

See like a Robot: Robot-Centric Pointmaps for Vision-Language-Action Models

Byungkun Lee^{1*}, Dongyoon Hwang^{1*}, Dongjin Kim¹,
Hojoon Lee², Minho Park¹, Jaegul Choo¹
¹KAIST AI, ²Holiday Robotics

Abstract: Vision-language-action (VLA) models predict robot actions from visual observations and language instructions. These actions are defined in the robot’s own 3D coordinate frame, yet most VLAs observe the scene in the camera frame, creating a frame mismatch between where the scene is observed and where actions are defined. The mismatch is benign under a fixed viewpoint, where the policy can memorize a single observation-to-action mapping, but grows harder as large-scale datasets aggregate demonstrations across diverse camera setups and the policy must generalize this mapping across viewpoints. We address this mismatch with robot-centric pointmaps, images whose pixels store the 3D coordinates of scene points in the robot frame. Pointmaps provide robot-frame 3D geometry while preserving the dense $H \times W$ grid expected by pretrained 2D VLAs, so they integrate into existing VLAs with minimal architectural change. On RoboCasa, pointmaps improve both $\pi_{0.5}$ and SmolVLA and outperform representative camera-viewpoint and 3D-aware baselines. In real-robot experiments, their advantage over an RGB-only policy widens when the camera is moved to a placement unseen during training. The project page is available [here](#).

Keywords: VLA, manipulation, 3D geometry, pointmap

1 Introduction

Vision-language-action (VLA) models learn robotic manipulation policies from large-scale datasets, predicting actions from visual observations and language instructions. These actions are commonly defined in the robot’s own 3D coordinate frame (*e.g.*, task-space end-effector commands in the robot base frame), so reliable manipulation requires reasoning about where the target object lies in metric 3D coordinates relative to the robot. However, most VLAs receive observations in the camera frame, whether as RGB images or depth maps [1–5]. This creates a frame mismatch: the policy observes the scene in the camera frame but predicts actions in the robot frame.

This frame mismatch becomes harder to handle when training data spans diverse camera viewpoints, as large-scale datasets increasingly aggregate demonstrations across institutions with different camera setups [6–8]. Under a fixed camera viewpoint, all observations share the same viewpoint relative to the robot, so a single observation-to-action mapping remains consistent across the dataset. Under a wide range of viewpoints, this no longer holds: the policy must additionally learn how observations from different viewpoints map to the robot-frame actions they should produce. The frame mismatch is present in both cases, but only under viewpoint variation must the policy generalize the observation-to-action mapping across viewpoints rather than memorize a single one.

This motivates revising the policy’s input observation: rather than making the policy infer robot-frame actions from camera-frame observations, we provide observations that already carry (1) 3D spatial information for manipulation and (2) that information in the robot’s own coordinate frame. As shown in Fig. 1, depth maps provide 3D cues, but remain tied to the camera frame rather than

*Equal contribution. Correspondence to byungkun.lee@kaist.ac.kr

Diverse camera viewpoints in training data

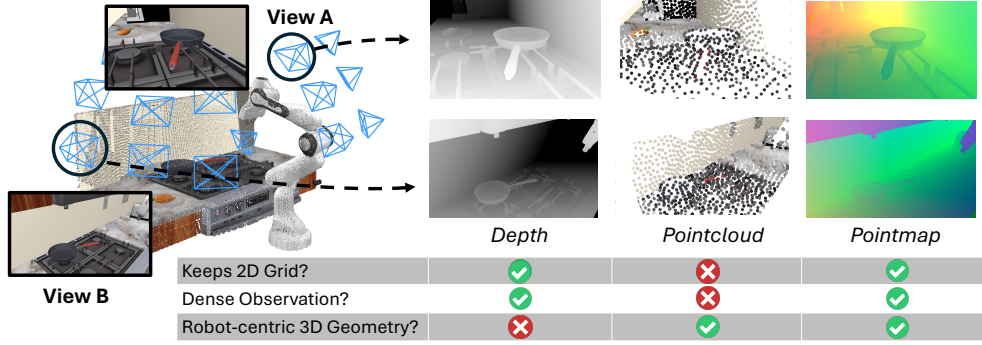


Figure 1: **Robot-centric pointmaps provide 3D geometry aligned with robot-frame actions.** Large-scale training data is collected from diverse camera viewpoints around the robot. Robot-centric pointmaps preserve the dense $H \times W$ grid expected by pretrained 2D VLAs while providing robot-centric 3D geometry.

