HoloBrain-0.5 Technical Report

Xuewu Lin, **Jiahao Wang**, **Jiawei Li**, **Mengdi Li**, **Zhengmao Sun**, **Yiwei Jin**, **Hongyu Xie**, **Zhengyang Feng**, **Kun Wu**, **Zihao Chen**, **Lianzhou Zhang**, **Lichao Huang**, **Zhizhong Su***

Horizon Robotics HoloBrain Team
*Corresponding author.

We present HoloBrain-0.5, a family of vision-language-action (VLA) models for manipulation across heterogeneous embodiments. Our goal is to plan motion in a shared action space across embodiments, using the robot state, kinematic structure, and calibrated camera observations as inputs. Building on HoloBrain-0, we introduce three improvements. First, we standardize joint axes, coordinate frames, and tool-center points in each robot description (URDF), then encode its kinematic graph in the policy. Second, we release HoloBrain-0.5-Continuous, a diffusion policy, and HoloBrain-0.5-Discrete, which quantizes actions with residual finite scalar quantization (FSQ) and generates the resulting codes by delayed autoregression. Both use the same 3D embodiment representation, which accommodates arbitrary kinematic configurations. Third, we scale pretraining to more than 10,000 hours of real-robot, simulation, and human-manipulation data spanning 15 embodiment categories, with calibrated camera geometry and canonicalized embodiment representations.

During pretraining, the discrete policy converges faster than the continuous policy and achieves a higher final mean success rate across six closed-loop simulation settings (56.77% versus 49.71%). On seven real-robot tasks, the discrete policy achieves a mean progress score of 87.76% without task-specific post-training. After post-training, the continuous policy outperforms $\pi_{0.5}$ on average on both the seven-task benchmark (94.52% versus 86.25%) and an 11-task dual-arm benchmark (79.07% versus 69.41%). The discrete-initialized continuous policy also transfers five tasks zero-shot from Piper to Piper-X, scoring 63.28% on average without demonstrations of those tasks on Piper-X. With target-robot demonstrations, the continuous policy reaches mean progress scores of 89.2% on a single-arm Franka Panda and 92.94% on a mobile dual-arm platform. Although pretraining includes no Unitree G1 data, post-training with a limited set of G1 demonstrations yields a mean progress score of 70.24% across three humanoid tasks, compared with 11.00% for $\pi_{0.5}$.

Code: HoloBrain | UniURDF
Model: Model weights on Hugging Face
Webpage: `horizonrobotics.github.io/robot_lab/holobrain_0_5`
Correspondence: Zhizhong Su at `zhizhong.su@horizon.auto`



1 Introduction

Vision-language-action (VLA) models ground visual observations and language instructions in executable robot actions, providing a scalable route toward general-purpose manipulation. Early large-scale systems established the value of vision-language pretraining and heterogeneous robot data [1, 2, 3], while recent models have improved action generation, open-world generalization, and cross-embodiment learning [4, 5, 6, 7, 8, 9].

A central challenge is that robot datasets differ not only in visual appearance and task distribution, but also in joint count, kinematic structure, camera configuration, and action conventions. Arbitrary coordinate choices in robot descriptions further obscure the relationship between an action value and its physical effect. HoloBrain-0 addressed part of this problem with a joint-centric architecture and explicit forward-kinematics features [10]. HoloBrain-0.5 extends this design with a canonical embodiment representation and two complementary action formulations. An overview is shown in Figure 1.

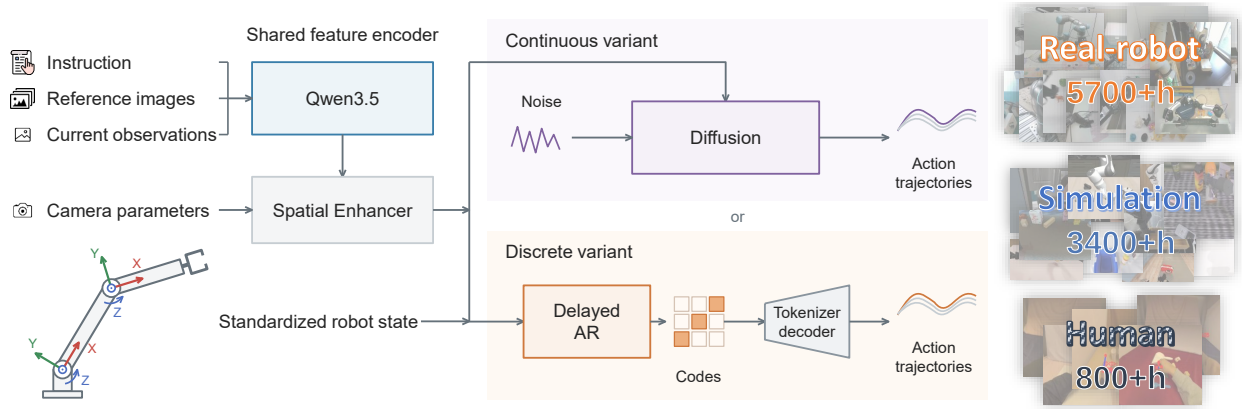


Figure 1: Overview of HoloBrain-0.5. Shared multimodal features and standardized robot state condition either the continuous diffusion policy or the discrete delayed-autoregressive policy with residual-FSQ tokenizer decoding. Both produce action trajectories. Standardized robot state includes embodiment geometry. The illustrated joint frames use $+x$ along the child link and outward $+z$ as the rotation axis; circled dots denote axes pointing out of the page. Pretraining combines real-robot, simulation, and human-manipulation data, illustrated at right. Robot geometry, noise, trajectories, and the code grid are schematic.

We introduce URDF standardization to remove embodiment-specific coordinate conventions before policy learning. Revolute joints are reparameterized about their local z-axes, prismatic joints translate along the positive local z-axis, and tool-center-point (TCP) frames point outward along z.

HoloBrain-0.5 provides two model variants under a largely shared architecture. HoloBrain-0.5-Continuous models continuous action chunks with diffusion, whereas HoloBrain-0.5-Discrete applies finite scalar quantization (FSQ) [11] in successive residual stages and multidimensional delayed autoregressive generation. The discrete formulation exposes an explicit token distribution and converges faster in our pretraining experiments. We pretrain both variants on more than 10,000 hours of data across 15 embodiment categories.

We track closed-loop performance at intermediate pretraining checkpoints using six simulation settings from three simulators, with a total of 21,430 evaluation episodes per model checkpoint across the six settings. We additionally evaluate out-of-the-box behavior, real-robot post-training, zero-shot task transfer between seen embodiments, and adaptation across substantially different robot morphologies. The first two real-robot benchmarks contain 18 tasks in total, while the additional morphology-adaptation suite spans 11 tasks across three further embodiments.

Our main contributions are summarized as follows:

- We introduce a canonical 3D embodiment representation and a shared action space based on standardized URDFs. Both are defined on the kinematic graph rather than a fixed set of joints.
- We release two policies built on this representation: the discrete policy converges faster during pretraining, while the continuous policy performs better after post-training.
- In real-robot trials, HoloBrain-0.5 improves on HoloBrain-0 in closed-loop performance and performs complex dexterous tasks. It achieves 63.28% mean progress on tasks transferred zero-shot to a target robot and, with target-robot demonstrations, adapts to single-arm, humanoid, and mobile platforms.

2 Related Work

2.1 Vision-Language-Action Models

RT-1 and RT-2 established scalable vision-language-conditioned robot control and transfer from web-scale representations [1, 2]. Open X-Embodiment [3], Octo [12], and OpenVLA [13] subsequently demonstrated the value of heterogeneous multi-robot pretraining. Recent systems pair large vision-language backbones with dedicated action modules: RDT-1B [14], π_0 [4], CogACT [15], and GR00T N1 [16] use diffusion- or flow-based generation, while $\pi_{0.5}$ [5] and $\pi_{0.7}$ [6] emphasize heterogeneous co-training and contextual conditioning.

Cross-embodiment scaling has also been explored through embodiment prompts or learned action representations in X-VLA [17], LingBot-VLA [9], Wall-OSS-0.5 [7], and G0.5 [8]. In parallel, RoboMamba [18], TinyVLA [19], NORA [20], SmolVLA [21], and VLA-Adapter [22] study more efficient VLA architectures. CEI [23] transfers demonstrations by aligning trajectories across functionally similar end effectors in 3D. HoloBrain-0.5 instead standardizes joint axes, coordinate frames, and end-effector conventions so that demonstrations from different robots can share an action representation.

2.2 Action-Space Modeling

Action-space design determines how a policy represents temporal structure and multimodality. Continuous policies use direct action chunks [24] or generative objectives such as diffusion and flow matching [25, 14, 4, 15, 16]. Discrete policies instead quantize actions and predict token sequences. OpenVLA [13] uses discretized action tokens, FAST [26] compresses action chunks in the frequency domain, and SpatialVLA [27] and G0.5 [8] introduce spatially or embodiment-aware tokenizations. FSQ [11] offers a codebook-free mechanism for constructing compact discrete representations. VQ-BeT [28] learns a hierarchy of discrete action codes through residual vector quantization.

Both HoloBrain-0.5 variants use the same canonical 3D joint graph to represent the embodiment and express actions as temporal sequences of joint graphs. This graph provides a complete kinematic representation of robot state and action, grounding both in the same 3D frame as calibrated visual observations. The continuous variant models these sequences with diffusion. The discrete variant uses residual FSQ to learn a discrete representation of the action prior from trajectories, then predicts the resulting codes by delayed autoregression.

3 Method

HoloBrain-0.5 retains the joint-centric policy interface of HoloBrain-0 while revising both the embodiment representation and action decoder. At control step h, the policy takes a language instruction $\boldsymbol{\tau}$, synchronized camera observations $\mathcal{I}_h = \{I_h^{(m)}\}_{m=1}^{M}$, the numerical robot state $\mathbf{s}_h$, the standardized URDF $\mathcal{U}$, and camera calibration parameters $\mathcal{C} = \{K_m, T_{BC_m}\}_{m=1}^{M}$, where K_m and T_{BC_m} are the intrinsic matrix and robot-base extrinsic transform of camera m, respectively. It predicts a future action chunk $\mathbf{A}_{h:h+H}$ according to

$$\mathbf{A}_{h:h+H} \sim \pi_\theta(\cdot \mid \mathcal{I}_h, \boldsymbol{\tau}, \mathbf{s}_h, \mathcal{U}, \mathcal{C})\,. \tag{1}$$

Unlike a conventional VLA interface, HoloBrain explicitly supplies the embodiment description and calibrated camera geometry. The URDF is converted into the canonical joint graph used for action modeling, while camera calibration aligns visual observations with the robot-base frame. Section 3.1 describes this standardization, followed by the continuous and discrete decoders and the pretraining and calibration pipeline in Sections 3.2–3.4.

