1). We then describe the trajectory forecaster and action retargeter shown in Fig. 2 (Sections 3.2 and 3.3), followed by trajectory supervision and two-stage training on human and robot demonstrations (Section 3.4).

3.1 Problem Statement and Overview

Given an instruction ℓ, a scene point cloud $\mathcal{P}_t$, and the current robot state $\boldsymbol{s}_t$, the goal is to predict an action chunk $\boldsymbol{A}_t \in \mathbb{R}^{T \times d_a}$ that carries out ℓ, where T is the horizon and d_a the action dimension. The point cloud $\mathcal{P}_t = \{(\boldsymbol{p}_t^{(i)}, \boldsymbol{c}_t^{(i)})\}_{i=1}^{N}$ contains 3D positions and colors from RGBD frames, while $\boldsymbol{s}_t$ records the end-effector pose and finger joint angles of each arm.

We approach this problem by jointly forecasting scene and hand trajectories and retargeting the hand trajectories to robot actions. To establish a common representation for both human and robot hands, we use semantically corresponding 3D keypoints, as in point-based policies (Haldar & Pinto, 2025; Kim et al., 2026a). Specifically, we represent the hands with K keypoints, ten per hand, as $\mathcal{Q}_t = \{\boldsymbol{q}_t^{(k)}\}_{k=1}^{K}$ in the scene's coordinate frame. Their positions come from robot states or tracked

human hands (Section 3.4). We express both trajectories as displacements from time t:

$$\boldsymbol{u}_{t,\tau}^{(i)} = \boldsymbol{p}_{t+\tau}^{(i)} - \boldsymbol{p}_t^{(i)}, \qquad \boldsymbol{v}_{t,\tau}^{(k)} = \boldsymbol{q}_{t+\tau}^{(k)} - \boldsymbol{q}_t^{(k)}, \qquad \tau = 1, \dots, T. \tag{1}$$

Scene trajectories supervise world modeling, and hand trajectories serve as an intermediate action representation. A trajectory forecaster $f(\cdot)$ predicts both from the scene, hand keypoints, and instruction, and an action retargeter $g(\cdot)$ maps forecast hand trajectories and robot state to actions:

$$(\mathbf{U}_t, \mathbf{V}_t) = f(\mathcal{P}_t, \mathcal{Q}_t, \ell), \qquad \boldsymbol{A}_t = g(\mathbf{V}_t, \boldsymbol{s}_t). \tag{2}$$

Here $\mathbf{U}_t \in \mathbb{R}^{N\times T\times 3}$ and $\mathbf{V}_t \in \mathbb{R}^{K\times T\times 3}$ stack the forecast scene and hand displacements, which we standardize per horizon step for training (Appendix B.5).

3.2 3D Trajectory Forecaster

Scene and hand features. We first encode the observed scene point cloud and hand keypoints. A frozen Mosaic3D encoder (Lee et al., 2025), pre-trained to align 3D point features with text, extracts a feature vector for each scene point that captures local geometry and semantics. For each hand keypoint, a multilayer perceptron (MLP) maps a one-hot encoding of its index, concatenated with a hand ID, to a feature of the same dimension as the scene features.

Tokenization. We place scene and hand features on a shared 1 cm voxel grid using their 3D coordinates, then aggregate neighboring features with a sparse-convolutional stem. We retain these stem features as $\boldsymbol{S}_t$ for decoding. Following Volt (Yilmaz et al., 2026), we divide this grid into non-overlapping patches of K_v^3 voxels. For each nonempty patch, we concatenate its voxel features in a fixed spatial order and linearly project them into one vision token $\boldsymbol{z}_j^v \in \mathbb{R}^d$, $j = 1, \dots, M$. A saved mapping $\mathcal{I}_t$ records the patch and voxel of each point. We linearly project the L features of the instruction ℓ from a frozen text encoder (Li et al., 2025b) into instruction tokens $\boldsymbol{z}_1^\ell, \dots, \boldsymbol{z}_L^\ell$.

Multimodal fusion and trajectory forecasting. We concatenate the instruction and vision tokens and process them with a transformer encoder backbone using global self-attention (Vaswani et al., 2017) and axial 3D rotary position embeddings (Su et al., 2024; Yilmaz et al., 2026), allowing each vision token to attend to all instruction and vision tokens:

$$[\boldsymbol{Z}_t^\ell; \boldsymbol{Z}_t^v] = \mathrm{TransformerEncoder}\big([\boldsymbol{z}_1^\ell, \dots, \boldsymbol{z}_L^\ell, \boldsymbol{z}_1^v, \dots, \boldsymbol{z}_M^v]^\top\big). \tag{3}$$

We then recover per-point features from the output vision tokens and combine them with the stem features $\boldsymbol{S}_t$ through a residual MLP over their concatenation $[\cdot;\cdot]$, which retains local detail:

$$\boldsymbol{F}_t = \boldsymbol{F}_t' + \mathrm{MLP}([\boldsymbol{F}_t'; \boldsymbol{S}_t]), \qquad \text{where} \quad \boldsymbol{F}_t' = \mathrm{Detokenizer}(\boldsymbol{Z}_t^v; \mathcal{I}_t). \tag{4}$$

Here Detokenizer uses a linear layer to expand each token into K_v^3 voxel features, then selects the feature for each point through $\mathcal{I}_t$. We then split $\boldsymbol{F}_t$ into scene and hand features, $\boldsymbol{F}_t^{\text{scene}}$ and $\boldsymbol{F}_t^{\text{hand}}$, and forecast their trajectories with two MLP heads:

$$\mathbf{U}_t = \mathrm{MLP}_{\boldsymbol{u}}(\boldsymbol{F}_t^{\text{scene}}), \qquad \mathbf{V}_t = \mathrm{MLP}_{\boldsymbol{v}}(\boldsymbol{F}_t^{\text{hand}}). \tag{5}$$

The scene head forecasts a trajectory for every point in $\mathcal{P}_t$, so it requires no task-specific selection of target objects or keypoints. More importantly, joint supervision encourages the shared backbone to learn how scene and hand points co-evolve in 3D over time.

3.3 Action Retargeter

The action retargeter maps forecast hand trajectories and the robot state to T robot actions; scene forecasts supervise the shared backbone but do not enter the retargeter. To construct its input, we encode the hand trajectories at each future step, add a step embedding, and append a state embedding to form a context $\boldsymbol{C}_t$ for action prediction. A transformer decoder (Vaswani et al., 2017) then uses this context and learnable action queries to predict the action chunk $\boldsymbol{A}_t$:

$$\begin{aligned} \boldsymbol{C}_t &= \big[\mathrm{MLP}_{\text{disp}}(\mathbf{V}_{t,1}) + \boldsymbol{b}_1,\ \dots,\ \mathrm{MLP}_{\text{disp}}(\mathbf{V}_{t,T}) + \boldsymbol{b}_T,\ \mathrm{MLP}_{\text{state}}(\boldsymbol{s}_t)\big]^\top \in \mathbb{R}^{(T+1)\times d_r}, \\ \boldsymbol{A}_t &= \mathrm{TransformerDecoder}(\boldsymbol{Q}, \boldsymbol{C}_t) \in \mathbb{R}^{T\times d_a}. \end{aligned} \tag{6}$$

Here $\mathbf{V}_{t,\tau} \in \mathbb{R}^{K\times 3}$ contains the hand displacements at future step τ, and $\boldsymbol{b}_\tau$ is a learned step embedding; both MLPs output d_r-dimensional features. Each decoder block applies self-attention among the learnable action queries $\boldsymbol{Q} \in \mathbb{R}^{T\times d_r}$, cross-attention to $\boldsymbol{C}_t$, and a feed-forward network. A final linear layer maps the features to T actions of end-effector poses and finger joint targets.

3.4 Learning from Human and Robot Demonstrations

We train the trajectory forecaster on human and robot demonstrations and the action retargeter on robot demonstrations alone, since only they record the end-effector poses and finger joint commands that the action retargeter learns to predict (Fig. 1, left).

Human demonstrations. Human demonstration videos lack recorded states, so we estimate both trajectories from the videos themselves. Hand trajectories come from the 3D hand poses annotated in EgoDex (Hoque et al., 2026) and reconstructed by VITRA (Li et al., 2026b), mapped to the robot hand's ten keypoints. Scene trajectories come from dense 2D point tracks of CoTracker3 (Karaev et al., 2025), lifted to 3D with monocular depth from VGGT-Ω (Wang et al., 2026b) scaled to the annotated hands. The resulting 1.15M episodes share the trajectory format of the robot demonstrations, so both supervise the same forecaster; Appendix B.1 details the preprocessing.

Robot demonstrations. Recorded robot states give hand trajectories through forward kinematics. Scene trajectories come from replaying DexJoCo (Wang et al., 2026a) demonstrations, which move each initial point with its object or rigid part, and on the real robot from CoTracker3 (Karaev et al., 2025) tracks lifted with Fast-FoundationStereo (Wen et al., 2026) depth. We process the RoboDojo-Precision demonstrations similarly to DexJoCo, and Appendix B.2 details all three settings.

Training. We first pre-train the backbone and both trajectory heads from scratch on human demonstrations using scene and hand trajectory supervision. On robot demonstrations, we then fine-tune these components and train the retargeter from scratch with joint trajectory and action supervision:

$$\begin{aligned} \mathcal{L}_{\text{traj}} &= \sum_{i,\tau} w_{i,\tau} \left\| \boldsymbol{u}_{t,\tau}^{(i)} - \boldsymbol{u}_{t,\tau}^{(i)*} \right\|_2^2 + \frac{1}{KT} \sum_{k,\tau} \left\| \boldsymbol{v}_{t,\tau}^{(k)} - \boldsymbol{v}_{t,\tau}^{(k)*} \right\|_2^2, \\ \mathcal{L}_{\text{robot}} &= \mathcal{L}_{\text{traj}} + \lambda \left\| \boldsymbol{A}_t - \boldsymbol{A}_t^* \right\|_1. \end{aligned} \tag{7}$$

Here $*$ marks ground truth, and $\lambda = 1$. The scene weights $w_{i,\tau}$ sum to one over points and steps and favor motion without excluding static points (Huang et al., 2026).

4 Experiments

In this section, we describe the experimental setup (Section 4.1) and answer three questions.

(1) Does PointWAM, pre-trained only on human demonstrations, outperform policies pre-trained on robot or video corpora in dexterous and high-precision manipulation? (Section 4.2)

(2) Does human-demonstration pre-training transfer to robot control, and does it scale with the number of human demonstrations and with model size? (Section 4.3)

(3) Does 3D world modeling enable more precise dexterous manipulation than 2D world modeling or a model without it, and does a 3D encoder help beyond lifted 2D features? (Section 4.4)

4.1 Experimental Setup

We pre-train PointWAM on 1.15M human demonstration episodes from EgoDex (Hoque et al., 2026) and VITRA (Li et al., 2026b), which provide scene and hand trajectories but no robot actions, and fine-tune it on the robot demonstrations of each benchmark and of the real robot.

4.1.1 Benchmarks and Protocols

DexJoCo. DexJoCo (Wang et al., 2026a) is a simulated benchmark of eleven dexterous tasks, six single-arm and five bimanual, on Franka Panda arms with 16-DoF Allegro hands. Its tasks require tool use, such as driving a nail with a hammer, and coordination of two hands. We use its object-placement setting, where no policy reported with the benchmark exceeds 5.3% success on bimanual assembly. We exclude bimanual unlock iPad, since 42 of its 100 demonstrations fail when replayed in the released environment; 959 of the other 1,000 demonstrations pass the replay check. We train one policy across the ten tasks (Table 1) and one per task. Each task has one fixed instruction, which tells the multi-task policy which task to perform. Following the official protocol, we run 50 closed-loop episodes per task on each of three evaluation seeds and report the success rate (SR) averaged

Table 1: **DexJoCo success rate (%).** One policy across the ten tasks in the object-placement setting. Subscripts are standard deviations over three seeds, /B marks bimanual tasks, and bold marks the row best. DP-T and $\pi_{0.5}$ results are from Wang et al. (2026a); the others follow the same protocol.