the robot frame. A point cloud can provide robot-centric 3D geometry, but it discards the regular $H \times W$ image grid used by pretrained 2D VLAs, making seamless integration difficult. It also reduces the dense observation to a sparse set of 3D points, which may lose fine-grained details useful for manipulation. These trade-offs suggest the need for an input that provides robot-frame 3D geometry while preserving the image-form structure of RGB observations.

In this paper, we implement this idea with robot-centric pointmaps as inputs to pretrained VLAs. A robot-centric pointmap is an image whose pixels store the 3D coordinates of the corresponding scene points in the robot frame. Fig. 2 shows how camera-frame observations are expressed in the robot frame. We first lift each RGB-D observation into 3D within its camera frame, then transform it into the shared robot frame in which actions are defined, while preserving the RGB image’s $H \times W$ layout. This lets the pointmap integrate into existing 2D VLAs with minimal architectural change. We encode the pointmap with a separate visual tower initialized from the RGB encoder, then add the resulting pointmap tokens element-wise to the corresponding RGB tokens. It requires no point cloud-specific encoder or voxelization module, and because pointmap tokens are added onto the RGB tokens rather than concatenated, it introduces no extra tokens.

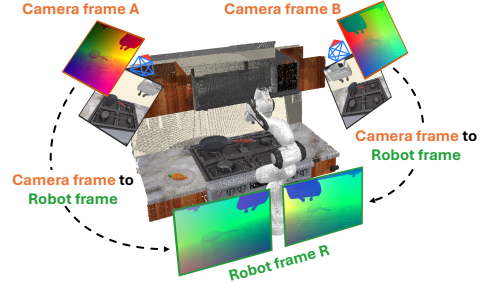


Figure 2: Observations from different camera frames are expressed in the coordinate frame where robot actions are defined.

The core contribution of this work is a simple mechanism that injects robot-frame 3D geometry into pretrained VLAs without a point cloud-specific encoder, along with a controlled analysis of the design choices behind the pointmap input and their effect on performance. On RoboCasa [9], whose training demonstrations include randomized camera viewpoints, our method improves $\pi_{0.5}$ and SmolVLA by +7.6 and +4.2 points, respectively, and outperforms representative camera-viewpoint and 3D-aware baselines. In real-robot experiments with a Franka robot, with demonstrations collected across multiple camera viewpoints, the gain over the RGB baseline grows from +5.0 points at a training camera placement to +11.7 points at a held-out camera placement unseen during training.

2 Related Work

2.1 Handling Camera Viewpoint Variation in VLAs

Camera viewpoint variation degrades VLA performance [10, 11], and it arises in two distinct regimes. At training time, large-scale datasets aggregate demonstrations across diverse camera se-

tups [6–8, 12], so the policy must fit many viewpoints at once. At evaluation time, the deployment camera may differ from those seen in training [13, 14], so the policy must generalize to an unseen viewpoint. Our work primarily targets the training-time regime, and we additionally test generalization to unseen evaluation cameras.

Existing methods can be grouped by how they supply viewpoint information. The first line leaves the policy’s input observation as a camera-view image, supplying viewpoint information around it, either as explicit camera or 3D positional cues [15–18] or as a camera-frame reparameterization of the action labels [19]. In both, the policy’s visual input stays a camera-view image, so the scene is never expressed in the robot frame that the actions live in. The second line synthesizes views, either expanding training-time coverage [20, 21] or canonicalizing test-time views toward the training distribution [22, 23], but both rely on a generative model whose synthesis grows unreliable for viewpoints far from the training views, capping the robustness they offer. The third line, closest to ours, acts on the observation itself by building shared 3D scene representations [24, 25], yet these representations no longer retain the image structure that pretrained VLAs expect. We instead express the observation as a robot-centric pointmap, which keeps that structure while assigning the same robot-frame coordinate to a physical point across viewpoints. This gives viewpoint-diverse demonstrations a consistent geometry, and the same invariance transfers to unseen evaluation cameras.