3.1 URDF Standardization

HoloBrain constructs its embodiment representation from a pruned URDF: it computes forward-kinematics (FK) losses over the retained kinematic tree, uses the 6-DoF poses of selected links to construct the joint-centric action space, and embeds the graph-hop distances among these links [10]. Consequently, URDF frame conventions directly enter the learned representation. Source URDFs may assign different local axes to physically equivalent joints, while manually selected representative links may be offset from the corresponding joint origins. We standardize each embodiment using a common, joint-centered convention, as illustrated in Figure 2.

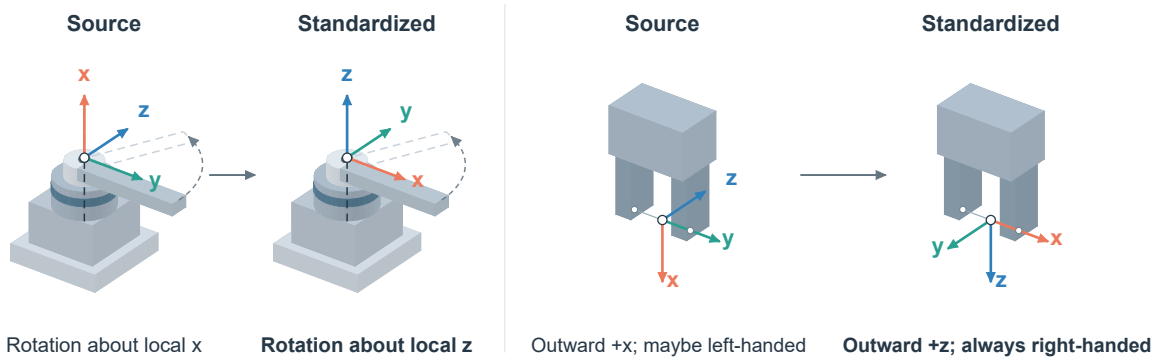
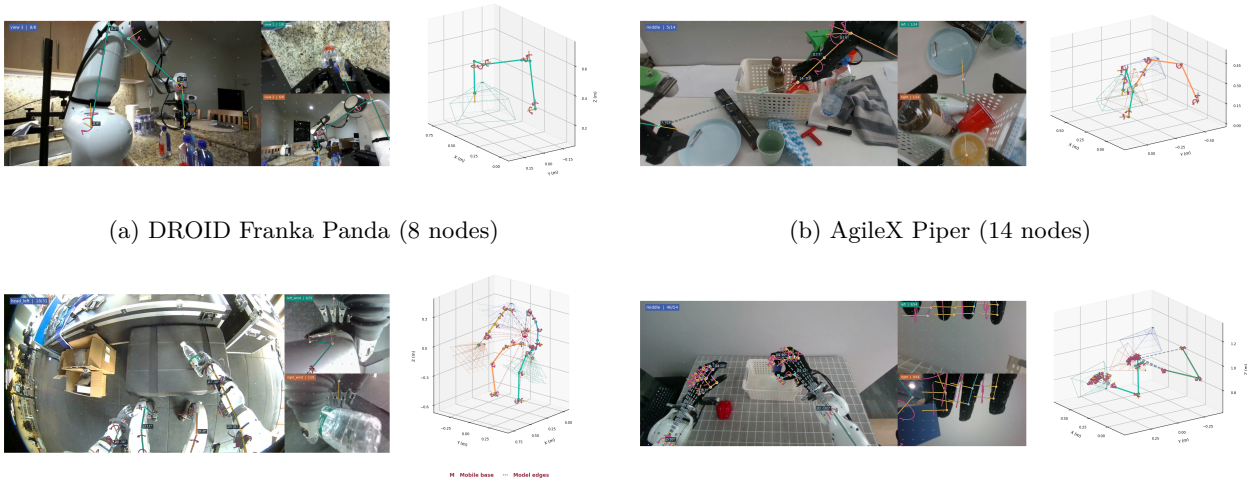


Figure 2: Joint and TCP frame standardization. (a) The joint's right-handed source frame is reoriented from local x-axis motion to local z-axis motion. Attached-transform compensation preserves physical geometry and joint trajectories; dashed geometry shows a second configuration. (b) A schematic left-handed dataset TCP frame is converted to a right-handed frame with outward $+z$. Handedness conversion is separate from the proper rotation in (a). The TCP origin is the midpoint of the two marked fingertip centers. Axis colors denote x (red), y (green), and z (blue). Source conventions and geometry are illustrative; wrist roll is set per embodiment.

We canonicalize every actuated joint and use the direct child link of each selected joint as its pose anchor. For joint j, let P_j and C_j denote its parent and direct child links, q_j its joint value, and $\mathbf{a}_j$ its original unit motion axis. We write T_{AB} for the transform from frame B to frame A, with $T_{o,j} = T_{P_jC_j}(0) \in SE(3)$ denoting the joint origin. We choose a proper rotation $S_j \in SO(3)$ satisfying $S_j\hat{\mathbf{z}} = \mathbf{a}_j$, where $\hat{\mathbf{z}} = [0, 0, 1]^\top$. We then post-rotate the joint origin by S_j and replace its axis with $\hat{\mathbf{z}}$; this preserves the origin translation while expressing revolute, continuous, and prismatic motion about or along the positive local z-axis. Axis alignment leaves the roll around z unconstrained, so a manually specified `rot_z` selects a consistent wrist-frame roll.

Figure 3 shows how the same action representation applies to a single arm, two dual-arm robots, and a humanoid. Each panel pairs camera projections with the same graph in the robot-base frame: node counts and camera layouts change, but a joint pose has the same geometric meaning across views and embodiments. Appendix C gives further examples, including mobile-base and TCP-only interfaces.



(a) DROID Franka Panda (8 nodes)

(b) AgileX Piper (14 nodes)

(c) Unitree G1 (31 nodes)

(d) Tianji WuJi (54 nodes)

Figure 3: Canonical joint graphs on four robots. Each panel shows the physical nodes projected into the available camera views (left) and expressed in the robot-base frame (right).

This frame rewrite changes the model-facing orientation of C_j but not the robot's physical behavior. Let $\widetilde{S}_j \in SE(3)$ be the homogeneous lift of S_j, and let $\mathcal{M}(\mathbf{u}, q)$ denote revolute or prismatic motion about or along axis $\mathbf{u}$. The rewrite and compensation are

$$\begin{aligned} \mathcal{M}(\mathbf{a}_j, q_j) &= \widetilde{S}_j \mathcal{M}(\hat{\mathbf{z}}, q_j) \widetilde{S}_j^{-1}, \\ T'_{o,j} = T_{o,j} \widetilde{S}_j, \qquad & T'_{C_j X} = \widetilde{S}_j^{-1} T_{C_j X}, \end{aligned} \tag{2}$$

where X denotes a child-local visual, collision, inertial, or outgoing-joint frame. It follows that $T'_{P_j C_j}(q_j) = T_{P_j C_j}(q_j) \widetilde{S}_j$, whereas

$$T'_{P_j X}(q_j) = T'_{P_j C_j}(q_j) T'_{C_j X} = T_{P_j C_j}(q_j) T_{C_j X} = T_{P_j X}(q_j). \tag{3}$$

Thus, the child-link frame acquires the desired canonical orientation, but attached geometry, inertial frames, and downstream kinematics retain their original poses for every q_j. Recorded joint trajectories therefore remain valid. If a camera extrinsic references a reoriented link, we add a fixed compatibility frame that recovers the original reference frame.

We complete the representation by standardizing end-effector frames and pruning to the model-relevant graph. Each embodiment defines an outward-facing TCP $+z$ and a wrist-roll `rot_z` once for all trajectories. We retain selected joints' child links, the declared TCP, gripper, and camera-reference links, and their root paths; other branches are deleted without changing retained transforms. HoloBrain evaluates FK on this pruned URDF, uses the selected child-link poses as its 6-DoF action representation, and embeds their hop distances.

We apply this procedure to all 15 embodiments. For each standardized URDF, we compare shared-link positions and visual-geometry poses with the source description at the zero configuration and at two deterministic nonzero configurations, ensuring that the check covers joint motion rather than only the rest pose. This process removes arbitrary frame conventions while preserving the physical semantics of the original trajectories.

3.2 HoloBrain-0.5-Continuous

The continuous variant follows the joint-centric encoder-decoder design and diffusion objective of HoloBrain-0 [10]. Unless stated otherwise, the input representation, forward-kinematics conditioning, action parame-

terization, diffusion training objective, and inference procedure remain unchanged. This section focuses on the architectural modifications introduced in HoloBrain-0.5.

Backbone and model capacity. We replace the Qwen2.5-VL-3B backbone [29] with Qwen3.5-2B [30] and increase the capacity of the action decoder. Specifically, the number of decoder layers is increased from 6 to 10, while the embedding dimension is increased from 256 to 384.

Parallel multimodal attention. HoloBrain-0 applies image cross-attention, text cross-attention, and temporal-joint self-attention sequentially. We find that this ordering can bias the decoder toward the last processed modality. In particular, placing temporal-joint self-attention last encourages the policy to over-rely on proprioception and become less responsive to visual observations. HoloBrain-0.5 therefore computes the three attention branches in parallel and combines them with a query-dependent routing mechanism, as shown in Figure 4.

This parallel design preserves modality-specific positional structures rather than imposing a shared positional representation across all branches.

Let $\mathbf{x}$ denote the action-query features, and let $\mathbf{C}_v$ and $\mathbf{C}_t$ denote the image and language features, respectively. We use three independently parameterized attention modules. Their outputs can be summarized as

$$\begin{aligned} \mathbf{x}_v &= \mathrm{Attn}_v(\mathbf{x}, \mathbf{C}_v^{\mathrm{3D}}), \\ \mathbf{x}_t &= \mathrm{Attn}_t(\mathbf{x}, \mathrm{RoPE}_{\mathrm{1D}}(\mathbf{C}_t)), \\ \mathbf{x}_s &= \mathrm{Attn}_s(\mathrm{JointRelative}(\mathrm{RoPE}(\mathbf{x}))), \end{aligned} \tag{4}$$

Here, $\mathbf{C}_v^{\mathrm{3D}}$ denotes image features equipped with 3D absolute positional encoding, $\mathrm{RoPE}_{\mathrm{1D}}$ denotes one-dimensional rotary positional encoding [31] for language features, and $\mathrm{JointRelative}(\mathrm{RoPE}(\mathbf{x}))$ denotes temporal rotary positional encoding combined with joint-relative positional encoding for the temporal-joint branch. The notation suppresses the query/key/value projections and residual connections inside each attention module.