Task	DP-T	$\pi_{0.5}$	GR00T N1.6	Fast-WAM	PointACT	PointWAM
hammer nail	$58.7_{5.0}$	$\mathbf{86.7}_{3.1}$	$82.0_{6.0}$	$43.3_{1.9}$	$16.0_{8.0}$	$85.3_{2.3}$
click mouse	$38.7_{3.1}$	$80.7_{3.1}$	$67.3_{7.6}$	$77.3_{3.4}$	$37.3_{3.1}$	$\mathbf{97.3}_{2.3}$
pick bucket	$55.3_{7.6}$	$83.3_{8.1}$	$88.7_{3.1}$	$81.3_{6.6}$	$74.7_{7.6}$	$\mathbf{98.0}_{2.0}$
pinch tongs	$6.0_{5.3}$	$45.3_{6.1}$	$28.0_{4.0}$	$47.3_{6.6}$	$26.7_{4.2}$	$\mathbf{52.0}_{5.3}$
fold glasses	$11.3_{5.0}$	$42.0_{6.0}$	$52.0_{8.0}$	$62.7_{3.8}$	$20.7_{7.6}$	$\mathbf{74.0}_{5.3}$
water plant	$60.0_{6.9}$	$84.0_{4.0}$	$82.7_{3.1}$	$42.7_{0.9}$	$83.3_{6.4}$	$\mathbf{86.7}_{3.1}$
hanoi /B	$8.0_{2.0}$	$6.0_{0.0}$	$34.0_{5.3}$	$42.0_{5.9}$	$10.0_{2.0}$	$\mathbf{50.0}_{9.2}$
assembly /B	$1.3_{2.3}$	$3.3_{2.3}$	$6.0_{4.0}$	$2.7_{1.9}$	$2.0_{0.0}$	$\mathbf{9.3}_{4.6}$
microwave /B	$42.7_{6.4}$	$39.3_{13.0}$	$78.7_{3.1}$	$\mathbf{88.7}_{4.1}$	$54.0_{8.7}$	$84.0_{4.0}$
photograph /B	$28.0_{6.0}$	$29.3_{1.2}$	$\mathbf{53.3}_{8.1}$	$40.7_{6.2}$	$12.0_{3.5}$	$\mathbf{53.3}_{1.2}$
avg.	$31.0_{1.7}$	$50.0_{1.9}$	$57.3_{1.8}$	$52.9_{1.5}$	$33.7_{1.8}$	$\mathbf{69.0}_{2.0}$

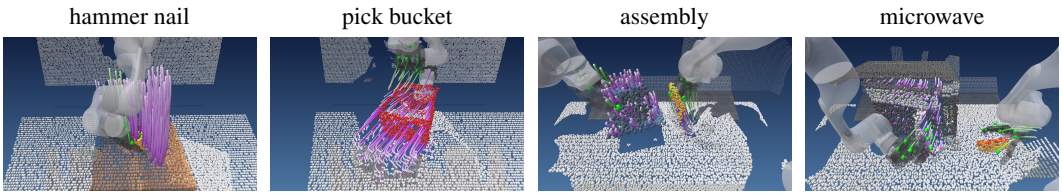


Figure 3: **Predicted 3D point trajectories.** Scene $\boldsymbol{u}$ and hand $\boldsymbol{v}$ trajectories that PointWAM predicts on unseen DexJoCo layouts; scene trajectories are subsampled for visibility.

over the ten tasks, with its standard deviation over seeds. Appendix C.2 details the protocol, and Appendix D reports the results of the single-task policies.

RoboDojo-Precision. Although PointWAM is designed for dexterous hands, its point-trajectory representation is not tied to one embodiment. We therefore test whether it transfers to parallel-jaw grippers, using the Precision tasks of RoboDojo (Chen et al., 2026b), which hinge on precise placement. Two ARX X5 arms perform these eight tasks, such as inserting a key and fastening screws, and each task comes with 100 RGBD demonstrations. We train and evaluate every method on these tasks alone. The tasks are hard; on the benchmark leaderboard, policies trained on all of its tasks rarely complete them (Chen et al., 2026b). We therefore also report its partial-progress score, which separates methods when success is rare. Appendices B.2 and C.3 detail the setup.

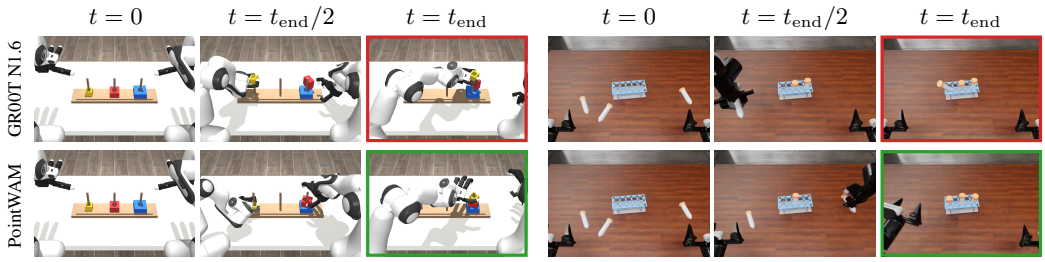
Real robot. To evaluate PointWAM's manipulation performance in the real world, we use an OpenArm humanoid with two 7-DoF arms, two 6-DoF Inspire RH56F1 hands, and a ZED 2i stereo camera, as shown in Fig. 5a. We evaluate two tasks: banana pick-and-place (PnP) and square-peg insertion. Banana PnP evaluates general manipulation ability using a single arm, while square-peg insertion evaluates coordinated dexterous manipulation with both hands. We collect 54 demonstrations for banana PnP and 64 for square-peg insertion, and evaluate each method over 24 trials per task. For banana PnP, half of the trials use the banana seen in the demonstrations (in-domain), and the other half use an unseen box whose color resembles the table (out-of-domain). For square-peg insertion, we also report a progress score, the mean number of completed stages per trial out of three, lifting each of the two parts and inserting the peg.

4.1.2 BASELINES

We take two baselines from DexJoCo (Wang et al., 2026a) with their published results. DP-T is the transformer variant of Diffusion Policy (Chi et al., 2023), trained from scratch on images, and $\pi_{0.5}$ (Black et al., 2025) is a VLA fine-tuned from a policy pre-trained on robot demonstrations and web data. We then add three recent families and evaluate them under the same protocol (Appendices C.2 and C.5). GR00T N1.6 (NVIDIA et al., 2025) represents VLAs whose pre-training

Table 2: **RoboDojo-Precision results.** Each cell reports the score (left) and success rate in % (right); the score measures partial task progress. One policy per method, one seed, 50 episodes per task.

Method	fasten screws	plug in charger	insert tubes	pour balls	play xylophone	deposit coin	insert key	build tower	avg.
GR00T N1.6	0.4 / 0.0	0.0 / 0.0	20.4 / 8.0	0.0 / 0.0	0.0 / 0.0	**9.2 / 4.0**	**7.8** / 0.0	**11.4 / 6.0**	6.2 / 2.3
Fast-WAM	0.0 / 0.0	0.0 / 0.0	0.4 / 0.0	2.0 / 2.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.3 / 0.2
PointACT	0.0 / 0.0	0.0 / 0.0	6.4 / 0.0	0.0 / 0.0	0.0 / 0.0	0.8 / 0.0	2.7 / 0.0	3.2 / 0.0	1.6 / 0.0
PointWAM	**1.2** / 0.0	0.0 / 0.0	**32.4 / 12.0**	**14.0 / 14.0**	**8.0 / 8.0**	6.4 / 2.0	3.0 / 0.0	5.2 / 2.0	**8.8 / 4.8**



(a) DexJoCo (bimanual hanoi) (b) RoboDojo-Precision (insert tubes)

Figure 4: **Qualitative comparison.** Rollouts from one initial state per task, each until success (green border) or failure (red border) at its own t_{end}. PointWAM succeeds on both; GR00T N1.6 times out.

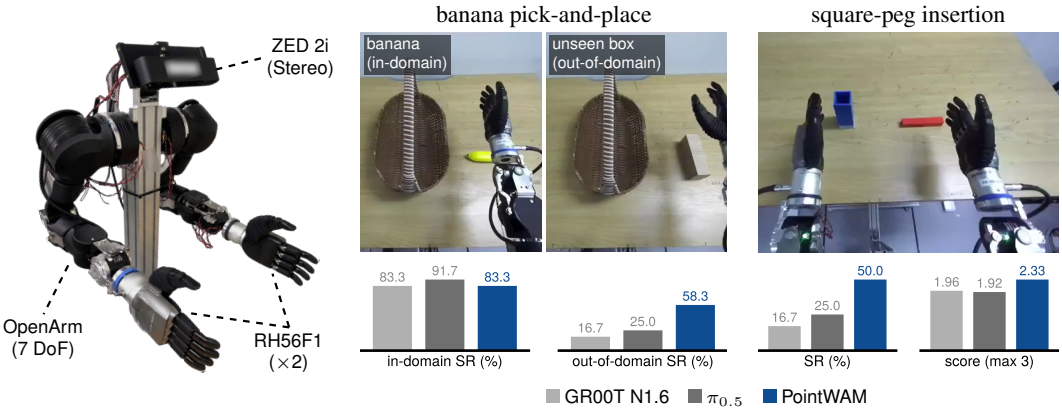
includes large-scale real and simulated robot demonstrations. Fast-WAM (Yuan et al., 2026) represents world action models; it builds on a video generation backbone and learns robot actions with video-prediction supervision. PointACT (Chen et al., 2026a) represents policies that use depth, adding pre-trained 3D point features to the action expert of a vision-language policy. Only Fast-WAM forecasts future scenes, and it forecasts images rather than the 3D motion of the scene and hands. On the real robot, we compare with GR00T N1.6 and $\pi_{0.5}$, fine-tuned on the same data.

4.2 Comparison with Previous Approaches

DexJoCo. With one policy across the ten tasks, PointWAM achieves the highest success rate on eight tasks and reaches 69.0% average SR (Table 1), outperforming the strongest baseline, GR00T N1.6, by a large margin of 11.7 percentage points, and $\pi_{0.5}$ by 19.0. Both pre-train on large-scale robot demonstrations and web data, whereas PointWAM uses no robot demonstrations beyond DexJoCo. PointWAM also ranks first with one policy per task, and its margin over the strongest baseline widens from 5.3 to 22.2 percentage points under full randomization of camera pose, lighting, and texture (Tables 5 and 6 in Appendix D). Figure 3 shows the scene and hand trajectories PointWAM predicts, and Fig. 4 shows bimanual hanoi rollouts from one initial state, where PointWAM succeeds and GR00T N1.6 times out; Appendix E shows more rollouts.

RoboDojo-Precision. With no robot data beyond the benchmark's own demonstrations, PointWAM reaches an average score of 8.8 and 4.8% SR (Table 2). Fast-WAM and PointACT also pre-train on no robot data, only on videos, image-text pairs, or 3D scenes, and stay near zero (0.3, 0.2% and 1.6, 0.0%), even though PointACT observes depth. GR00T N1.6, whose pre-training includes large-scale robot data with grippers, reaches 6.2 and 2.3%, still below PointWAM, and leads on three of the eight tasks. PointWAM, pre-trained on human hands alone, thus shows initial transfer to grippers, and depth alone does not account for its lead over Fast-WAM and PointACT.