2.2 Injecting 3D Geometry into VLAs

Manipulation is inherently a 3D task, yet VLAs typically perceive the scene only through 2D RGB images. A growing body of work therefore augments VLAs with explicit 3D geometry. These methods can be grouped by how they inject it, and each either adds a dedicated module or stage or forgoes metric 3D at deployment. The first line attaches a dedicated 3D encoder or geometric expert [26–31], introducing new modules that cannot inherit the VLA’s pretrained visual weights. The second line re-renders the geometry as 2D views [32, 33] or predicts future 3D scenes with a generative world model [34], adding a rendering or generation stage to every inference step. The third line leverages a geometric foundation model, injecting its features as inputs [35] or aligning VLA features to them during training [36], so at deployment the policy receives no metric 3D input. We instead provide explicit geometry as a pointmap, an image whose pixels store 3D coordinates. Because pointmaps retain the same $H \times W$ structure as RGB, the pointmap encoder reuses the pretrained weights of the VLA’s RGB encoder, adding no point cloud-specific encoder and no per-inference stage while still providing metric 3D input at deployment. Pointmaps are already a standard representation in modern 3D vision [37–44], and recent work applies them to from-scratch diffusion policies [45, 46]. Yet their use within pretrained VLAs, and the frame in which to express them, remain underexplored.

3 Method

We reduce the observation-to-action frame mismatch by lifting each RGB-D observation into a robot-centric pointmap before policy learning. Alongside the RGB input, the VLA receives a robot-centric pointmap, encoded by a separate pointmap encoder and fused with the RGB visual tokens. We detail the pointmap construction and the fusion mechanism below.

Robot-centric pointmap. Fig. 3 illustrates how an RGB-D observation is lifted into a camera-frame pointmap and then expressed in the robot frame. Consider an RGB image of resolution $H \times W$ captured by camera c , with intrinsics K_c , rotation R_c and translation t_c from the camera frame to the robot base frame, and per-pixel depth D_c . We first lift each pixel into a 3D point in the camera frame,

$$P_c^{\text{cam}}(u, v) = D_c(u, v) K_c^{-1} [u, v, 1]^\top. \quad (1)$$

We then apply the camera-to-robot transform to express the point in the robot base frame,

$$P_c^{\text{R}}(u, v) = R_c P_c^{\text{cam}}(u, v) + t_c. \quad (2)$$

The robot-frame pointmap P_c^R has the same $H \times W$ grid as the RGB image, with one 3D coordinate per pixel. The same physical scene point receives the same robot-frame coordinate regardless of the camera viewpoint, although the pixel (u, v) where it appears may change.

We additionally re-center the pointmap on the current end effector,

$$P_c^{EE}(u, v) = P_c^R(u, v) - t_{EE}, \quad (3)$$

where t_{EE} is the end-effector position in the robot base frame. We refer to this as an end-effector-centered pointmap. This centering expresses scene geometry relative to the current end-effector position. Because actions are defined as motions of that end effector, the observation and the action space now share a common origin. We ablate this choice in Section 4.3.

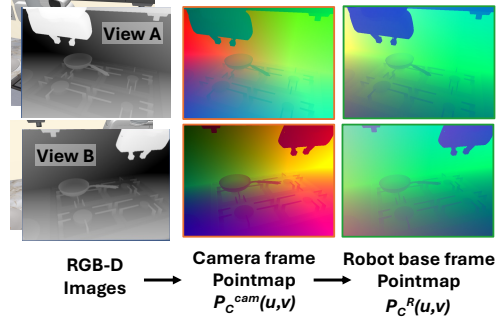


Figure 3: RGB-D observations are lifted into camera-frame pointmaps and transformed into robot-frame pointmaps.

Pointmap-RGB fusion. Because the pointmap preserves the same $H \times W$ grid as the RGB image, we process it with a visual encoder of the same architecture as the one the VLA uses for RGB. We use a separate pointmap encoder g_ϕ , initialized from the RGB encoder f_θ , to map P_c^{EE} into tokens with the same shape as the RGB tokens. We add these tokens element-wise to the corresponding RGB tokens,

$$z_c = f_\theta(I_c) + g_\phi(P_c^{EE}) \in \mathbb{R}^{N_{\text{tok}} \times d}, \quad (4)$$

where f_θ is the RGB encoder and I_c is the RGB image of camera c . The fused tokens z_c then serve as the visual input to the VLA in place of the original RGB tokens, incorporating robot-centric 3D geometry into the VLA’s existing visual stream without introducing a point cloud encoder, voxel module, or additional token sequence.