The three branch outputs are combined by a query-dependent router. The router predicts a weight for each modality, and the final output is their weighted sum:

$$\boldsymbol{\alpha} = \mathrm{softmax}\left(\frac{\mathrm{Router}(\mathbf{x})}{\tau}\right), \qquad \mathbf{x}_{\mathrm{out}} = \alpha_v \mathbf{x}_v + \alpha_t \mathbf{x}_t + \alpha_s \mathbf{x}_s, \tag{5}$$

where $\boldsymbol{\alpha} = (\alpha_v, \alpha_t, \alpha_s)$ and τ is the routing temperature. In the implementation, the routing weights are predicted independently for different attention heads; the head index is omitted here for readability.

Reference images from task-level key frames. During pretraining, we randomly provide reference images for a subset of training samples. In many tasks, the textual instructions alone do not fully specify the task objective, the relevant scene configuration, or the desired execution process. As a result, the model may receive insufficient conditioning information to learn the intended visual-to-action mapping. To provide more informative task conditions and support in-context learning, we annotate task-level key frames for a subset of tasks. During pretraining, a list of key frames is randomly sampled and the corresponding images are provided as reference images. Examples are shown in Figure 5.

Modified joint-relative position encoding. Joint self-attention relies on a joint-relative position matrix, whose entries measure the kinematic separation between joint pairs. In HoloBrain-0, these entries are hop counts on the kinematic graph: adjacent joints within an arm are one hop apart, and the two arms of a dual-arm robot are connected at their bases by a single unit-hop edge. The two end effectors are therefore separated by nearly the combined length of both chains, even though their actions may be strongly correlated. HoloBrain-0.5 instead connects the two arm ends with a single unit-hop edge, so every cross-arm distance becomes the shortest path from one joint to the end of its own arm, across this edge, and on to the other joint, while the within-arm kinematic topology is preserved. The original and modified matrices are compared in Figure 6.

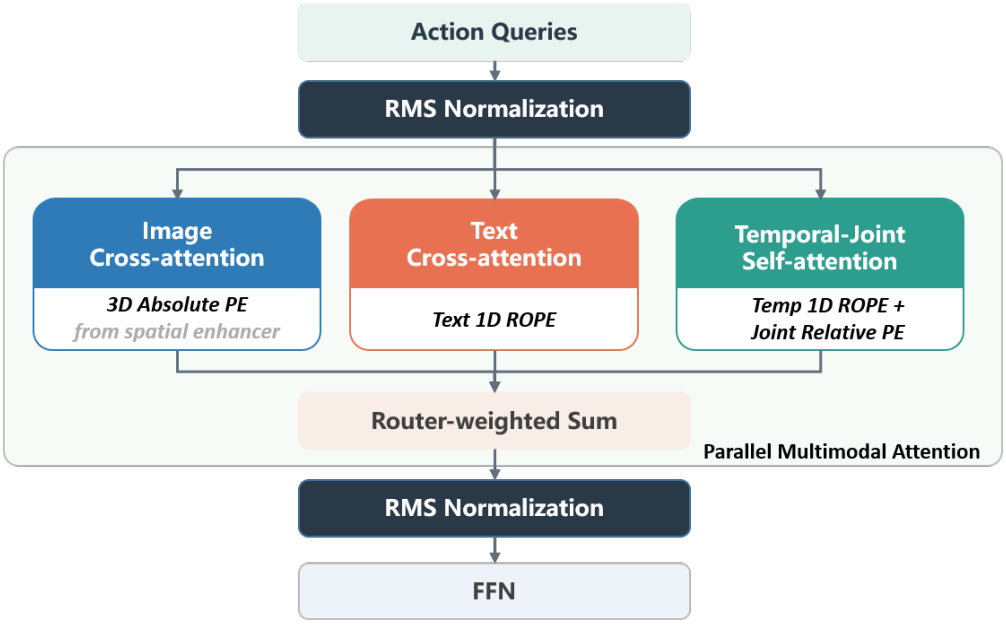


Figure 4: Parallel multimodal attention in the HoloBrain-0.5 action decoder. Image, text, and temporal-joint features are processed by independently parameterized attention branches and combined by a query- and head-dependent router through weighted summation.

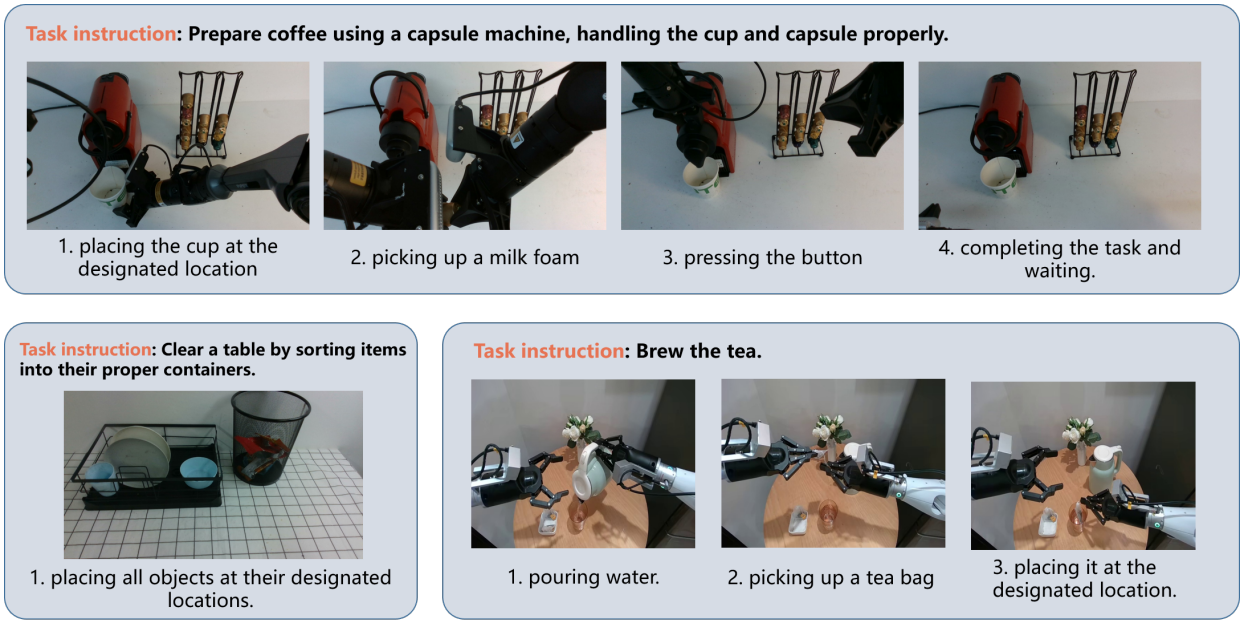
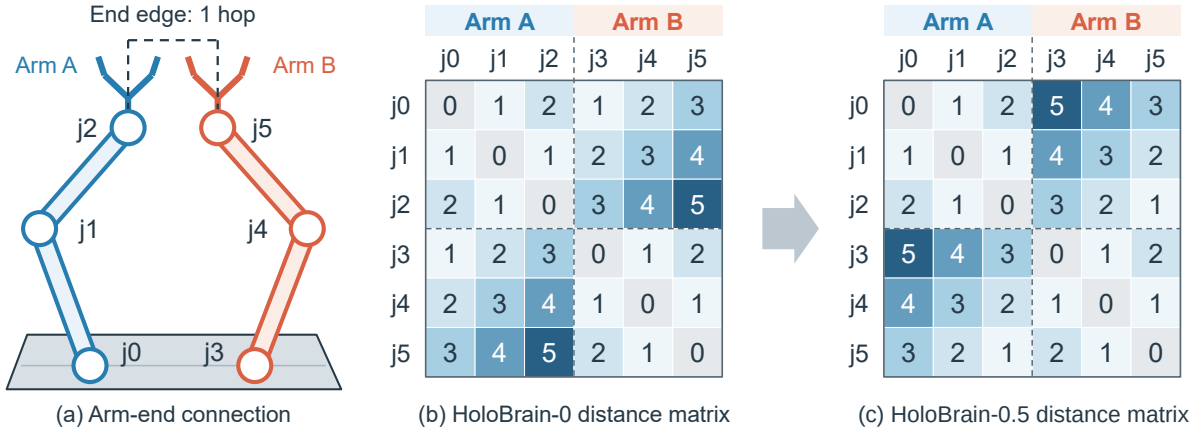


Figure 5: Reference images sampled from task-level key frames. (a, top) Preparing coffee: place the cup at the designated location, retrieve the milk-foam capsule, press the button, and wait for completion. (b, bottom left) Place all objects at their designated locations and finish. (c, bottom right) Brew tea: pour water, pick up a tea bag, and place it in the cup.



(a) Arm-end connection (b) HoloBrain-0 distance matrix (c) HoloBrain-0.5 distance matrix

Figure 6: Joint-relative position encoding for an illustrative dual-arm embodiment. Entries give shortest-path hop counts on two three-joint chains with unit within-arm edges. (a) The dashed connection joins j_2 and j_5 in HoloBrain-0.5. HoloBrain-0 (b) joins j_0 and j_3 with one hop, whereas HoloBrain-0.5 (c) replaces this edge with the one-hop j_2–j_5 connection. The distal-joint distance changes from five hops to one; within-arm distances are unchanged.

3.3 HoloBrain-0.5-Discrete

HoloBrain-0.5-Discrete uses the same canonical joint-centric representation and multimodal conditioning interface as the continuous variant, but replaces direct action generation with categorical prediction. Training is divided into two stages. We first learn a morphology-aware action tokenizer that compresses continuous trajectories into discrete codes and reconstructs them without image or language input. We then freeze the tokenizer and train the VLA policy to predict its codes from visual observations, language instructions, and proprioceptive state. Because tokenizer parameters are shared and code indices come from a fixed alphabet, every embodiment's trajectories map into the same categorical output space even though the number of modeled joints or links varies.