Real robot. Figure 5b summarizes the real-robot results. PointWAM outperforms both GR00T N1.6 and $\pi_{0.5}$ over all trials of each task. On banana PnP, it performs comparably with the in-domain banana but reaches 58.3% SR with the out-of-domain unseen box, against 16.7% and 25.0%. Since the box resembles the table in color, the RGB-based baselines likely struggle to find it, while PointWAM can rely on its 3D shape. On square-peg insertion, PointWAM reaches 50.0% SR against 16.7% and 25.0%. All three methods usually lift both parts, but the baselines often fail at the insertion, which needs precise bimanual placement; PointWAM inserts in 12 of 21 such trials, against 4 of 19 and 6 of 18. Figure 12 in Appendix E shows example rollouts.



(a) Platform (b) Tasks, success rates (SR), and scores over 24 trials

Figure 5: **Real-robot evaluation.** (a) The OpenArm platform. (b) The two tasks, one single-arm and one bimanual, with the success rate and score of each method.

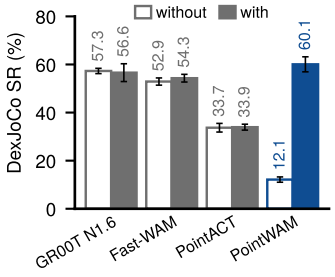


Figure 6: **Pre-training gain.** Average SR (%) with and without EgoDex pre-training.

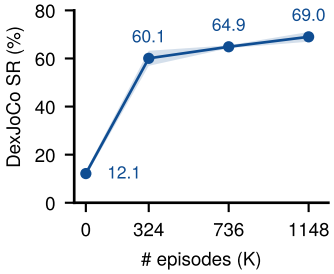


Figure 7: **Data scaling.** Each point plots average SR (%) versus pre-training episodes.

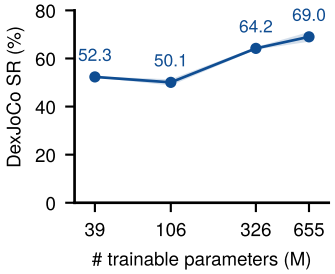


Figure 8: **Model scaling.** Each point plots average SR (%) versus trainable parameters.

4.3 Human-Demonstration Pre-Training and Scaling

Transfer from human demonstration videos. We first measure how much EgoDex (Hoque et al., 2026) pre-training improves each method (Fig. 6; whiskers show ±1 standard deviation over three seeds), fixing Volt-H and 60k fine-tuning iterations for PointWAM. The baselines already build on large *pre-trained* models; GR00T N1.6 inherits vision-language and robot-policy pre-training (NVIDIA et al., 2025), and Fast-WAM and PointACT build on pre-trained Wan2.2 (Team Wan et al., 2025) video and Qwen2.5-VL (Bai et al., 2025) backbones (Yuan et al., 2026; Chen et al., 2026a). PointWAM instead trains its backbone *from scratch*, keeping only its frozen encoders pre-trained, and hence starts from a lower SR than the baselines.

PointWAM nevertheless benefits far more from EgoDex pre-training. Its SR rises nearly fivefold, from 12.1% to 60.1%, whereas the baselines change by at most 1.4 percentage points (GR00T N1.6 from 57.3% to 56.6%, Fast-WAM from 52.9% to 54.3%, and PointACT from 33.7% to 33.9%). Even Fast-WAM, which also forecasts the future but as video frames, gains little. These results suggest that the gain comes from the shared 3D representation, in which human and robot demonstrations take the same scene and hand point trajectories, rather than from world modeling alone.

Data scaling. We next vary the amount of human demonstrations with the Volt-H backbone and 60k fine-tuning iterations fixed, adding half and all of VITRA (Li et al., 2026b) to the 324K EgoDex episodes (Fig. 7; shading shows ±1 standard deviation over three seeds). SR rises from 60.1% with EgoDex alone to 64.9% with 736K episodes and 69.0% with the full 1.15M episodes. EgoDex alone already surpasses GR00T N1.6 (57.3%), and the full corpus adds 8.9 percentage points.

Model scaling. We then vary the trainable capacity of the backbone with the full EgoDex and VITRA corpus and 60k fine-tuning iterations fixed (Fig. 8; shading shows ±1 standard deviation

Table 3: **Component ablations.** Average SR (%) of a Volt-L12 model pre-trained on EgoDex, varying one setting per panel (Section 4.4). Shading marks the reference, and subscripts are standard deviations over three seeds of 50 episodes per task.

(a) vision encoder

case	SR (%)
DINOv3	$59.0_{1.3}$
Mosaic3D	$64.4_{1.7}$

(b) world modeling

case	SR (%)
without	$53.5_{2.7}$
2D frame	$60.1_{1.5}$
3D points	$64.4_{1.7}$

(c) hand representation

case	points	SR (%)
keypoints	10	$64.4_{1.7}$
surface points	10	$43.6_{0.8}$
surface points	512	$44.6_{0.9}$

(d) voxel size

size	SR (%)
0.5 cm	$59.9_{1.5}$
1 cm	$64.4_{1.7}$
2 cm	$52.2_{1.2}$

over three seeds). SR stays near 50% for Volt-S (39M, 52.3%) and Volt-B (106M, 50.1%) and rises to 64.2% for Volt-L (326M) and 69.0% for Volt-H (655M). Appendix C lists the configurations.

4.4 Component Analysis

We change one setting at a time from a common reference, a Volt-L12 backbone, named for halving the depth of Volt-L to 12 layers, pre-trained on EgoDex and fine-tuned for 60k iterations (Table 3). Since the ablations require many training runs, the reference is smaller and uses fewer human demonstration videos than the benchmark model, so its SR is lower than in Table 1. Appendix C gives the training settings, and Appendix D.2 further varies the patch size and encoder fine-tuning.

Vision encoder (a). The 3D encoder Mosaic3D, which computes features on the point cloud, reaches 64.4%. Lifting DINOv3 (Siméoni et al., 2026) image features onto the points, as in PointWorld (Huang et al., 2026), lowers SR to 59.0%. Lifted features carry 3D positions but are computed from images. The gap suggests that encoding the 3D structure of the scene in the features themselves, not only in their positions, matters for dexterous manipulation.

World modeling (b). Removing scene-trajectory supervision from both pre-training and fine-tuning, with the depth input, 3D backbone, and hand head unchanged, lowers SR from 64.4% to 53.5%. Forecasting the scene thus improves success beyond observing it in 3D, and Fig. 3 shows the predicted scene moving with the hands. The 2D frame row uses the DINOv3 encoder of (a) and forecasts the DINOv3 latents of the frame at $t+T$ through learnable queries that read the backbone's output tokens, as in FLARE (Zheng et al., 2025) but with DINOv3 rather than SigLIP-2 latents, so that its input and its target share one feature space. It reaches 60.1%, 6.6 percentage points above the model without world modeling but 4.3 below the reference. Forecasting the 3D trajectories of scene points from 3D features thus outperforms forecasting future image latents from image features.

Hand representation and voxel size (c–d). In (c), anatomical keypoints reach 64.4%, whereas depth-sampled hand surface points reach 43.6% with 10 points per hand and 44.6% with 512. Keypoints likely help because they correspond between human and robot hands (Section 3.1). In (d), 1 cm voxels reach 64.4%, above both finer 0.5 cm voxels (59.9%) and coarser 2 cm voxels (52.2%). Coarser voxels may blur the contact geometry, while finer ones add little detail beyond the 224×224 depth input. We therefore use 1 cm voxels in all experiments, including Table 1.

5 Conclusion

We have introduced PointWAM, a 3D world action model that decomposes the world into a scene and hands. It jointly forecasts both as 3D point trajectories in a shared space-time coordinate frame and retargets the forecast hand motion to robot actions. Since human videos provide the same trajectories, PointWAM pre-trains on them without selecting task-specific points, raising average DexJoCo success from 12.1% to 69.0% and surpassing the strongest baseline by 11.7 percentage points. Observing the scene in 3D per se is not enough; forecasting the scene adds 10.9 percentage points over forecasting the hands alone. These results suggest that a shared 3D point-trajectory representation lets robots learn dexterous manipulation largely from human videos. Since PointWAM sees the world only through points, it can miss details finer than the point spacing, such as printed text, and surfaces that depth sensing cannot capture, such as transparent objects. Future work could extend what each point carries, from finer appearance to physical properties such as mass and friction, which govern how objects move under contact.

AI Use Statement

In this work, we used generative AI tools for implementing methods, cleaning and reformatting datasets, and assisting with translation. We have not used generative AI tools for proposing or refining hypotheses, developing theoretical models or conceptual frameworks, designing or providing feedback on research methodology or experiments, or interpreting results. Generating synthetic data sets, formulating mathematical claims, providing ingredients for or writing proofs, and supporting qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for drafting parts of the research paper. We have reviewed all AI-assisted work. The implemented methods are the model and training code, and the reformatted datasets are the human and robot demonstrations that our pipelines convert into point trajectories; two authors verified and tested this code for correctness. For the manuscript, AI tools assisted translation and drafting, and the authors reviewed, revised, and finalized every passage of the paper and its appendix. Three authors carried out and verified every analysis, including recomputing each reported success rate from the evaluation logs. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Ethics Statement

This work collects no new data from human subjects. We pre-train on two public human demonstration datasets under their terms of use. EgoDex (Hoque et al., 2026) is released under CC BY-NC-ND. VITRA releases its annotations under the MIT license (Li et al., 2026b), and its videos come from Ego4D (Grauman et al., 2022), EPIC-KITCHENS (Damen et al., 2021), Ego-Exo4D (Grauman et al., 2024), and Something-Something V2 (Goyal et al., 2017), which keep their own terms. We use both datasets for non-commercial research only. The benchmark experiments run in simulation, and the real-robot experiments use one robot platform (Section 4.1.1). Robot policies that act near people need safety measures that this work does not study.

Reproducibility Statement

Section 3 describes the model, and Appendix B gives the tokenization, decoder, retargeter, normalization, and loss details. Appendix C lists the training settings (Table 4), the evaluation protocols of both benchmarks, the checkpoint-selection rule, and the baseline configurations. Section 3.4 and Appendices B.1 and B.2 describe how human and robot demonstrations are converted into point trajectories. DexJoCo results for every method use the same official closed-loop evaluator with 50 episodes per task on each of three evaluation seeds; for PointWAM, the client also passes depth-derived point clouds. RoboDojo-Precision results use its official evaluator with 50 episodes per task on one seed. We will release the code, configurations, and checkpoints upon publication.

References

Shan An, Ziyu Meng, Chao Tang, et al. Dexterous manipulation through imitation learning: A survey. *IEEE Transactions on Automation Science and Engineering (T-ASE)*, 23, 2026.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv*, 2016.

Shuai Bai, Keqin Chen, Xuejing Liu, et al. Qwen2.5-VL technical report. *arXiv*, 2025.

Chen Bao, Helin Xu, Yuzhe Qin, and Xiaolong Wang. DexArt: Benchmarking generalizable dexterous manipulation with articulated objects. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

Homanga Bharadhwaj, Roozbeh Mottaghi, Abhinav Gupta, and Shubham Tulsiani. Track2Act: Predicting point tracks from internet videos enables generalizable robot manipulation. In *European Conference on Computer Vision (ECCV)*, 2024.

Hongzhe Bi, Lingxuan Wu, Tianwei Lin, Hengkai Tan, Zhizhong Su, Hang Su, and Jun Zhu. H-RDT: Human manipulation enhanced bimanual robotic manipulation. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2026.

Antonio Bicchi. Hands for dexterous manipulation and robust grasping: A difficult road toward simplicity. *IEEE Transactions on Robotics and Automation (T-RA)*, 16(6), 2000.

Kevin Black, Noah Brown, James Darpinian, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *Conference on Robot Learning (CoRL)*, 2025.

Shizhe Chen, Paul Pacaud, and Cordelia Schmid. PointACT: Vision-language-action models with multi-scale point-action interaction. In *Robotics: Science and Systems (RSS)*, 2026a.

Tianxing Chen, Yue Chen, Zixuan Li, et al. RoboDojo: A unified sim-and-real benchmark for comprehensive evaluation of generalist robot manipulation policies. *arXiv*, 2026b.

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

Chris Choy and NVIDIA Research. WarpConvNet: High-performance 3D deep learning library. GitHub repository, 2025.

Dima Damen, Hazel Doughty, Giovanni Maria Farinella, et al. The EPIC-KITCHENS dataset: Collection, challenges and baselines. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 43(11), 2021.

Lingwei Dang, Juntong Li, Zonghan Li, Hongwen Zhang, Liang An, Wei Min, Yebin Liu, and Qingyao Wu. HarmoHOI: Harmonizing appearance and 3D motion for multi-view hand-object interaction synthesis. In *SIGGRAPH Asia Conference Papers*, 2026.

Yilun Du, Mengjiao Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Joshua B. Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning universal policies via text-guided video generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Bardienus P. Duisterhof, Kaifeng Zhang, Adam Hung, Bowen Wen, Stan Birchfield, Yunzhu Li, Deva Ramanan, and Jeffrey Ichnowski. PointZero: 3D point track completion for learning transferable 3D dynamics. *arXiv*, 2026.

Yankai Fu, Qiuxuan Feng, Ning Chen, et al. CordViP: Correspondence-based visuomotor policy for dexterous manipulation in real-world. In *Robotics: Science and Systems (RSS)*, 2025.

GEAR Team. GR00T N1.6: An improved open foundation model for generalist humanoid robots. NVIDIA Research, 2025.

Raktim Gautam Goswami, Amir Bar, David Fan, Tsung-Yen Yang, Gaoyue Zhou, Prashanth Krishnamurthy, Michael Rabbat, Farshad Khorrami, and Yann LeCun. World models for learning dexterous hand-object interactions from human videos. In *European Conference on Computer Vision (ECCV)*, 2026.

Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, et al. The "something something" video database for learning and evaluating visual common sense. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

Kristen Grauman, Andrew Westbury, Eugene Byrne, et al. Ego4D: Around the world in 3,000 hours of egocentric video. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Kristen Grauman, Andrew Westbury, Lorenzo Torresani, et al. Ego-Exo4D: Understanding skilled human activity from first- and third-person perspectives. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Jun Guo, Qiwei Li, Peiyan Li, Zilong Chen, Nan Sun, Yifei Su, Heyun Wang, Yuan Zhang, Xinghang Li, and Huaping Liu. Unified 4D world action modeling from video priors with asynchronous denoising. *arXiv*, 2026.

Siddhant Haldar and Lerrel Pinto. Point policy: Unifying observations and actions with key points for robot manipulation. In *Conference on Robot Learning (CoRL)*, 2025.

Jisang Han, Seonghu Jeon, Jaewoo Jung, René Zurbrügg, Honggyu An, Tifanny Portela, Marco Hutter, Marc Pollefeys, Seungryong Kim, and Sunghwan Hong. Geometric action model for robot policy learning. *arXiv*, 2026.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv*, 2016.

Ryan Hoque, Peide Huang, David J. Yoon, Mouli Sivapurapu, and Jian Zhang. EgoDex: Learning dexterous manipulation from large-scale egocentric video. In *International Conference on Learning Representations (ICLR)*, 2026.

Yucheng Hu, Yanjiang Guo, Pengchao Wang, Xiaoyu Chen, Yen-Jen Wang, Jianke Zhang, Koushil Sreenath, Chaochao Lu, and Jianyu Chen. Video prediction policy: A generalist robot policy with predictive visual representations. In *International Conference on Machine Learning (ICML)*, 2025.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision (ECCV)*, 2016.

Wenlong Huang, Yu-Wei Chao, Arsalan Mousavian, Ming-Yu Liu, Dieter Fox, Kaichun Mo, and Fei-Fei Li. PointWorld: Scaling 3D world models for in-the-wild robotic manipulation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Adam Hung, Bardienus Pieter Duisterhof, and Jeffrey Ichnowski. 3PoinTr: From human videos to robot policies with 3D point-track plans. *arXiv*, 2026.

Jiahao Jiang, Jianing Zhang, Zhenhan Yin, et al. WSA_1: a 3D-centric world-spatial-action model for generalizable robot control. *arXiv*, 2026.

Nikita Karaev, Yuri Makarov, Jianyuan Wang, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. CoTracker3: Simpler and better point tracking by pseudo-labelling real videos. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

Beomjun Kim, Seong Hyeon Park, Seunghoon Sim, Seungjun Moon, Sanghyeok Lee, and Jinwoo Shin. Dexterous point policy: Learning point-based dexterous hand policies from human demonstrations. *arXiv*, 2026a.

Byungjun Kim, Taeksoo Kim, Junyoung Lee, and Hanbyul Joo. Dexterous world models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026b.

Jisoo Kim, Jungbin Cho, Sanghyeok Chu, et al. Pri4R: Learning world dynamics for vision-language-action models with privileged 4D representation. In *Conference on Robot Learning (CoRL)*, 2026c.

Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, et al. Cosmos policy: Fine-tuning video models for visuomotor control and planning. In *International Conference on Learning Representations (ICLR)*, 2026d.

Junha Lee, Chunghyun Park, Jaesung Choe, Yu-Chiang Frank Wang, Jan Kautz, Minsu Cho, and Chris Choy. Mosaic3D: Foundation dataset and model for open-vocabulary 3D segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Seungjae Lee, Yoonkyo Jung, Jusuk Lee, Jonghun Shin, Amir Hossein Shahidzadeh, Yao-Chih Lee, H. Jin Kim, Jia-Bin Huang, and Furong Huang. μ_0: A scalable 3D interaction-trace world model. In *Conference on Robot Learning (CoRL)*, 2026.

Baoyu Li, Xinchen Yin, Mengying Lin, Yixin Zhang, and Danfei Xu. EgoWAM: World action models beyond pixels with in-the-wild egocentric human data. In *Conference on Robot Learning (CoRL)*, 2026a.

Qixiu Li, Yu Deng, Yaobo Liang, et al. Scalable vision-language-action model pretraining for robotic manipulation with real-life human activity videos. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2026b.

Shuang Li, Yihuai Gao, Dorsa Sadigh, and Shuran Song. Unified video action model. In *Robotics: Science and Systems (RSS)*, 2025a.

Xianhang Li, Haoqin Tu, Mude Hui, et al. What if we recaption billions of web images with LLaMA-3? In *International Conference on Machine Learning (ICML)*, 2025b.

Yue Liao, Pengfei Zhou, Siyuan Huang, et al. Genie envisioner: A unified world foundation platform for robotic manipulation. In *International Conference on Learning Representations (ICLR)*, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

Jiangran Lyu, Ziming Li, Xuesong Shi, Chaoyi Xu, Yizhou Wang, and He Wang. DyWA: Dynamics-adaptive world action model for generalizable non-prehensile manipulation. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

NVIDIA, Johan Bjorck, Fernando Castañeda, et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv*, 2025.

NVIDIA, Aditi, Niket Agarwal, et al. Cosmos 3: Omnimodal world models for physical AI. *arXiv*, 2026.

Allison M. Okamura, Niels Smaby, and Mark R. Cutkosky. An overview of dexterous manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

Yuzhe Qin, Binghao Huang, Zhao-Heng Yin, Hao Su, and Xiaolong Wang. DexPoint: Generalizable point cloud reinforcement learning for sim-to-real dexterous manipulation. In *Conference on Robot Learning (CoRL)*, 2022a.

Yuzhe Qin, Yueh-Hua Wu, Shaowei Liu, Hanwen Jiang, Ruihan Yang, Yang Fu, and Xiaolong Wang. DexMV: Imitation learning for dexterous manipulation from human videos. In *European Conference on Computer Vision (ECCV)*, 2022b.

Delin Qu, Haoming Song, Qizhi Chen, et al. SpatialVLA: Exploring spatial representations for visual-language-action model. In *Robotics: Science and Systems (RSS)*, 2025.

J. Kenneth Salisbury and John J. Craig. Articulated hands: Force control and kinematic issues. *The International Journal of Robotics Research (IJRR)*, 1(1), 1982.

Oriane Siméoni, Huy V. Vo, Maximilian Seitzer, et al. DINOv3. *Transactions on Machine Learning Research (TMLR)*, 2026.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568, 2024.

Team Wan, Ang Wang, Baole Ai, et al. Wan: Open and advanced large-scale video generative models. *arXiv*, 2025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Hanwen Wang, Weizhi Zhao, Xiangyu Wang, et al. DexJoCo: A benchmark and toolkit for task-oriented dexterous manipulation on MuJoCo. *arXiv*, 2026a.

Jianyuan Wang, Minghao Chen, Shangzhan Zhang, Nikita Karaev, Johannes Schönberger, Patrick Labatut, Piotr Bojanowski, David Novotny, Andrea Vedaldi, and Christian Rupprecht. VGGT-Ω. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026b.

Bowen Wen, Shaurya Dewan, and Stan Birchfield. Fast-FoundationStereo: Real-time zero-shot stereo matching. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Chuan Wen, Xingyu Lin, John So, Kai Chen, Qi Dou, Yang Gao, and Pieter Abbeel. Any-point trajectory modeling for policy learning. In *Robotics: Science and Systems (RSS)*, 2024.

Xiaoyang Wu, Li Jiang, Peng-Shuai Wang, Zhijian Liu, Xihui Liu, Yu Qiao, Wanli Ouyang, Tong He, and Hengshuang Zhao. Point Transformer V3: Simpler, faster, stronger. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Lishan Yang, Wenxuan Song, Xi Wang, et al. 4D-WAM: Infusing spatiotemporal awareness into world action models through trajectory fields. *arXiv*, 2026.

Kadir Yilmaz, Adrian Kruse, Tristan Höfer, Daan de Geus, and Bastian Leibe. Volume Transformer: Revisiting vanilla transformers for 3D scene understanding. In *European Conference on Computer Vision (ECCV)*, 2026.

Chengbo Yuan, Chuan Wen, Tong Zhang, and Yang Gao. General flow as foundation affordance for scalable robot learning. In *Conference on Robot Learning (CoRL)*, 2024.

Tianyuan Yuan, Zibin Dong, Yicheng Liu, and Hang Zhao. Fast-WAM: Do world action models need test-time future imagination? *arXiv*, 2026.

Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3D diffusion policy: Generalizable visuomotor policy learning via simple 3D representations. In *Robotics: Science and Systems (RSS)*, 2024.

Huajian Zeng, Lingyun Chen, Jiaqi Yang, Yuantai Zhang, Fan Shi, Peidong Liu, and Xingxing Zuo. FlowHOI: Flow-based semantics-grounded generation of hand-object interactions for dexterous robot manipulation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2026.