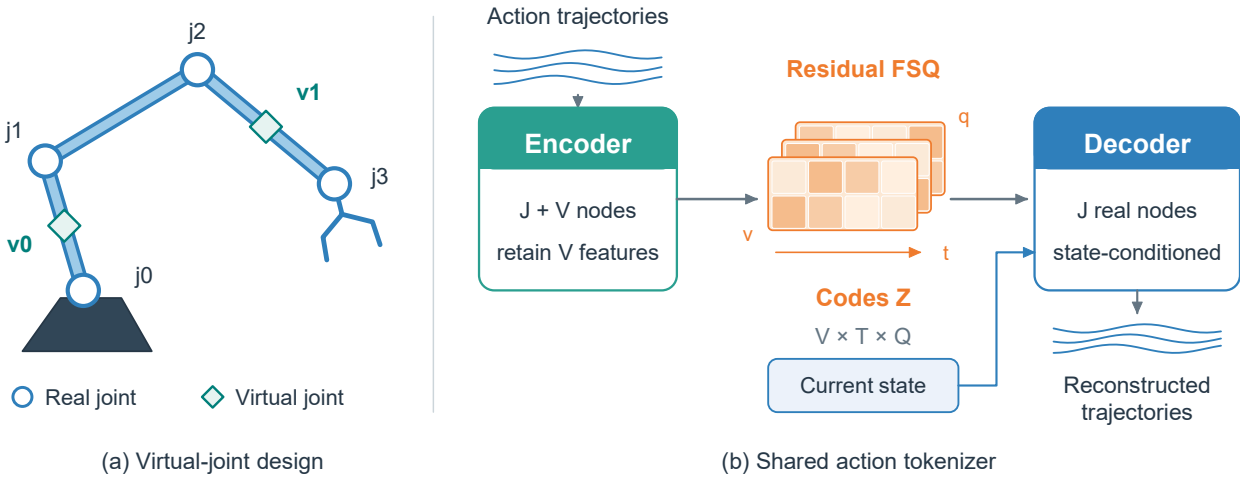
Virtual joints. Let $\mathbf{A} \in \mathbb{R}^{H\times J\times d}$ be a future action trajectory, where H is the prediction horizon and J is the embodiment-dependent number of represented nodes. Each node contributes d channels: a joint or gripper scalar and the pose of its link. The trajectory is divided into T non-overlapping temporal patches. To summarize trajectories using their graph structure, we introduce latent aggregation nodes, termed *virtual joints*, and place them on the kinematic graph (Figure 7). For a motion chain c with n_c declared link and finger entries, we allocate

$$V_c = \lfloor n_c/\rho \rfloor + 1, \qquad V = \sum_c V_c, \tag{6}$$

where ρ controls compression; all tokenizer parameters are shared across these embodiment-dependent counts. Virtual joints are positioned to minimize the maximum distance from a real joint to its nearest virtual joint (Appendix A.1). Each virtual joint's graph position determines its distances to the real nodes and to the other virtual joints. Adding these entries to the hop distances among the real nodes (Figure 6) yields the augmented matrix $\Delta \in \mathbb{R}^{(J+V)\times(J+V)}$, which conditions the tokenizer's attention.

Graph aggregation and discretization. Each temporal patch of a real-node trajectory is embedded into a feature. We append V virtual nodes initialized from a shared learned vector and process the $J+V$ node streams with graph attention over the augmented distances Δ. The attention itself is inherited from HoloBrain-0 [10]: a sinusoidal embedding and MLP map each distance Δ_{ij} to a channel-wise gate that modulates query–key interactions, with temporal rotary encoding supplying time dependence. We retain the virtual-node features $\mathbf{H} \in \mathbb{R}^{V\times T\times C}$ as the compression bottleneck, where C is the feature width.

We apply finite scalar quantization [11] to each bottleneck feature $\mathbf{H}_{vt}$ through Q residual stages: the first stage quantizes the feature, and each subsequent stage quantizes the remaining residual at a successively



(a) Virtual-joint design

(b) Shared action tokenizer

Figure 7: Virtual-joint design and action tokenization. (a) Virtual nodes share a learned initialization and obtain graph-relative identities from their positions on the kinematic graph. (b) The encoder processes real and virtual nodes jointly and retains the virtual-node features. Residual FSQ produces a code grid indexed by virtual joint v, temporal patch t, and residual stage q. State-conditioned decoding and restoration reconstruct action trajectories. Network parameters are shared across embodiment-dependent node counts. Trajectories and grid dimensions are schematic; implementation settings appear in Appendix A.4.

finer scale. The indices emitted by the Q stages form the code grid $\mathbf{Z} \in \{0, \ldots, K-1\}^{V \times T \times Q}$, indexed by virtual joint v, temporal patch t, and residual stage $q = 0, \ldots, Q-1$, where K is the number of indices per stage. Appendix A.2 details the quantization configuration and implementation.

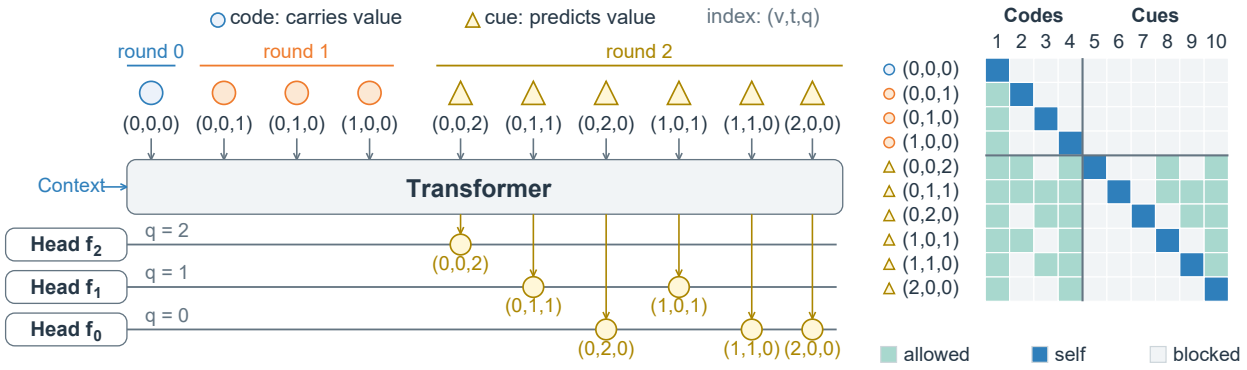
Reconstruction. Real-node queries initialized from the current proprioceptive state attend to the quantized virtual features. The decoder upsamples T patches to H frames and predicts changes in joint values and link poses relative to the current state. Applying these changes to that state recovers absolute joint and link-pose trajectories. Gripper openings are predicted directly as absolute values. Training every residual prefix encourages earlier stages to support coarse reconstruction and later stages to refine it. Let $\widehat{\mathbf{A}}^{(r)}$ be the reconstructed trajectory using the first r stages. Tokenizer training minimizes

$$\mathcal{L}_{\text{tok}} = \sum_{r=1}^{Q} \lambda_r \, \mathcal{L}_{\text{rec}}\left(\widehat{\mathbf{A}}^{(r)}, \mathbf{A}\right), \tag{7}$$

where λ_r weights the reconstruction loss of the r-stage prefix, and $\mathcal{L}_{\text{rec}}$ compares joint scalars, positions, and rotations, with forward-kinematics supervision and pose-consistency terms where kinematics are available (Appendix A.3). Figure 7(b) summarizes the tokenizer's structure.

Code/cue delayed autoregression. The tokenizer represents each action chunk with codes for multiple virtual joints, temporal patches, and residual stages. Generating every code serially preserves all dependencies but requires VTQ sequential policy evaluations; predicting them all in one pass removes this cost but cannot condition on code values sampled elsewhere in the same chunk. We instead predict groups of codes together, using permitted earlier predictions as context, following the delayed-codebook principle of [32] (Figure 8).

Each cell $i = (v, t, q)$ occupies two positions in the transformer sequence: a *code position* carrying its value Z_i, and a *cue position* from which that value is predicted. Their initial features are $\mathbf{h}_i^{\text{code},0} = e(Z_i) + e_q^{\text{code}}$ and $\mathbf{h}_i^{\text{cue},0} = e_q^{\text{cue}}$, where e is a learned embedding lookup for code values and e_q^{code}, e_q^{cue} are learned embeddings for residual stage q. Temporal rotary encoding identifies the patch, and graph distances identify the virtual joint's relationships to other nodes. The output feature at each cue gives $p_{\theta,i} = \text{softmax}(f_q(\mathbf{h}_i^{\text{cue}}))$, where f_q is the classification head for residual stage q, shared across virtual joints and patches.



(a) Code/cue generation (b) Visibility mask

Figure 8: Code/cue delayed generation. (a) Round-2 snapshot of a grid with $V, T, Q \geq 3$ and $\delta = t + v + q$: four known codes (circles, rounds 0–1) and six prediction cues (triangles, round 2). Each cue uses its residual-stage head f_q, shared across virtual joints and patches. (b) Visibility among these positions: rows query columns; teal allows attention, blue marks self-attention, and gray blocks it. Positions 1–4 carry codes; 5–10 are cues whose target values remain unavailable. Figure 9 decomposes the mask.

We advance diagonally through the code grid: cells whose virtual-joint, temporal-patch, and residual-stage indices have the same sum are predicted together, and each such diagonal defines a delay step. Using zero-based indices, we assign each cell a delay δ_i and collect the cells predicted at step ℓ in $\mathcal{G}_\ell$; a one-way mask B additionally prevents conditioning on a later time or a finer residual stage even when its delay is smaller:

$$\delta_i = t + v + q, \quad \mathcal{G}_\ell = \{i : \delta_i = \ell\}, \quad B(i,j) = \mathbf{1}[t' \leq t]\,\mathbf{1}[q' \leq q], \tag{8}$$

where $j = (v', t', q')$ is a key cell. Let $\mathcal{P}(i) = \{j : \delta_j < \delta_i,\ B(i,j) = 1\}$ denote the permitted earlier code positions. Given the multimodal context $\mathbf{c} = (o_h, i_h, s_h)$, comprising the observation, instruction, and current proprioceptive state at control step h, the policy factorizes as

$$p_\theta(\mathbf{Z} \mid \mathbf{c}) = \prod_{\ell} \prod_{i \in \mathcal{G}_\ell} p_\theta\big(Z_i \mid \mathbf{Z}_{\mathcal{P}(i)}, \mathbf{c}\big)\,. \tag{9}$$

Each code or cue position at i reads $\mathcal{P}(i)$ and itself; cues also read same-step cues under B.

Figure 9 decomposes the round-2 visibility of Figure 8(b) into its role/delay, time, and residual components. The schedule therefore generates VTQ codes in $T + V + Q - 2$ rounds while retaining causal dependencies across temporal patches, virtual-joint summaries, and residual quantization stages.

With the tokenizer frozen, the action expert is trained by cross-entropy over the ground-truth grid (Appendix A.3),

$$\mathcal{L}_{\mathrm{AR}} = -\sum_{i} \log p_\theta\big(Z_i \mid \mathbf{Z}_{\mathcal{P}(i)}, \mathbf{c}\big)\,. \tag{10}$$

At inference, cues predict each round's codes from codes sampled in earlier rounds. The completed code grid is mapped back to quantized features by residual FSQ, and the state-conditioned tokenizer decoder reconstructs a continuous action chunk.

3.4 Data Recipe

Compared with HoloBrain-0, we scale the pretraining corpus from approximately 3,500 to 10,006.5 hours and increase its embodiment coverage from 7 to 15 categories. The corpus contains 5,704.3 hours of real-robot data, 3,470.3 hours of simulation, and 831.9 hours of human-manipulation data. Figure 10 and Table 1 summarize its composition.



Figure 9: Factorization of the visibility mask in Figure 8(b). Rows are queries and columns are keys, in the same position order. R applies the code/cue role and delay rules, including self-attention; T admits keys at time $t' \le t$; S admits keys at residual stage $q' \le q$. Their intersection gives the final visibility M.

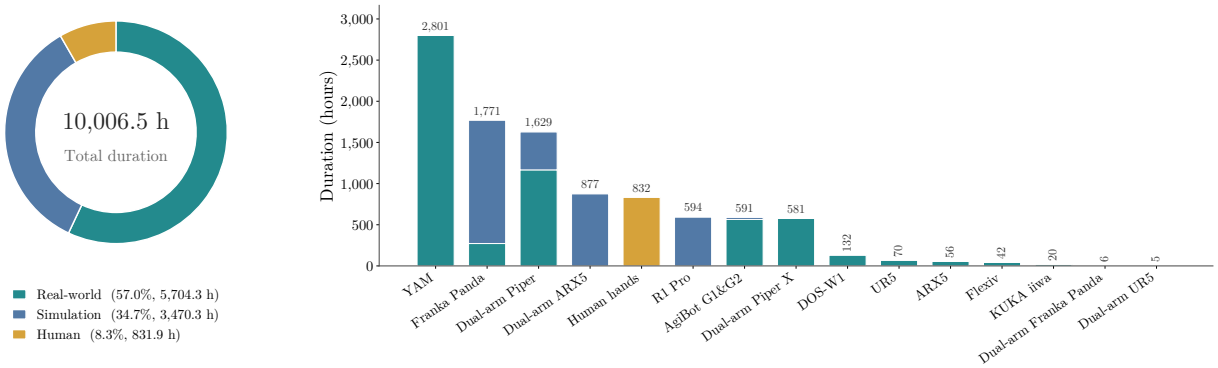


Figure 10: Composition of the HoloBrain-0.5 pretraining corpus. The left plot shows the duration contributed by real-robot, simulation, and human-manipulation data; the right plot breaks the same corpus down by embodiment category.

A key property of the corpus is its consistent 3D geometric metadata. Robot trajectories include transformations between the robot and all cameras together with standardized URDF descriptions. For human-manipulation trajectories, we use tracked 3D hand joints to construct a gripper abstraction and define its pose directions according to the same convention used for robot grippers. This procedure places robot and human-hand data in a consistent 3D action representation. For open-source datasets that lack reliable camera calibration, we recover the missing parameters through manual correspondence annotation followed by bundle adjustment [33].

Annotators mark physically identical landmarks across frames and camera views. Given manually initialized camera parameters, the corresponding 3D landmarks are initialized by multi-view triangulation. Camera intrinsics, camera poses, and landmark positions are then jointly refined by minimizing reprojection error:

$$\min_{\{\boldsymbol{\theta}_c,\, \mathbf{T}_c,\, \mathbf{X}_k\}} \sum_{c,k} \left\| \mathbf{u}_{c,k} - \pi\left(\boldsymbol{\theta}_c, \mathbf{T}_c, \mathbf{X}_k\right) \right\|_2^2, \tag{11}$$

where $\mathbf{u}_{c,k}$ is the annotated image location of landmark k in camera c, $\mathbf{X}_k$ is its 3D position, $\boldsymbol{\theta}_c$ denotes camera intrinsics, $\mathbf{T}_c$ denotes camera extrinsics, and $\pi(\cdot)$ is the projection function. One calibrated camera or robot frame is fixed to remove gauge ambiguity. The optimized camera system is then aligned with the robot coordinate frame using known geometric constraints, and all results are manually checked by multi-view reprojection before entering the training corpus.

Table 1: HoloBrain-0.5 pretraining data by dataset family. FPS is the effective frame rate, rounded to the nearest whole number; family and total durations are rounded independently to the nearest hour.

Data type	Dataset name	Embodiment	FPS	Duration (h)
Real-world	`Horizon dataset`	Dual-arm Piper / Dual-arm Piper X	30	1,653
	`ABC-130K` [34]	YAM	30	2,801
	`AgiBot` [35]	AgiBot G1&G2	30	564
	`DROID` [36]	Franka Panda	14	265
	`RH20T` [37]	Flexiv / Franka Panda / KUKA iiwa / UR5	10	89
	`Table30` [38]	Dual-arm Piper / ARX5 / DOS-W1 / UR5	28	331
Simulation	`Genie Sim 3.0` [39]	AgiBot G1&G2	30	26
	`BEHAVIOR-1K` [40]	R1 Pro	30	594
	`Isaac Pick-Place`	Franka Panda / Dual-arm Piper X	30	1
	`LIBERO` [41]	Franka Panda	20	4
	`RoboCasa` [42]	Franka Panda	20	1,495
	`RoboDojo` [43]	Dual-arm ARX5	25	21
	`RoboTwin` [44, 45]	Dual-arm ARX5 / Dual-arm Franka Panda / Dual-arm Piper / Dual-arm UR5	17	170
	`InternData-A1` [46]	Dual-arm Piper / Dual-arm ARX5	30	1,159
Human	`EgoDex` [47]	Human hands	30	832
		Total	–	**10,007**

4 Experiments

We track closed-loop performance throughout pretraining and evaluate the resulting policies before and after task-specific post-training. Our real-robot evaluation spans five embodiments: Piper, Piper-X, a single-arm Franka Panda, a mobile dual-arm platform, and a Unitree G1 humanoid. We distinguish two forms of cross-embodiment evaluation: transfer without demonstrations of the target task on the target robot, and adaptation using a limited set of demonstrations from the target embodiment.

4.1 Closed-Loop Performance of Pretrained Models

We evaluate each pretrained checkpoint over 21,430 episodes across six simulation settings: three 50-task RoboTwin settings (clean, randomized, and UR5-WSG) [45], LIBERO [41], LIBERO-Plus [48], and RoboCasa [42]. Figure 11 reports the unweighted mean of the six per-setting success rates.

After pretraining, HoloBrain-0.5-Continuous achieves a mean simulation success rate of 49.71%, while HoloBrain-0.5-Discrete reaches 56.77%. The discrete variant reaches 47.34% at 50,000 steps and remains above the continuous curve at the plotted checkpoints.

We further train an additional continuous variant by initializing its shared components with the parameters of HoloBrain-0.5-Discrete and replacing its discrete action head with a diffusion head. We denote this model HoloBrain-0.5-Continuous (D-Init.). This variant achieves higher success rates than the continuous baseline at the plotted checkpoints, suggesting that discrete-action pretraining can provide a useful initialization for continuous action generation.

In addition to simulation-based evaluation, we conduct out-of-the-box real-robot experiments with all three pretrained models. The corresponding results are summarized in Table 2.

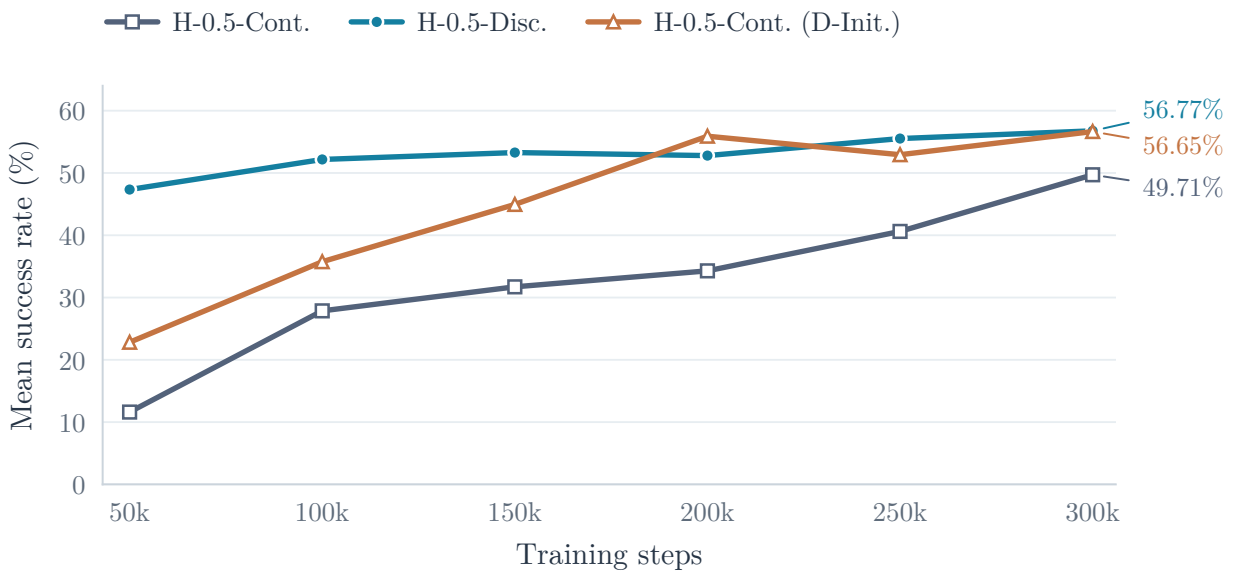


Figure 11: Closed-loop simulation performance throughout pretraining. Each model checkpoint is evaluated over 21,430 episodes in total across six simulation settings spanning three simulators. The plotted metric is the unweighted mean of the six setting-specific success rates.

Table 2: Out-of-the-box and post-trained real-robot performance, measured by progress score (%). The upper rows report pretrained models evaluated without task-specific post-training; rows marked with ✓ report task-specific post-trained models. Bold scores mark the best results within each training setting, including ties. H-0.5-Cont. and H-0.5-Disc. denote HoloBrain-0.5-Continuous and HoloBrain-0.5-Discrete, respectively, and D-Init. denotes initialization from HoloBrain-0.5-Discrete.

Post-train	Model	Place shoes	Fold towel	Stack three blocks	Place bottles in dustbin
	H-0.5-Cont.	91.00	63.20	64.44	**91.67**
	H-0.5-Disc.	**96.50**	**81.60**	**82.22**	84.76
	H-0.5-Cont. (D-Init.)	91.60	76.30	72.22	86.90
✓	π_0	75.48	61.58	70.00	55.24
✓	$\pi_{0.5}$	84.19	61.58	80.00	94.29
✓	HoloBrain-0	93.23	84.74	83.33	95.24
✓	H-0.5-Cont.	**98.40**	**97.40**	**100.00**	**100.00**
✓	H-0.5-Disc.	95.50	71.10	57.78	97.62
✓	H-0.5-Cont. (D-Init.)	**98.40**	89.50	**100.00**	**100.00**

Post-train	Model	Place empty cup	Place in slot	Stack three bowls	Average
	H-0.5-Cont.	73.50	77.33	96.44	79.65
	H-0.5-Disc.	**91.00**	80.89	**97.33**	**87.76**
	H-0.5-Cont. (D-Init.)	77.00	**85.78**	95.56	83.62
✓	π_0	48.50	68.00	97.78	68.08
✓	$\pi_{0.5}$	**99.50**	84.22	**100.00**	86.25
✓	HoloBrain-0	78.50	62.22	**100.00**	85.32
✓	H-0.5-Cont.	90.50	78.44	96.89	**94.52**
✓	H-0.5-Disc.	37.50	30.44	98.22	69.74
✓	H-0.5-Cont. (D-Init.)	60.50	**85.78**	99.56	90.53

4.2 Post-Training Performance

We evaluate the post-training performance of HoloBrain-0.5 under two settings that differ in data scale and task diversity.