Jianing Zhang, Chenhao Zheng, Yajun Yang, et al. MolmoMotion: Forecasting point trajectories in 3D with language instruction. *arXiv*, 2026.

Yujia Zhang, Xiaoyang Wu, Yixing Lao, Chengyao Wang, Zhuotao Tian, Naiyan Wang, and Hengshuang Zhao. Concerto: Joint 2D-3D self-supervised learning emerges spatial representations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Robotics: Science and Systems (RSS)*, 2023.

Haoyu Zhen, Xiaowen Qiu, Peihao Chen, Jincheng Yang, Xin Yan, Yilun Du, Yining Hong, and Chuang Gan. 3D-VLA: A 3D vision-language-action generative world model. In *International Conference on Machine Learning (ICML)*, 2024.

Ruijie Zheng, Jing Wang, Scott Reed, et al. FLARE: Robot learning with implicit world modeling. In *Conference on Robot Learning (CoRL)*, 2025.

A Extended Related Work

In this section, we extend Section 2 to further world action models, 3D representations for robot policies, and other ways of learning from human demonstration videos.

Future targets of world action models. Beyond the dense 3D scene motion forecast by EgoWAM and PointWorld, world action models differ in what future they forecast and what they condition it on. Several forecast geometric futures, such as object poses in DyWA (Lyu et al., 2025) and depth in GAM (Han et al., 2026). 3D-VLA (Zhen et al., 2024) forecasts goal RGBD images or point clouds, and WSA_1 (Jiang et al., 2026) forecasts latent 3D scene states. Others keep an image future. Fast-WAM (Yuan et al., 2026) and Cosmos 3 (NVIDIA et al., 2026) build on video generation backbones, 4D-WAM (Yang et al., 2026) aligns its features with 3D trajectory fields during training, and FLARE (Zheng et al., 2025) aligns future tokens of its policy with image features of a future frame. X-WAM (Guo et al., 2026) forecasts multi-view RGB-D video alongside actions. In each of these, the forecast future takes a different form from the motion that produces the action, whereas PointWAM expresses the scene and the hands with the same 3D point trajectories and retargets only the hands to actions. Dexterous World Model (Kim et al., 2026b) conditions its interaction video on given hand trajectories, much as PointWorld conditions on future robot trajectories, so it also requires a supplied hand trajectory rather than forecasting one itself for robot control.

3D representations for robot policies. PointWAM observes the scene as a point cloud, which gives a policy direct access to 3D geometry. DexPoint (Qin et al., 2022a) and DexArt (Bao et al., 2023) combine observed geometry with kinematics-derived point clouds for dexterous reinforcement learning, and DP3 (Ze et al., 2024) learns imitation policies from encoded point clouds. Even the image-based SpatialVLA (Qu et al., 2025) augments its features with 3D positions from estimated depth. In all of these, geometry serves only as input, and the control objective alone trains the backbone. CordViP (Fu et al., 2025) adds contact and hand–arm correspondence objectives. PointACT (Chen et al., 2026a) keeps a 2D vision–language backbone and admits pre-trained 3D point features only in its action expert, through multi-scale point–action attention, so geometry conditions the action but never the backbone's representation. PointWAM also builds on pre-trained 3D features but supervises the backbone with future hand and scene trajectories, which human demonstration videos supply without recorded robot actions.

Other ways of learning from human demonstration videos. Beyond the point-based policies in Section 2, methods differ in what they transfer from the video to the robot. Retargeting the human hand pose (Qin et al., 2022b) transfers actions directly but sets no scene-forecasting objective, and neither do the hand and object poses of FlowHOI (Zeng et al., 2026). DexWM (Goswami et al., 2026) learns a latent world model from egocentric human videos with hand keypoints as actions, so it forecasts the effect of given hand motion rather than the motion itself, and HarmoHOI (Dang et al., 2026) co-generates hand–object interaction videos and 3D point tracks for synthesis rather than robot control. Track-based methods (Wen et al., 2024; Bharadhwaj et al., 2024) forecast the future image positions of arbitrary points, but the tracks and the features behind them stay in image space. General Flow (Yuan et al., 2024) forecasts 3D flow only at query points sampled on the manipulated object, MolmoMotion (Zhang et al., 2026) forecasts 3D trajectories of query points on an object of interest, and μ_0 (Lee et al., 2026) forecasts 3D traces of interaction keypoints selected with a vision-language model. In all three, something outside the model decides which scene points it supervises. PointWAM instead encodes the full point cloud and supervises the trajectory of every observed point and of the hand, learned from human demonstration videos without robot actions.

B Implementation Details

Appendix B.1 describes how we convert human demonstrations into point trajectories, and Appendix B.2 does the same for DexJoCo and ends with the changes for RoboDojo-Precision. The remaining subsections give the model and training settings.

B.1 Human-Demonstration Preprocessing

Human videos record neither robot states nor simulator states, so we estimate hand and scene trajectories from the videos themselves. EgoDex (Hoque et al., 2026) annotates an $SE(3)$ transform for each of 25 joints per hand, and VITRA (Li et al., 2026b) reconstructs 21 joints for each visible hand. We keep the joint positions and map them to the hand keypoints described below. When a hand is missing, we mark its keypoints invalid instead of imputing them.

Hand keypoints. We turn each human hand into the same ten keypoints as the robot hand (Appendix B.2). Human demonstrations are first rotated, with their scenes, into a z-up frame like the robot-base frame of robot demonstrations. At each time t, we build a palm frame $(\boldsymbol{o}_t, \boldsymbol{B}_t)$ on the human hand, with its origin at the middle knuckle and its axes along the wrist-to-knuckle direction, along the knuckle row, and their cross product. Indexing the ten keypoints of one hand by k, the six palm keypoints overlay the robot palm on the human palm, placing the Allegro palm points at their fixed offsets $\boldsymbol{d}^{(k)}$ from the robot's palm in this frame. The four fingertip keypoints are the tracked tips $\boldsymbol{f}_t^{(k)}$ of the thumb, index, middle, and ring fingers:

$$\boldsymbol{q}_t^{(k)} = \boldsymbol{o}_t + \boldsymbol{B}_t\, \boldsymbol{d}^{(k)} \quad (k = 1, \dots, 6), \qquad \boldsymbol{q}_t^{(k)} = \boldsymbol{f}_t^{(k)} \quad (k = 7, \dots, 10). \tag{8}$$

The little finger is omitted because the Allegro hand has no counterpart to it.

Clips. We split each episode into 1 s clips of 11 frames at 10 Hz that start every 0.5 s, and discard clips whose summed wrist displacement is below 2 cm. We linearly resample each clip to 31 frames, so that its future steps match the horizon $T = 30$ of the DexJoCo demonstrations. EgoDex contributes 5.2M clips from 324K episodes, and VITRA contributes 824,157 shorter episodes.

Depth. VGGT-Ω (Wang et al., 2026b) predicts per-frame depth only up to scale. We recover the metric scale from the hands, which ARKit annotates in EgoDex and VITRA reconstructs. For each chunk of frames, we project the hand joints into the images, discard depth ratios $z_{\text{GT}}/z_{\text{pred}}$ more than a factor of two from their median, and take the median of the remaining ratios as the scale.

Scene tracks. We resize frames to 640×360, place queries on every second pixel of each clip's first frame, and track them with CoTracker3 (Karaev et al., 2025) in offline mode. Backward tracking is unnecessary, since every query starts at the first frame. Each track reads the scaled depth at its pixels and is lifted to the world frame with the camera intrinsics and poses. We keep points that are visible with valid depth in the first frame, deduplicate them on a 1.5 cm grid, and subsample at most 12,000. Since queries restart in every clip, point correspondence holds only within a clip.

B.2 Robot-Demonstration Preprocessing

Point cloud and scene trajectories. In DexJoCo, robot fine-tuning and evaluation use 224×224 depth images, which we unproject with the camera calibration, reduce to one point per 1 cm voxel, and crop to a box around the robot base, 1 m to each side and from 0.2 m below to 1 m above it. The box adapts the bounds filter of PointWorld (Huang et al., 2026), a cube about the scene center, to exclude the floor and walls. Since the DexJoCo dataset records object poses only at reset, we replay each demonstration to recover them over time. For each successful replay, simulator segmentation removes robot points from the initial point cloud and assigns every remaining point, background included, to an object or rigid part, whose rigid motion defines the point's trajectory.

Robot hand keypoints. Each robot hand has ten keypoints, six on the palm and four on the fingertips. The fingertip keypoints lie on the contact pads, since points on the distal joint axes would miss the motion of four finger joints during flexion. Separate kinematic chains compute the right and left hands, which record their joints in different orders, and each hand's measured pose places its keypoints in the scene frame without mirroring one hand onto the other. Bimanual inputs list the right hand before the left and give each hand its own ID.

RoboDojo-Precision. Its two ARX X5 arms carry parallel-jaw grippers, so we keep the pre-trained interface and change only what the embodiment requires. Following Point Policy (Haldar & Pinto, 2025), which places robot points by a rigid transform about the gripper pose, each arm again has ten keypoints, the six palm offsets placed on the gripper frame from forward kinematics and the two fingertips, with the last point repeated to fill the ten slots. The point cloud fuses the depth of the head and two wrist cameras and drops points within 5 cm of the arm and finger segments. The

action replaces the sixteen finger joints with one gripper command, so $d_a = 7$ per arm, and the 30-step chunk spans 1.2 s at the benchmark's 25 Hz. Since the released demonstrations carry no object poses or segmentation, scene trajectories cannot be reconstructed, and fine-tuning supervises only the hand keypoints and actions while the scene head stays frozen.

B.3 3D Trajectory Forecaster

Tokenization. We build on the Volt (Yilmaz et al., 2026) implementation in WarpConvNet (Choy & NVIDIA Research, 2025). Where Volt projects the raw voxel features of each patch, this implementation first computes a feature $\boldsymbol{g}^{(c)} \in \mathbb{R}^{d_{\text{stem}}}$ for every occupied voxel c with a sparse-convolutional stem. The stem applies a 3^3 sparse convolution to $d_{\text{stem}} = 64$ channels, LayerNorm (Ba et al., 2016), GELU (Hendrycks & Gimpel, 2016), and two residual blocks. The token of patch j concatenates the stem features of its K_v^3 voxels in a fixed spatial order, with zeros for empty voxels, and projects them to width d:

$$\boldsymbol{z}_j^v = \boldsymbol{W}_v[\boldsymbol{g}^{(c_1)}; \ldots; \boldsymbol{g}^{(c_{K_v^3})}] + \boldsymbol{b}_v \in \mathbb{R}^d. \tag{9}$$

Each patch spans $K_v = 5$ voxels, or 5 cm, per side, and the fixed order preserves where each voxel sits inside the patch, which pooling would discard.

Backbone. The backbone is Volt-H, with 32 layers, 20 heads, and width $d = 1280$. The L instruction features from the frozen Recap-CLIP text encoder (Li et al., 2025b) are projected to width d and precede the M vision tokens in Eq. (3). Axial 3D rotary position embeddings (Su et al., 2024; Yilmaz et al., 2026) encode the patch coordinates of vision tokens. Instruction tokens receive no rotation, since the text encoder already encodes word order. Attention is bidirectional and stays within one observation, and only the output vision tokens are decoded.

Trajectory decoder. Before the Detokenizer of Eq. (4) expands a token, the token passes through LayerNorm, GELU, a linear layer, LayerNorm, and GELU. The saved mapping $\mathcal{I}_t$ then selects each point's feature from the K_v^3 expanded voxel features. The stem features $\boldsymbol{S}_t \in \mathbb{R}^{(N+K)\times d_{\text{stem}}}$ give each scene point or hand keypoint the feature $\boldsymbol{g}^{(c)}$ of its voxel. The fusion MLP consists of a linear layer, LayerNorm, GELU, and a zero-initialized linear layer, so training starts from the detokenized features alone. A final LayerNorm and GELU applied to the residual sum give the per-point features $\boldsymbol{F}_t$ that the scene and hand heads of Eq. (5) read to forecast trajectories.

B.4 Action Retargeter

The retargeter of Eq. (6) stacks two decoder blocks with four attention heads. MLP_{disp} and $\text{MLP}_{\text{state}}$ each have two linear layers with a SiLU between them, and MLP_{disp} reads the K rows of $\mathbf{V}_{t,\tau}$ as one vector in a fixed order, which preserves hand and keypoint identities. Each block applies LayerNorm before its self-attention, cross-attention, and feed-forward layers. Self-attention among the action queries is unmasked, so the decoder predicts the whole action chunk in parallel rather than one step at a time. With $\boldsymbol{H}_t \in \mathbb{R}^{T\times d_r}$ the output queries of the last block, the final linear layer follows a LayerNorm:

$$\boldsymbol{A}_t = \text{LayerNorm}(\boldsymbol{H}_t)\,\boldsymbol{W}_a + \boldsymbol{b}_a. \tag{10}$$

The retargeter learns the map from robot demonstrations instead of solving inverse kinematics, so it enforces no explicit joint limits or collision constraints. Its targets are recorded actions, which already respect joint limits and avoid collisions since a robot executed them.

B.5 Normalization and Loss Weighting

Trajectory normalization. The heads of Eq. (5) forecast standardized displacements. We standardize scene and hand displacements separately, with coordinate-wise means $\boldsymbol{\mu}_\tau^{\text{scene}}, \boldsymbol{\mu}_\tau^{\text{hand}} \in \mathbb{R}^3$ and standard deviations $\boldsymbol{\sigma}_\tau^{\text{scene}}, \boldsymbol{\sigma}_\tau^{\text{hand}} \in \mathbb{R}^3$ for each horizon step, and fine-tuning keeps the statistics of pre-training. With $\bar{\boldsymbol{u}}_{t,\tau}^{(i)}$ and $\bar{\boldsymbol{v}}_{t,\tau}^{(k)}$ the entries of $\mathbf{U}_t$ and $\mathbf{V}_t$, we recover the displacements of Eq. (1) as

$$\boldsymbol{u}_{t,\tau}^{(i)} = \boldsymbol{\mu}_\tau^{\text{scene}} + \boldsymbol{\sigma}_\tau^{\text{scene}} \odot \bar{\boldsymbol{u}}_{t,\tau}^{(i)}, \qquad \boldsymbol{v}_{t,\tau}^{(k)} = \boldsymbol{\mu}_\tau^{\text{hand}} + \boldsymbol{\sigma}_\tau^{\text{hand}} \odot \bar{\boldsymbol{v}}_{t,\tau}^{(k)}, \tag{11}$$

Table 4: **Training settings for human-demonstration pre-training and DexJoCo fine-tuning.**

	Pre-training	Fine-tuning
Data	EgoDex + VITRA (1.15M episodes)	DexJoCo (959 demos)
Iterations	60k	60k
Global batch size	192	192
GPUs	4 NVIDIA B200	4 NVIDIA B200
Optimizer	AdamW (Loshchilov & Hutter, 2019)	AdamW
Learning rate	10^{-4}	10^{-4}
Learning-rate schedule	constant	constant
Weight decay	0.01	0.01
Warm-up	none	none
Gradient-norm clipping	5.0	5.0
Stochastic depth (Huang et al., 2016)	0 to 0.3 over depth	0 to 0.3 over depth
Precision	bfloat16 mixed	bfloat16 mixed
Window length	31 frames (1 s of video)	31 frames (0.6 s simulated)
Voxel size	1 cm	1 cm
Patch size K_v	5	5
Maximum scene points	12,000	unrestricted
Hand keypoints	20	10 or 20
Trainable / frozen parameters	651M / 592M	655M / 592M
Frozen encoders	Mosaic3D, Recap-CLIP text	Mosaic3D, Recap-CLIP text
Retargeter	not used	trained
Initialization	random	pre-training checkpoint (60k)
Training time	34 h	19 h

where $\odot$ denotes coordinate-wise multiplication. Adding them to $\boldsymbol{p}_t^{(i)}$ and $\boldsymbol{q}_t^{(k)}$ gives the forecast positions. Before standardization, we clip every coordinate of the scene displacements to the range $[-0.5, 0.5]$ m, and the losses of Eq. (7) compare standardized displacements.

Action normalization. Following the Franka–Allegro interface of DexJoCo (Wang et al., 2026a), each arm has three end-effector position coordinates, three rotation coordinates, and sixteen finger-joint targets, so $d_a = 22$ per arm. We express each target relative to the robot state $\boldsymbol{s}_t$, as differences for positions and finger joints and as a rotation vector for orientation, then standardize each coordinate per horizon step with statistics of the robot demonstrations. The target $\boldsymbol{A}_t^*$ of Eq. (7) is a recorded chunk after this map, which we invert at inference. Single-arm and bimanual embodiments have separate retargeters and action statistics but share the trajectory forecaster, whose per-point outputs do not depend on the layout of the action space.

Loss weighting. In Eq. (7), a hand keypoint is valid when its hand is present and fills a real slot of the bimanual layout rather than padding, and the hand loss averages over valid keypoint–step pairs. Following PointWorld (Huang et al., 2026), the scene weights exclude points without a valid target and softly emphasize motion between adjacent frames. Their unnormalized values are $\mathrm{sigmoid}(5(\delta_{i,\tau}/0.005 - 1))$, with $\delta_{i,\tau} = \|\boldsymbol{p}_{t+\tau}^{(i)} - \boldsymbol{p}_{t+\tau-1}^{(i)}\|_2$ in meters, so a static point keeps a small positive weight; we normalize them over valid point–step pairs in the batch. Each squared error is divided by 3 and the action error by Td_a, so every term is a per-coordinate mean. Hand and action errors average over valid batch elements, and all loss coefficients equal one.

C Experimental Details

C.1 Training Settings

Main model. Table 4 lists the settings of the Volt-H multi-task model in Table 1. We pre-train it for 60k iterations on EgoDex and VITRA (1,148,324 episodes in total) and fine-tune it for 60k iterations on DexJoCo, with the 3D and text encoders frozen in both stages.

Component ablations. Ablations use Volt-L12, a backbone with width 1024, 12 layers, and 16 heads, which trains 175M parameters. We pre-train it on EgoDex for 40k iterations and fine-tune it for 60k, starting from the pre-training snapshot with the lowest held-out hand-trajectory error (35k for the reference). Every variant, surface points included, changes both pre-training and fine-tuning.

Model scaling. The model-scaling study varies backbone width and depth while keeping the other components fixed. The Volt-S, Volt-B, Volt-L, and Volt-H configurations in Fig. 8 have widths of 384, 768, 1024, and 1280 and depths of 12, 12, 24, and 32. All share frozen encoders with 592M parameters, and during fine-tuning they train 39M, 106M, 326M, and 655M parameters, including the 4.6M-parameter retargeter. Table 1 uses the largest configuration, Volt-H.

C.2 DexJoCo Evaluation Protocol

Inference timing. DexJoCo's evaluator runs asynchronously. It requests a new chunk once fewer than 24 of the 30 actions remain and holds the robot's pose whenever the buffer empties, so inference latency affects the executed behavior. On one B200 GPU, the Volt-L12 reference of Table 3 takes 55 ms per forward pass, 37 ms of it in the frozen encoder. Each simulator control step spans 20 ms, so a 30-step chunk covers 0.6 s. Human clips carry the same 31 frames over 1 s (Appendix B.1), so one predicted step spans about 33 ms in pre-training and 20 ms in fine-tuning.

Checkpoint selection. For the Volt-H multi-task model, we consider the 20k, 40k, and 60k fine-tuning snapshots and select on the first seed's ten-task average, which selects 60k; all component ablations use 60k. Single-task policies are trained with the view augmentation of Appendix D for 20k iterations (30k for bimanual hanoi and microwave) and evaluated at that final snapshot.

C.3 RoboDojo-Precision Evaluation Protocol

We use the official RoboDojo evaluator (Chen et al., 2026b). Unlike the DexJoCo evaluator, it runs synchronously. The simulator waits for each query, executes the whole action chunk, and then queries again, so inference latency does not affect the outcome. The evaluation seed selects a fixed set of initial layouts, and we run 50 episodes per task on seed 0, taking the layouts in order. An episode ends at success or at a task-specific step limit, from 300 steps for insert key and deposit coin to 1,900 steps for fasten screws, that is, 12 to 76 s at the benchmark's 25 Hz.

We report the success rate and the benchmark's partial-progress score, each averaged over the eight tasks with equal weight. PointWAM is fine-tuned once from the 60k pre-training checkpoint, and we evaluate its 60k fine-tuning snapshot without further selection. The published benchmark instead trains three seeds per policy and evaluates each for 50 trials; we use one training seed and one evaluation seed for every method in Table 2. We leave out the public leaderboard entries, since they are trained on all RoboDojo tasks rather than the Precision demonstrations alone.

C.4 Real-world Experiments Details

Banana PnP. This task evaluates basic single-arm pick-and-place and, more importantly, the ability to rely on geometry when color cues are ambiguous. We collect 54 demonstrations in which the robot picks up a banana from various positions on the table and places it in a basket at a fixed location. We run 24 evaluation trials per method with positions sampled from those in the demonstrations, 12 with the banana and 12 with a box unseen in the demonstrations whose color closely matches the tabletop. A trial succeeds if the object lies entirely inside the basket at the end of the episode.

Square-peg insertion. This task evaluates precise grasping and tight bimanual coordination: the robot must lift a square peg and a blue hole, align them, and insert the peg into the hole. Because the peg is slippery, an imprecise grasp allows it to tilt between the fingers, which causes the insertion to fail. We collect 64 demonstrations with the peg and hole placed at various positions, and run 24 evaluation trials per method with positions sampled from those in the demonstrations. Since one end of the hole is capped, a trial is successful only if the peg remains in the hole at the end of the episode. To capture partial progress, we additionally score each trial on a scale from 0 to 3:

- **0**: fails to lift the peg.
- **1**: lifts the peg but fails to lift the hole.
- **2**: lifts both objects but fails to insert the peg.
- **3**: inserts the peg, which remains in the hole until the end of the episode (success).

C.5 Baselines

C.5.1 GR00T N1.6

Following the official recipe,[1] we fine-tune from the released GR00T N1.6-3B checkpoint (GEAR Team, 2025). We use the AdamW optimizer with a learning rate of 1×10^{-4}, an action horizon of 30, and input images at 224×224, with all remaining hyperparameters left at their defaults. With a global batch size of 256, we train a separate model per task for 30K steps (single-arm) or 60K steps (bimanual) in the DexJoCo single-task setting, and a single model across tasks for 60K steps in each of the DexJoCo multi-task and RoboDojo settings. For evaluation, we follow the default protocols of DexJoCo and RoboDojo-Precision. For the EgoDex (Hoque et al., 2026) pre-training experiment in Fig. 6, we follow H-RDT (Bi et al., 2026) and convert the ARKit hand annotations of EgoDex into a 44-D bimanual state per frame, consisting of the 3D position and quaternion orientation of each wrist, which serves as the end-effector pose, and the 3D positions of the five fingertips of each hand. Since human demonstrations carry no commanded actions, we define the action at step t as the next state, so that the model directly regresses future hand states. We pre-train for 30K steps with a global batch size of 192, the same budget as for the other baselines.