In the first setting, we post-train the models on seven real-robot tasks using a single embodiment, with 200 demonstration trajectories provided for each task. This controlled setting evaluates adaptation under a limited data budget. As shown in Table 2, HoloBrain-0.5-Continuous achieves an average progress score of 94.52% after post-training, outperforming $\pi_{0.5}$ and HoloBrain-0 by 8.27 and 9.20 percentage points, respectively.

Under the same few-shot fine-tuning setting, the average progress score of HoloBrain-0.5-Discrete decreases from 87.76% to 69.74%. The current discrete model is therefore not well suited to few-shot post-training. We will investigate this performance degradation in future work.

In the second setting and in the transfer and adaptation experiments below, we use the continuous policy initialized from the discrete model (D-Init.). We perform large-scale multi-task post-training using approximately 300 hours of data collected with two dual-arm robot embodiments, Piper and Piper-X. The complete dataset covers 38 tasks, from which we select 11 representative tasks for evaluation. Depending on task complexity, the amount of training data assigned to each task ranges from 2 to 40 hours, with more challenging tasks generally receiving larger data budgets. This setting evaluates the ability of HoloBrain-0.5 to leverage large-scale heterogeneous data and adapt to more complex real-world manipulation tasks. The results are presented in Table 3.

Table 3: Large-scale multi-task post-training performance on real-robot tasks, measured by progress score (%). H-0.5-Cont. (D-Init.) denotes HoloBrain-0.5-Continuous initialized from HoloBrain-0.5-Discrete. In Tables 3–7, the green number is the lead over the second-best average, in percentage points.

Model	Grasp anything	Fold paper box	Object to cabinet	Object to drawer	Objects to bag	Sweep trash
π_0	88.00	65.50	73.00	70.50	84.40	57.50
$\pi_{0.5}$	**99.00**	68.00	74.80	68.30	**91.60**	42.50
H-0.5-Cont. (D-Init.)	97.50	**81.00**	**84.50**	**96.00**	86.70	**72.00**

Model	Close laptop	Pen to holder	Hard slot	Phone to stand	Drawstring bag	Average
π_0	46.70	16.70	56.67	**73.10**	56.50	62.60
$\pi_{0.5}$	46.70	66.70	78.00	64.40	**63.50**	69.41
H-0.5-Cont. (D-Init.)	**66.70**	**88.00**	**82.00**	66.90	48.50	**79.07** +9.66

4.3 Cross-Embodiment Generalization

We evaluate zero-shot task transfer from Piper to Piper-X on five real-robot tasks. The model has encountered Piper-X on other tasks, but the five evaluated task–Piper-X combinations are absent from both pretraining and post-training data. Training demonstrations for these five tasks come only from Piper. This setting tests generalization to unseen task–embodiment combinations on a previously seen embodiment.

Piper and Piper-X use identical sensor configurations and have the same image and joint-state dimensionalities, so input dimensions alone do not identify the embodiment. Models without an explicit embodiment condition, such as the standard π-series configurations used here, do not receive a direct signal specifying the target robot's kinematic structure. HoloBrain-0.5 conditions action generation on standardized URDF and joint-level kinematic information, providing an explicit route for adapting the learned skill to the target embodiment.

The results are summarized in Table 4. We report the progress score for each task and the average across the five tasks. This evaluation measures whether skills learned from Piper demonstrations transfer to Piper-X without demonstrations of the evaluated task–Piper-X combinations, despite differences in kinematic structure.

Table 4: Zero-shot task transfer from Piper to Piper-X, measured by progress score (%). Piper-X is seen on other tasks, but the evaluated task–Piper-X combinations are absent from both pretraining and post-training data. All models are post-trained using Piper demonstrations only and evaluated directly on Piper-X.

Model	Object in bag	Object to drawer	Close laptop	Objects to basket	Phone to stand	Average
π_0	9.40	72.00	26.70	2.00	28.10	27.64
$\pi_{0.5}$	1.90	47.00	13.30	2.50	22.50	17.44
H-0.5-Cont. (D-Init.)	**38.10**	**91.00**	**53.30**	**91.50**	**42.50**	**63.28** +35.64

4.4 Adaptation to Different Robot Morphologies

The Piper-to-Piper-X experiment in Section 4.3 asks whether a policy can transfer a task without seeing it demonstrated on the target robot. Here we ask how well the policy adapts when a small amount of target-embodiment data is available. We consider three distinct configurations: a single-arm Franka Panda, a mobile base with two manipulator arms, and a Unitree G1 humanoid. For each configuration, the comparison uses the same set of target-embodiment demonstrations to post-train HoloBrain-0.5-Continuous and $\pi_{0.5}$. We compare the resulting policies within each embodiment using task progress scores.

For the single-arm Franka, we post-train both models exclusively on the same subset of DROID [36] demonstrations. Neither model is fine-tuned on data collected in our evaluation environment or on demonstrations collected specifically for the five tabletop evaluation tasks. On these tasks (Table 5), HoloBrain-0.5-Continuous attains a mean progress score of 89.2%, compared with 85.0% for $\pi_{0.5}$. It scores higher on four of the five tasks. The largest difference is on placing a marker in a cup (85.0% versus 77.5%); on cup placement, $\pi_{0.5}$ scores slightly higher (95.8% versus 95.0%).

The mobile dual-arm tasks (Table 6) are pressing an elevator button, placing a basket, and hanging a garment on a rack. HoloBrain-0.5-Continuous achieves an average progress score of 92.94% on these three tasks, compared with 73.83% for $\pi_{0.5}$. The Unitree G1 tasks (Table 7) include retrieving a drink from a refrigerator and placing it on a table, placing a bag of dirty clothes on a table and returning, and handing a bag of clothes to a person. For the Unitree G1 experiments, we use HoloMotion-1 [49] as the whole-body motion tracker. HoloBrain-0.5-Continuous achieves an average progress score of 70.24% on the three G1 tasks, compared with 11.00% for $\pi_{0.5}$.

Table 5: Single-arm Franka Panda progress scores (%) after post-training on the same DROID subset. Neither model uses fine-tuning demonstrations collected in our evaluation environment or specifically for the five tasks. The average weights all five tasks equally.

Model	Place a cup	Transfer a towel	Move a bowl to the left	Put an object on a plate	Put a marker in a cup	Average
$\pi_{0.5}$	**95.8**	77.5	92.5	81.7	77.5	85.0
H-0.5-Cont. (D-Init.)	95.0	**82.5**	**97.5**	**85.8**	**85.0**	**89.2** +4.20

Table 6: Task progress scores (%) on the mobile dual-arm robot after post-training on matched target-embodiment data. The average weights all three tasks equally.

Model	Press an elevator button	Place a basket	Hang a garment on a rack	Average
$\pi_{0.5}$	80.00	62.50	79.00	73.83
H-0.5-Cont. (D-Init.)	**97.62**	**89.69**	**91.50**	**92.94** +19.11

Table 7: Task progress scores (%) on the Unitree G1 humanoid after post-training on matched target-embodiment data. The average weights all three tasks equally.

Model	Retrieve drink from fridge	Bag to table + return	Bag to person	Average
$\pi_{0.5}$	23.00	10.00	0.00	11.00
H-0.5-Cont. (D-Init.)	**83.00**	**100.00**	**27.73**	**70.24** +59.24

5 Conclusion

We introduce HoloBrain-0.5, a family of VLA models that extends HoloBrain-0 in model architecture, pretraining scale, and evaluation methodology. First, we further unify heterogeneous action spaces through URDF standardization to support learning across embodiments with different joint configurations and kinematic structures. Second, we develop a discrete-action variant that exhibits faster convergence during pretraining and achieves stronger out-of-the-box performance. Finally, we pretrain HoloBrain-0.5 on more than 10,000 hours of heterogeneous manipulation data and systematically evaluate its performance throughout pretraining. HoloBrain-0.5 represents a step toward scalable, embodiment-aware generalist manipulation policies. The results support the use of explicit kinematic canonicalization and complementary action-space formulations for learning from diverse robotic data.

6 Future Work

HoloBrain-0.5 is currently an action-supervised VLA model designed primarily for reactive visuomotor control. We plan to extend its capabilities along four directions:

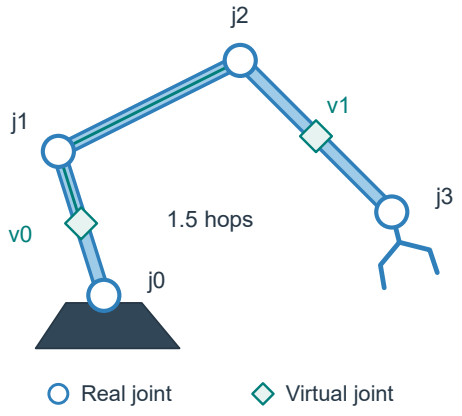
- **Broader multi-task supervision.** We will incorporate a more diverse collection of tasks into joint training, including visual question answering with textual responses and 3D grounding. Such supervision may improve semantic understanding, spatial reasoning, and the connection between high-level instructions and physical actions.
- **Scaling up training data.** We will further expand the scale of training data. HoloBrain's current use of egocentric human data is exploratory; we plan to substantially increase its proportion in the training mixture and build a more balanced data pyramid.
- **Temporal modeling and memory.** HoloBrain-0.5 currently makes decisions from visual observations at the current time step. Future versions will incorporate memory mechanisms and temporal pretraining, enabling the model to retain and use historical observations.
- **Richer physical modalities.** Vision, language, and action alone provide an incomplete representation of physical interaction. We plan to incorporate additional sensory signals, including tactile observations and joint-level force/torque feedback.

A HoloBrain-0.5-Discrete Details

This appendix supplements Section 3.3 with the virtual-joint placement procedure, the residual-quantization configuration, loss and regularization details, and implementation settings.

A.1 Virtual-Joint Placement

The placement algorithm initializes V virtual-joint positions on the kinematic graph and refines them locally. Both stages use a lexicographic objective: first minimize the maximum distance from a real joint to its nearest virtual joint, then the mean of these distances, and finally maximize the minimum pairwise distance between virtual joints. This objective favors coverage of the real joints while spreading the virtual joints along the chains. Fractional positions along physical edges extend the joint-relative relationships to real–virtual and virtual–virtual pairs, measured in physical-hop units; Figure 12 illustrates the augmented distances on a single motion chain. The real-to-real block uses the modified cross-arm relationships of Figure 6 while retaining the within-arm distances.



(a) Fractional graph positions

	Real joints				Virtual joints	
	j0	j1	j2	j3	v0	v1
j0	0	1	2	3	0.5	2.5
j1	1	0	1	2	0.5	1.5
j2	2	1	0	1	1.5	0.5
j3	3	2	1	0	2.5	0.5
v0	0.5	0.5	1.5	2.5	0	2
v1	2.5	1.5	0.5	0.5	2	0

(b) Augmented distance matrix

Figure 12: Illustrative distance augmentation for one motion chain. (a) Four real nodes and two midpoint virtual nodes, with a highlighted 1.5-hop path from v_0 to j_2. Each real-node edge has unit hop length. (b) The augmented distance matrix contains real–real, real–virtual, and virtual–virtual entries; the highlighted (v_0, j_2) entry matches the 1.5-hop path in (a). This single-chain example illustrates fractional offsets. Cross-arm relationships in dual-arm embodiments follow Figure 6.

A.2 Residual Quantization Configuration

The quantizer follows the open-source residual FSQ implementation in `vector-quantize-pytorch`[1], which chains finite scalar quantization [11] stages residually. For each latent $\mathbf{h} = \mathbf{H}_{vt}$, a shared affine projection produces four quantization coordinates $u \in \mathbb{R}^4$, soft-bounded as $r_0 = b \odot \tanh(u \oslash b)$ with $b = 1 + (L-1)^{-1}$ and $L = (8, 5, 5, 5)$, all elementwise. Stage q divides the residual by $s_q = L^{-q}$, quantizes each coordinate by scalar FSQ with straight-through rounding, rescales the quantized contribution $\widetilde{c}_q = s_q \odot c_q$, and passes on the residual $r_{q+1} = r_q - \mathrm{sg}(\widetilde{c}_q)$, with the subtracted contribution detached. The four level indices of a stage are packed with the mixed-radix basis $(1, 8, 40, 200)$ into a single index $z_{vtq} \in \{0, \ldots, 999\}$, so $K = 1000$. An r-stage prefix is decoded by summing its contributions and applying the shared output projection once, $\overline{\mathbf{h}}^{(r)} = W_{\text{out}} \sum_{q<r} \widetilde{c}_q + b_{\text{out}}$.

[1] `https://github.com/lucidrains/vector-quantize-pytorch`

A.3 Losses and Regularization

Let $\ell_w(\mathbf{B}, \mathbf{A})$ sum weighted smooth-L_1 errors on node scalars, positions, and rotation-matrix entries, using $\beta = 10^{-4}$ and dataset-dependent channel weights. Errors are summed over node and channel entries and averaged over valid frames; a temporal patch is valid only if all its frames are valid. The reconstructed absolute trajectory and the target are compared in the same space: normalized scalars and link positions in the robot's metric frame, with quaternion outputs converted to 3×3 rotation matrices for the rotation term. When kinematics are supplied, define $F(\mathbf{B})$ to preserve $\mathbf{B}$'s scalar channels and replace its pose channels with forward kinematics of the predicted joint scalars, expressed in the same loss space. The reconstruction objective in Eq. (7) is

$$\mathcal{L}_{\text{rec}}(\mathbf{B}, \mathbf{A}) = \ell_w(\mathbf{B}, \mathbf{A}) + \mathbf{1}_{\text{FK}} \left[\ell_w(F(\mathbf{B}), \mathbf{A}) + \ell_w\big(\mathbf{B}, \text{sg}(F(\mathbf{B}))\big) \right], \tag{12}$$

where the forward-kinematics term differentiates through the kinematic model to the predicted scalars, and the consistency term pulls predicted poses toward detached forward-kinematics poses. When kinematics are absent, only the direct term is used.

The policy is trained on ground-truth codes with corrupted inputs, label smoothing, and residual-stage masks:

$$\mathcal{L}_{\text{policy}} = \mathbb{E}_{\widetilde{Z}|Z} \left[\sum_{q=0}^{Q-1} \alpha_q \frac{\sum_{v,t} M_{v,t,q} \, \text{CE}_\epsilon\Big(z_{v,t,q}, \, p_{\theta,v,t,q}(\widetilde{Z}; \mathbf{c})\Big)}{\max\Big(1, \sum_{v,t} M_{v,t,q}\Big)} \right] + \mathcal{L}_{\text{depth}}, \tag{13}$$

where M combines valid-patch and cascading corruption masks, $\epsilon = 0.1$, and $\alpha = (1, 0.8, 0.6)$. Input codes are independently replaced by uniform indices with stage probabilities $(0.2, 0.5, 0.7)$ while targets remain unchanged; for a given (v, t), corruption at stage q removes loss terms at later residual stages through

$$M_{v,t,q} = m_{v,t} \prod_{k=0}^{q-1} (1 - U_{v,t,k}), \tag{14}$$

where $m_{v,t}$ marks a valid temporal patch and $U_{v,t,k}$ marks an input-code replacement. The policy additionally drops state conditioning with probability 0.05 during training and inherits auxiliary visual depth supervision from HoloBrain-0.

A.4 Implementation Settings

Figure 7(b) in Section 3.3 summarizes the tokenization and reconstruction pipeline, whose encoder and decoder use pre-normalization, residual connections, and a final RMSNorm [50]. Table 8 lists the default settings.

Table 8: Default tokenizer settings. One action chunk contains the full future trajectory; a temporal patch is the shorter input unit.

Setting	Value
Action chunk length H / current-state length	64 / 1 frames
Frames per temporal patch / patch count T	4 / 16
Hidden width C / attention heads	384 / 8
Tokenizer encoder / decoder blocks	4 / 4
Virtual-joint allocation ratio ρ	6
Residual stages Q / indices per stage K	3 / 1000
FSQ scalar levels	$(8, 5, 5, 5)$
Residual-prefix loss weights λ	$(0.25, 0.5, 1)$

Distance encoding uses 256 sinusoidal features with maximum period 10,000 and input scale one; temporal rotary encoding uses base 10,000. The policy uses the same Qwen3.5 backbone as the continuous variant.

Its robot-state encoder is initialized from the tokenizer and then trained independently while the tokenizer remains frozen. Each of the Q classification heads is shared across virtual joints and temporal patches. For $V = 4$, the default settings produce 192 code targets and 384 code/cue positions, generated in 21 rounds.

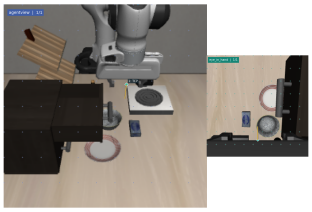
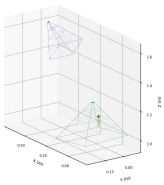
B Real-Robot Evaluation Protocol

All real-robot results are reported as progress scores normalized to $[0, 100]\%$. Each task is decomposed into observable milestones, and the score is the normalized sum of the milestone weights. The protocol for the reported real-robot tasks is summarized in Table 9. Time limits and retry limits are included where they affect scoring.

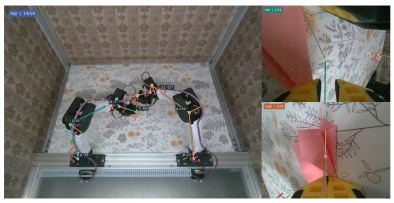
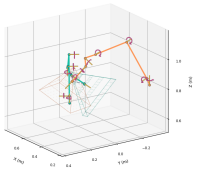
Table 9: Compact real-robot progress-scoring protocol. Scores are normalized to 100%; the listed milestones are the primary components of each task score.

Task	Scoring milestones	Constraints
Place shoes	Grasp shoe (.4); partial placement (.2); complete placement (.2); correct orientation (.2).	–
Fold towel	Corner grasp (.1); first fold (.3); opposite-side grasp (.1); second fold (.2); final quality (.2).	–
Stack three blocks	Per block: grasp (.25) and successful placement (.25).	–
Place bottles in dustbin	Per bottle: grasp (.3) and successful placement (.2).	–
Place empty cup	Grasp (.5); partial placement (.2); complete placement (.3).	–
Place in slot	Per block: grasp (.25); within 2 cm (.15); complete placement (.1).	At most 3 attempts per block.
Stack three bowls	Per bowl: grasp (.25); partial stack (.15); complete stack (.1).	At most 3 attempts per bowl.
Grasp anything	Per object: grasp and place (.1), or grasp only (.05).	At most 3 attempts per object.
Fold paper box	Left grasp (.2); right grasp (.1); first fold (.3); second fold (.3); placement (.1).	–
Object to cabinet	Unlock (.2); open (.15); each object (.1); close (.15); relock (.2).	3 min.
Object to drawer	Reach/grasp handle (.15/.2); open (.1); each object (.1); close (.25).	Time limit.
Objects to bag	Unzip (.3); open bag (.05); each object (.15); close zipper (.35, or .2 if partial).	–
Sweep trash	Grasp broom (.3) and dustpan (.2); sweep (.2); empty (.2); return tools (.1).	3 min.
Close laptop	Fully close the laptop lid (1.0).	Binary success.
Pen to holder	Start from 1.0; subtract .1 for each failed attempt.	3 min.
Hard slot	Per item: grasp (.1); place (.4), with partial credit at 80% overlap.	25 s per item.
Phone to stand	Position stand (.2); grasp phone (.4); place phone (.4, or .2 if imperfect).	–
Drawstring bag	Open (.3); each object (.1); grasp both cords (.2); pull closed (.3).	–
Retrieve drink from fridge	Open refrigerator (.3); grasp drink (.3); close refrigerator (.2); place drink (.1); stable placement (.1).	Each subtask limited to 30 s.
Bag to table + return	Grasp clothes bag (.2); place on table (.6); return to starting position (.2).	Each subtask limited to 30 s.
Bag to person	Walk to table (.1); grasp clothes bag (.7); return and hand bag to person (.2).	Each subtask limited to 30 s.