C.5.2 Fast-WAM

We use the official codebase[2] with the pre-trained Wan2.2-5B model. We train for 100K steps with a cosine learning-rate schedule initialized at 1×10^{-4}, an action horizon of 32, and a global batch size of 64. For DexJoCo, we concatenate three 224×224 camera views, while for RoboDojo-Precision, we concatenate three 192×256 views. For evaluation, we follow the default protocols of DexJoCo and RoboDojo-Precision. In Fig. 6, we pretrain Fast-WAM on EgoDex with the same hand-state actions and budget as for GR00T N1.6, with single-view 224×224 videos as visual supervision.

C.5.3 PointACT

Following the official implementation[3], we initialize the VLM backbone from Qwen2.5-VL-3B-Instruct (Bai et al., 2025) and keep it frozen, initialize the point encoder from the pretrained Concerto (Zhang et al., 2025) PTv3-Large (Wu et al., 2024) checkpoint, and train the remaining modules from scratch. We use the AdamW optimizer with a learning rate of 5×10^{-5} and a cosine learning rate schedule, with random rotations of the point cloud about the gravity axis and image augmentations as in the official implementation. With a global batch size of 256, we train a separate model per task for 30K steps in the DexJoCo single-task setting, and a single model across tasks for 60K steps in each of the DexJoCo multi-task and RoboDojo settings. Depth maps for both benchmarks are obtained in the same manner as for PointWAM. In Fig. 6, we pretrain PointACT on EgoDex with the same hand-state actions and budget as for GR00T N1.6.

D Additional Quantitative Results

D.1 Single-Task DexJoCo Results

Table 5 trains one policy per task in the object-placement setting. PointWAM averages 61.8%, 5.3 percentage points above the strongest published baseline, $\pi_{0.5}$ at 56.5%, and is best on five of the ten tasks. It falls below $\pi_{0.5}$ on hammer nail and fold glasses, below DP-C on pinch tongs, below GR00T N1.6 and DP-T on bimanual microwave, and below DP-T, $\pi_{0.5}$, and both GR00T models on bimanual photograph, where it reaches only 24.0%.

Table 6 evaluates the same policies under full randomization, which also varies the table height, camera pose, lighting, and texture. The single-task PointWAM policies of both tables are trained with view augmentation in 3D. Under full randomization, PointWAM averages 62.9% and is best on all ten tasks. Every baseline loses 9.4 to 32.8 percentage points, so the margin of PointWAM over the strongest baseline widens from 5.3 to 22.2 percentage points.

[1] https://github.com/NVIDIA/Isaac-GR00T
[2] https://github.com/yuantianyuan01/FastWAM
[3] https://github.com/cshizhe/robot-PointAct

Table 5: **Single-task success rate (%) on DexJoCo.** One policy per task in the object-placement setting; notation as in Table 1. Our Avg. is computed per seed over the ten retained tasks and averaged over the three seeds. ACT (Zhao et al., 2023), DP (Chi et al., 2023), $\pi_{0.5}$, and GR00T N1.5 (NVIDIA et al., 2025) results are from Wang et al. (2026a); GR00T N1.6 follows the same protocol.

Task	DP-T	DP-C	ACT	$\pi_{0.5}$	GR00T N1.5	GR00T N1.6	PointWAM
hammer nail	$81.3_{3.1}$	$58.7_{4.2}$	$50.0_{7.2}$	$\mathbf{84.7}_{5.0}$	$67.3_{4.2}$	$58.0_{2.0}$	$82.0_{4.0}$
click mouse	$62.0_{2.0}$	$74.0_{5.3}$	$61.3_{3.1}$	$64.7_{8.1}$	$85.3_{3.1}$	$66.0_{4.0}$	$\mathbf{89.3}_{4.2}$
pick bucket	$83.3_{3.1}$	$70.0_{2.0}$	$64.0_{4.0}$	$84.0_{7.2}$	$72.0_{6.0}$	$94.0_{2.0}$	$\mathbf{96.7}_{1.2}$
pinch tongs	$22.7_{5.8}$	$\mathbf{57.3}_{6.4}$	$31.3_{3.1}$	$24.0_{6.9}$	$12.7_{2.3}$	$33.3_{6.4}$	$50.0_{12.2}$
fold glasses	$53.3_{3.1}$	$54.0_{15.9}$	$47.3_{11.0}$	$\mathbf{72.0}_{3.5}$	$27.3_{2.3}$	$51.3_{8.3}$	$62.0_{4.0}$
water plant	$84.0_{3.5}$	$63.3_{3.1}$	$47.3_{4.6}$	$88.7_{3.1}$	$72.7_{1.2}$	$77.3_{9.0}$	$\mathbf{94.7}_{1.2}$
hanoi /B	$24.7_{4.6}$	$12.7_{3.1}$	$6.0_{2.0}$	$15.3_{3.1}$	$0.7_{1.2}$	$24.7_{1.2}$	$\mathbf{36.0}_{13.9}$
assembly /B	$4.7_{3.1}$	$3.3_{1.2}$	$0.0_{0.0}$	$5.3_{1.2}$	$0.7_{1.2}$	$4.0_{2.0}$	$\mathbf{11.3}_{7.0}$
microwave /B	$73.3_{11.6}$	$54.0_{12.5}$	$66.0_{2.0}$	$70.0_{3.5}$	$50.7_{4.6}$	$\mathbf{76.7}_{8.1}$	$72.0_{10.0}$
photograph /B	$\mathbf{56.7}_{4.6}$	$24.0_{8.7}$	$7.3_{1.2}$	$\mathbf{56.7}_{5.0}$	$40.7_{7.0}$	$30.0_{6.9}$	$24.0_{2.0}$
avg.	$54.6_{1.6}$	$47.1_{2.4}$	$38.1_{1.5}$	$56.5_{1.6}$	$43.0_{1.2}$	$51.5_{1.8}$	$\mathbf{61.8}_{1.1}$

Table 6: **Single-task success rate (%) on DexJoCo under full randomization.** Object placement, table height, camera pose, lighting, and texture vary. ACT, DP, $\pi_{0.5}$, and GR00T N1.5 results are from Wang et al. (2026a); notation follows Table 5. PointWAM evaluates the same viewpoint-augmented policies as Table 5, trained as described in Appendix D.

Task	DP-T	DP-C	ACT	$\pi_{0.5}$	GR00T N1.5	GR00T N1.6	PointWAM
hammer nail	$18.7_{1.2}$	$19.3_{3.1}$	$22.7_{6.1}$	$17.3_{5.0}$	$38.7_{8.3}$	$26.7_{7.0}$	$\mathbf{77.3}_{6.4}$
click mouse	$25.3_{8.1}$	$34.7_{4.2}$	$48.7_{5.0}$	$54.7_{7.0}$	$74.0_{2.0}$	$50.7_{4.2}$	$\mathbf{94.7}_{1.2}$
pick bucket	$58.7_{15.0}$	$68.0_{3.5}$	$36.0_{5.3}$	$78.7_{6.1}$	$69.3_{6.1}$	$85.3_{3.1}$	$\mathbf{96.0}_{4.0}$
pinch tongs	$18.7_{3.1}$	$28.7_{11.7}$	$23.3_{7.0}$	$18.7_{1.2}$	$5.3_{2.3}$	$38.0_{2.0}$	$\mathbf{47.3}_{5.8}$
fold glasses	$11.3_{1.2}$	$15.3_{7.6}$	$7.3_{3.1}$	$39.3_{3.1}$	$20.7_{3.1}$	$38.0_{2.0}$	$\mathbf{66.0}_{10.6}$
water plant	$56.0_{8.7}$	$54.0_{5.3}$	$52.7_{8.1}$	$75.3_{6.4}$	$66.0_{5.3}$	$76.7_{4.2}$	$\mathbf{94.0}_{2.0}$
hanoi /B	$0.7_{1.2}$	$9.3_{6.1}$	$4.7_{2.3}$	$15.3_{2.3}$	$0.0_{0.0}$	$22.0_{6.0}$	$\mathbf{47.3}_{6.1}$
assembly /B	$0.0_{0.0}$	$0.0_{0.0}$	$0.0_{0.0}$	$0.0_{0.0}$	$1.3_{1.2}$	$0.0_{0.0}$	$\mathbf{3.3}_{1.2}$
microwave /B	$21.3_{4.6}$	$62.7_{6.4}$	$50.0_{6.9}$	$54.7_{6.1}$	$42.0_{7.2}$	$52.7_{7.0}$	$\mathbf{74.0}_{4.0}$
photograph /B	$7.3_{1.2}$	$8.7_{4.2}$	$3.3_{1.2}$	$21.3_{2.3}$	$18.7_{7.0}$	$16.7_{8.1}$	$\mathbf{29.3}_{1.2}$
avg.	21.8	30.1	24.9	37.5	33.6	$40.7_{2.6}$	$\mathbf{62.9}_{2.2}$

Table 7: **Additional component ablations.** Same Volt-L12 reference and protocol as Table 3; shading marks the reference setting, and M is the average number of vision tokens per sample.

(a) patch size

K_v	M	SR (%)
3	1150	$67.1_{1.6}$
5	452	$64.4_{1.7}$
7	249	$44.5_{1.6}$

(b) encoder fine-tuning

case	SR (%)
frozen	$64.4_{1.7}$
fine-tuned	$54.9_{0.9}$

D.2 Patch Size and Encoder Fine-Tuning

Table 7 varies the patch size and encoder fine-tuning from the Volt-L12 reference of Table 3. With 1 cm voxels, reducing K_v from 5 to 3 raises SR from 64.4% to 67.1% but costs 1.63× the training step time and 1.66× the peak memory, while $K_v = 7$ lowers SR to 44.5% (a). We therefore keep $K_v = 5$, which balances manipulation success against training time and memory. Fine-tuning the Mosaic3D encoder instead of keeping it frozen lowers SR from 64.4% to 54.9% (b), so the encoder stays frozen in all other experiments of this paper.

E Additional Qualitative Results

Figures 9 and 10 show successful and failed PointWAM rollouts on the ten DexJoCo tasks. For each task, we take the first success and the first failure among eight rollouts, one episode on each of evaluation seeds 0–7; water plant succeeds in all sixteen rollouts we ran, so it has no failure row. In most failures, PointWAM grasps the object but stops short of the last step until the time limit. It holds the hammer above the nail, the peg beside the hole, the glasses next to the box, and the camera without pressing the shutter. Figure 11 shows rollouts on the eight RoboDojo-Precision tasks, and Figure 12 compares GR00T N1.6, $\pi_{0.5}$, and PointWAM on the two real-robot tasks.



Figure 9: **Successful PointWAM rollouts on DexJoCo.** One rollout per task from the multi-task policy, the first success among evaluation seeds 0–7, shown at four points up to its own t_{end}.

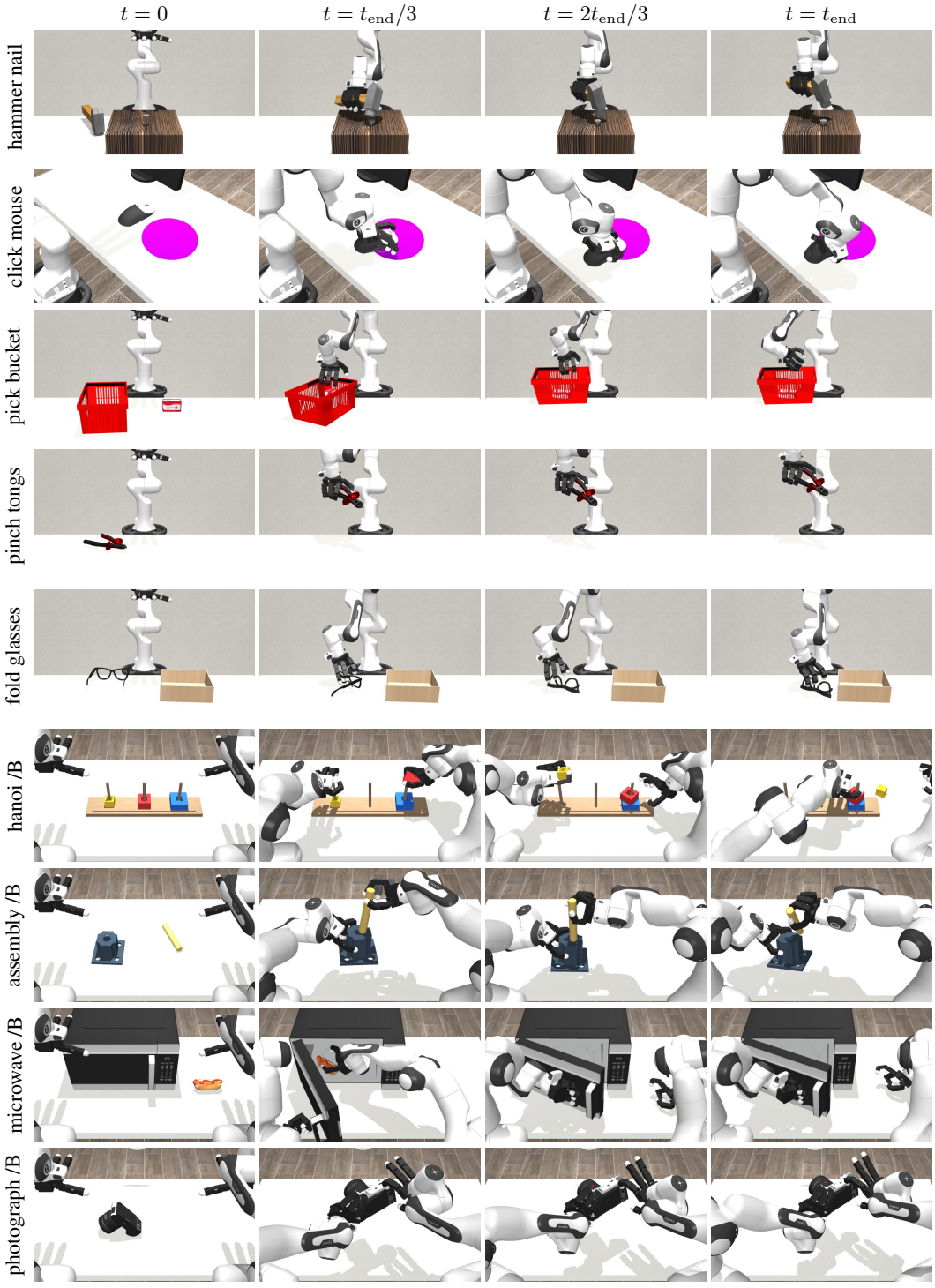


Figure 10: **Failed PointWAM rollouts on DexJoCo.** One failed rollout per task from the multi-task policy, the first failure among the same rollouts as Figure 9, shown at four points up to its own t_{end}. Water plant has no row: all 16 of its rollouts (seeds 0–15) succeed.

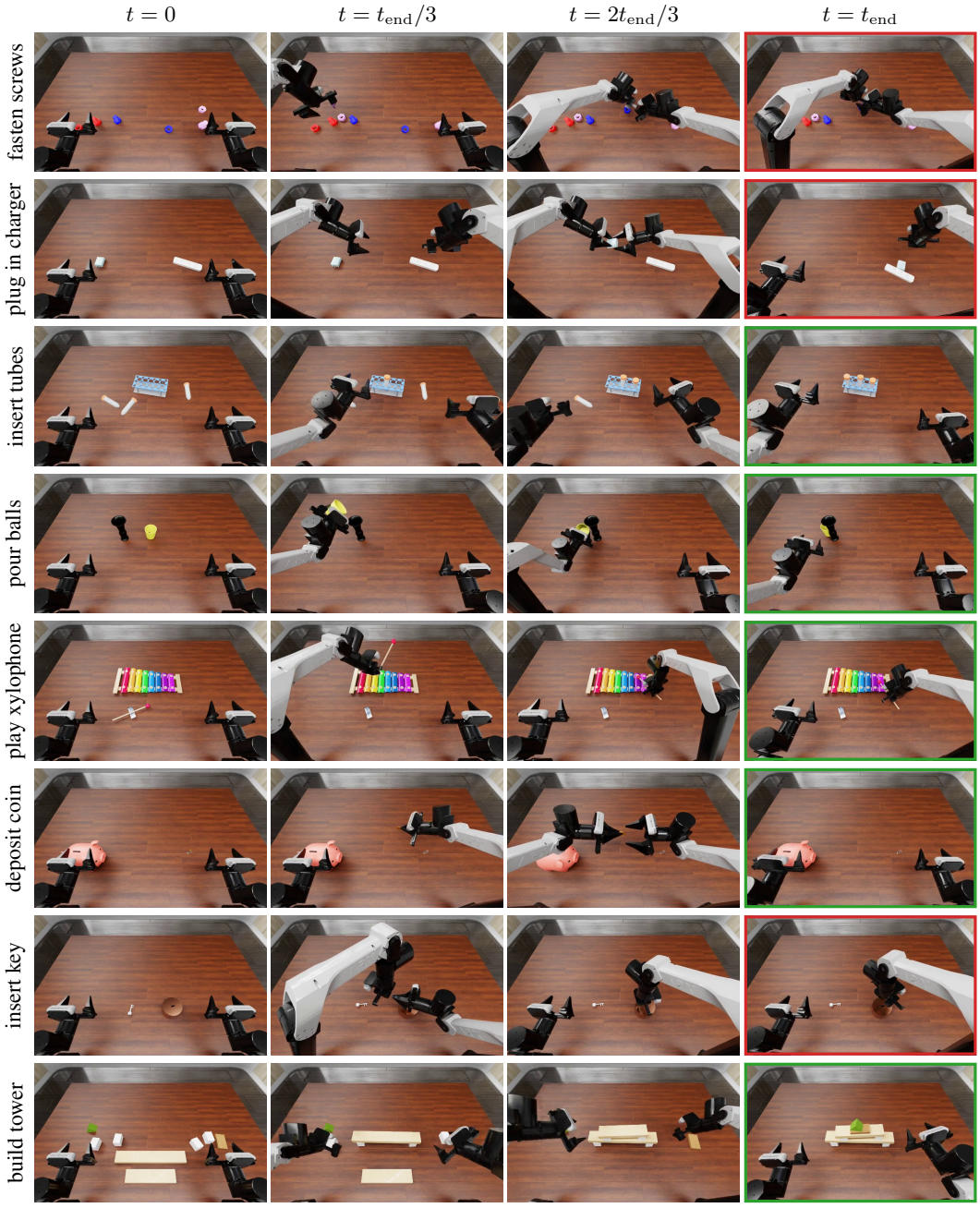


Figure 11: **PointWAM rollouts on RoboDojo-Precision.** One rollout per task, shown at four points up to its own t_{end}: a successful one (green border) for the five tasks PointWAM solves at least once, and a failed one (red border) for the other three tasks.

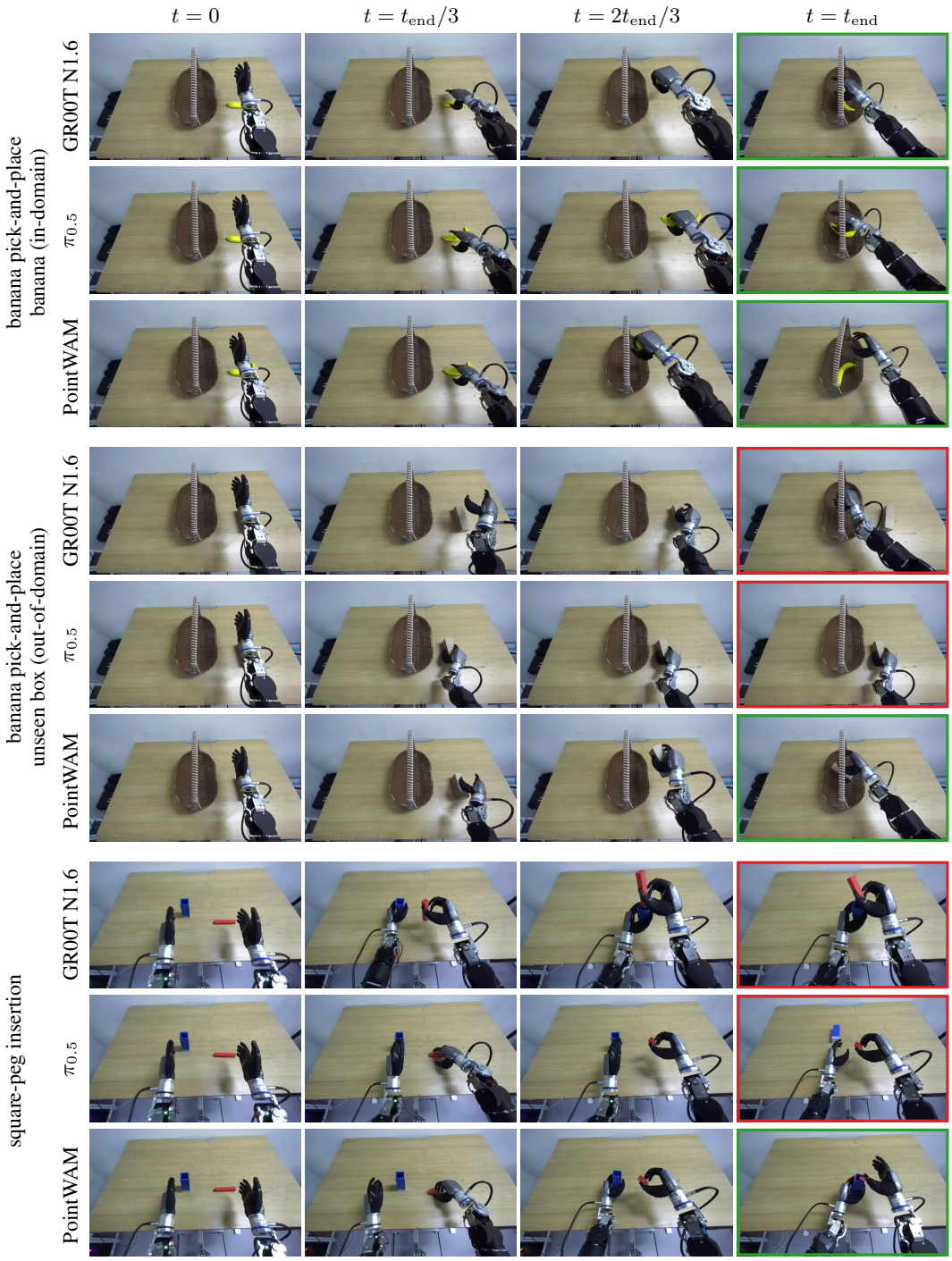


Figure 12: **Real-robot rollouts.** GR00T N1.6, $\pi_{0.5}$, and PointWAM on the two tasks, each shown at four points up to its own t_{end} with a green border for success and a red border for failure. Banana pick-and-place shows one trial each with the in-domain banana and the out-of-domain unseen box.