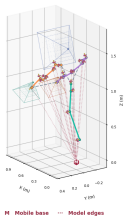
C Joint-Graph Examples

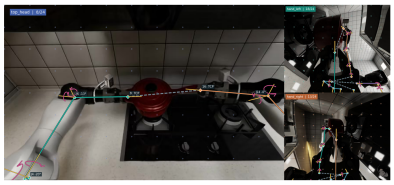
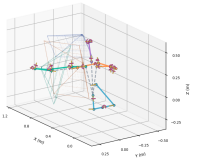
(a) LIBERO Goal (TCP only)

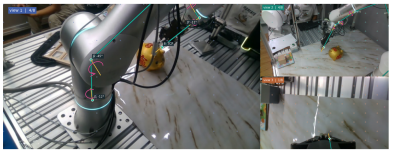
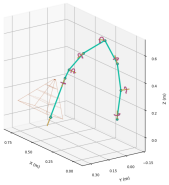
(b) YAM dual-arm robot

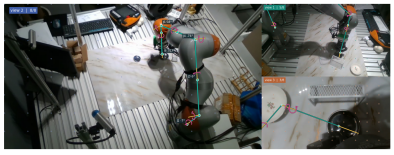
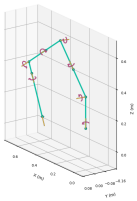
(c) BEHAVIOR R1 Pro mobile base

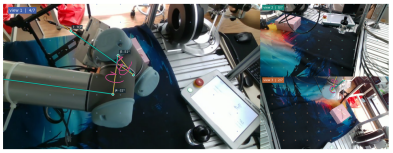
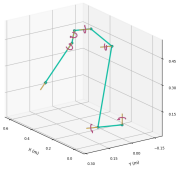
(d) AgiBot G2 OmniPicker

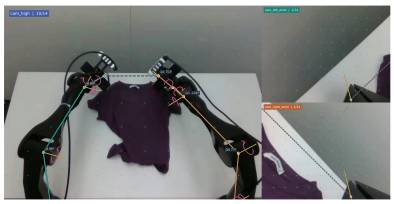
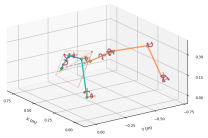
(e) RH20T: Flexiv Rizon

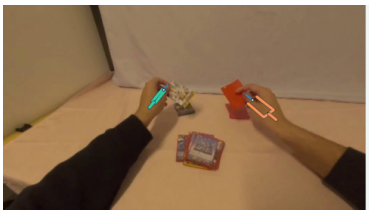
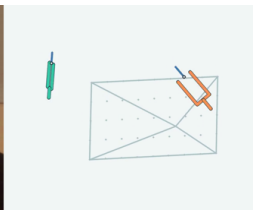
(f) RH20T: KUKA LBR iiwa

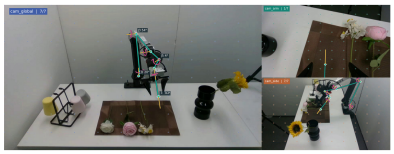
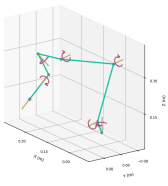
(g) RH20T: UR5

(h) Table30: DOS-W1

(i) EgoDex: hands represented as grippers

(j) Table30: ARX5

Figure 13: Additional joint graphs. In each panel, camera views (left) and the 3D graph (right) show the same configuration. The TCP-only interface is represented by one node; the mobile platform also represents motion of the base.

References

[1] Anthony Brohan, Noah Brown, Justice Carbey, Yevgen Chebotar, Krzysztof Choromanski, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[2] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Chelsea Finn, Chuyuan Fu, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

[3] Open X-Embodiment Collaboration, Abhishek Padalkar, Adam Pooley, Ajay Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.

[4] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. pi0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[5] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

[6] Physical Intelligence, Bo Ai, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Greg Balke, Kevin Black, et al. pi0.7: A steerable generalist robotic foundation model with emergent capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

[7] X Square Robot et al. Wall-oss-0.5 technical report: Pretrain once, act anywhere. *arXiv preprint arXiv:2605.30877*, 2026.

[8] Yicheng Liu, Zibin Dong, Baijun Ye, Tianyuan Yuan, Tao Jiang, Anqi Yang, Shicheng Cao, Haonan Liu, Yue Sun, Zihan Guo, Xiao Liu, Dong Ke, Changxun Pan, Chenru Wu, Tailai Cheng, Xiaoshu Ren, Xinlei Zhang, Jianning Cui, Zijie Zhao, Haoyu Zhang, Kaiming Xu, Haodong Yang, Bowen Zhang, Jiahui Niu, Shaoting Zhu, Shiduo Zhang, and Hang Zhao. G0.5: One autoregressive stream for robot reasoning and action. *arXiv preprint arXiv:2608.11739*, 2026.

[9] Wei Wu, Fan Lu, Yunnan Wang, Shuai Yang, Shi Liu, Fangjing Wang, Qian Zhu, He Sun, Yong Wang, Shuailei Ma, Yiyu Ren, Kejia Zhang, Hui Yu, Jingmei Zhao, Shuai Zhou, Zhenqi Qiu, Houlong Xiong, Ziyu Wang, Zechen Wang, Ran Cheng, Yong-Lu Li, Yongtao Huang, Xing Zhu, Yujun Shen, and Kecheng Zheng. A pragmatic vla foundation model. *arXiv preprint arXiv:2601.18692*, 2026.

[10] Xuewu Lin, Tianwei Lin, Yun Du, Hongyu Xie, Yiwei Jin, Jiawei Li, Shijie Wu, Qingze Wang, Mengdi Li, Mengao Zhao, et al. Holobrain-0 technical report. *arXiv preprint arXiv:2602.12062*, 2026.

[11] Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. Finite scalar quantization: Vq-vae made simple. *arXiv preprint arXiv:2309.15505*, 2023.

[12] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

[13] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[14] Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, et al. Rdt-1b: A diffusion foundation model for bimanual manipulation. *arXiv preprint arXiv:2410.07864*, 2024.

[15] Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, et al. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024.

[16] NVIDIA, Johan Bjorck, Fernando Castaneda, Nikita Cherniadev, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

[17] Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. *arXiv preprint arXiv:2510.10274*, 2025.

[18] Jiaming Liu, Mengzhen Liu, Zhenyu Wang, Pengju An, et al. Robomamba: Efficient vision-language-action model for robotic reasoning and manipulation. *arXiv preprint arXiv:2406.04339*, 2024.

[19] Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, et al. Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation. *arXiv preprint arXiv:2409.12514*, 2024.

[20] Chia-Yu Hung, Qi Sun, Pengfei Hong, Amir Zadeh, et al. Nora: A small open-sourced generalist vision language action model for embodied tasks. *arXiv preprint arXiv:2504.19854*, 2025.

[21] Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

[22] Yihao Wang, Pengxiang Ding, Lingxiao Li, Can Cui, et al. Vla-adapter: An effective paradigm for tiny-scale vision-language-action model. *arXiv preprint arXiv:2509.09372*, 2025.

[23] Tong Wu, Shoujie Li, Junhao Gong, Changqing Guo, et al. CEI: A unified interface for cross-embodiment visuomotor policy learning in 3D space. *arXiv preprint arXiv:2601.09163*, 2026.

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

[25] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

[26] Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, et al. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

[27] Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, et al. Spatialvla: Exploring spatial representations for visual-language-action model. *arXiv preprint arXiv:2501.15830*, 2025.

[28] Seungjae Lee, Yibin Wang, Haritheja Etukuru, H. Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior generation with latent actions. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

[29] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, et al. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*, 2025.

[30] Qwen Team. Qwen3.5-2B model card. `https://huggingface.co/Qwen/Qwen3.5-2B`, 2026.

[31] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

[32] Jade Copet, Felix Kreuk, Itai Gat, Tal Remez, David Kant, Gabriel Synnaeve, Yossi Adi, and Alexandre Défossez. Simple and controllable music generation. In *Advances in Neural Information Processing Systems*, 2023.

[33] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment—a modern synthesis. In *Vision Algorithms: Theory and Practice*, pages 298–372. Springer, 2000.

[34] Arthur Allshire, Himanshu Gaurav Singh, Ritvik Singh, Adam Rashid, et al. Scalable behavior cloning with open data, training, and evaluation. *arXiv preprint arXiv:2606.27375*, 2026.

[35] AgiBot World Team et al. Agibot world colosseo: A large-scale manipulation platform for scalable and intelligent embodied systems. *arXiv preprint arXiv:2503.06669*, 2025.

[36] Alexander Khazatsky et al. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.

[37] Hao-Shu Fang, Hongjie Fang, Zhenyu Tang, Jirong Liu, et al. Rh20t: A comprehensive robotic dataset for learning diverse skills in one-shot. *arXiv preprint arXiv:2307.00595*, 2023.

[38] Adina Yakefu, Bin Xie, Chongyang Xu, Enwen Zhang, et al. RoboChallenge: Large-scale real-robot evaluation of embodied policies. *arXiv preprint arXiv:2510.17950*, 2025.

[39] Chenghao Yin, Da Huang, Di Yang, Jichao Wang, et al. Genie Sim 3.0: A high-fidelity comprehensive simulation platform for humanoid robot. *arXiv preprint arXiv:2601.02078*, 2026.

[40] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, et al. Behavior-1k: A human-centered, embodied ai benchmark with 1,000 everyday activities and realistic simulation. *arXiv preprint arXiv:2403.09227*, 2024.

[41] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, and Yuke Zhu. Libero: Benchmarking knowledge transfer for lifelong robot learning. *arXiv preprint arXiv:2306.03310*, 2023.

[42] Soroush Nasiriany et al. Robocasa: Large-scale simulation of everyday tasks for generalist robots. *arXiv preprint arXiv:2406.02523*, 2024.

[43] Tianxing Chen, Yue Chen, Zixuan Li, Junyuan Tang, et al. RoboDojo: A unified sim-and-real benchmark for comprehensive evaluation of generalist robot manipulation policies. *arXiv preprint arXiv:2607.04434*, 2026.

[44] Yao Mu et al. Robotwin: Dual-arm robot benchmark with generative digital twins. *arXiv preprint arXiv:2409.02920*, 2024.

[45] Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, et al. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

[46] Yang Tian, Yuyin Yang, Yiman Xie, Zetao Cai, et al. InternData-A1: Pioneering high-fidelity synthetic data for pre-training generalist policy. *arXiv preprint arXiv:2511.16651*, 2025.

[47] Ryan Hoque, Peide Huang, David J. Yoon, Mouli Sivapurapu, and Jian Zhang. Egodex: Learning dexterous manipulation from large-scale egocentric video. *arXiv preprint arXiv:2505.11709*, 2025.

[48] Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, et al. LIBERO-Plus: A progressive robustness benchmark for visual-language-action models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 38574–38583, 2026.

[49] Maiyue Chen, Kaihui Wang, Bo Zhang, Xihan Ma, Zhiyuan Yang, Yi Ren, Qijun Huang, Zihao Zhu, Yucheng Wang, and Zhizhong Su. Holomotion-1 technical report. *arXiv preprint arXiv:2605.15336*, 2026.

[50] Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, 2019.