4 Design Choices for Robot-Centric 3D Observations in VLAs

The previous sections motivate robot-centric pointmaps as a way to reduce the observation-to-action frame mismatch under camera viewpoint variation. In this section, we isolate the design choices of Section 3 through controlled studies. We hold the backbone and training recipe fixed while changing only how 3D geometry is provided to the policy. We answer four questions:

- RQ1.** Does giving the policy camera information suffice under viewpoint variation, or must the camera-to-robot transform be applied to the observation itself? (Section 4.1)
- RQ2.** Once 3D geometry is expressed in a robot-centric frame, should the 3D geometry be represented as an image-form pointmap or as a point cloud? (Section 4.2)
- RQ3.** Given an image-form pointmap, which coordinate origin best supports robustness to viewpoint variation? (Section 4.3)
- RQ4.** Does our method remain robust to increasing training-time camera viewpoint variation? (Section 4.4)

We run these controlled studies on RoboCasa [9], where each policy is trained on 50 human demonstrations per task across 24 atomic tasks. Because RoboCasa randomizes camera viewpoints across demonstrations by default, it directly exercises the viewpoint variation we study. For the VLA backbone, we use a π -style architecture [3] initialized from a base PaliGemma checkpoint [47] rather than the robot-pretrained π checkpoint, removing large-scale robot pretraining as a confounding factor. We use RoboCasa’s default randomization for the design ablations in Sections 4.1–4.3. For

Table 1: **Directly providing robot-centric 3D geometry vs. learning it.** D/K/E indicate whether depth, intrinsics, and extrinsics are provided to the model. The last two rows use the same depth and calibration information, but differ in whether robot-frame geometry is pre-computed.

Input	D	K	E	Transform	SR
RGB	–	–	–	None	27.9
RGB + Plücker	–	✓	✓	Learned	28.7
RGB + Plücker + Depth	✓	✓	✓	Learned	31.6
RGB + Pointmap	✓	✓	✓	Pre-computed	34.7

Section 4.4, we additionally vary the amount of camera randomization across demonstrations to test whether our method remains robust as training-time viewpoint variation increases. Each variant is trained for 30k steps and evaluated over 50 episodes per task. For all tables in this section, SR denotes success rate.

4.1 Robot-frame pointmaps outperform conditioning on camera information

Under viewpoint variation, camera-frame observations force the policy to learn a viewpoint-dependent mapping to robot-frame actions, whereas a robot-frame pointmap assigns each scene point the same coordinate regardless of viewpoint. We therefore ask whether the policy should learn this mapping from raw camera information, or whether pre-computing the camera-to-robot transform, so the observation already carries robot-frame geometry, is more effective.

Table 1 compares four inputs, that differ in how much camera information they carry and whether it is already converted into robot-frame geometry. While RGB provides neither depth nor camera calibration, RGB + Plücker provides camera-ray information through per-pixel Plücker rays. A Plücker ray is a six-dimensional encoding of the 3D ray passing through each image pixel, determined by the camera intrinsics and extrinsics. Thus, Plücker rays tell the policy where each pixel lies relative to the camera and how the camera is posed relative to the robot. RGB + Plücker + Depth additionally provides per-pixel depth, so it contains the same depth, intrinsics, and extrinsics used to construct a pointmap. The policy, however, must still convert this depth and calibration into robot-frame coordinates on its own. RGB + Pointmap instead pre-computes the robot-frame pointmap using Eq. 2, so each observed scene point is assigned its robot-frame coordinate before entering the policy.

Table 1 shows that providing depth and camera calibration to the policy is useful, but constructing robot-centric 3D geometry before policy learning is more effective. RGB + Plücker modestly improves over RGB (28.7 vs. 27.9), and adding depth raises success to 31.6. RGB + Pointmap reaches 34.7, even though it is constructed from the same depth, intrinsics, and extrinsics available to RGB + Plücker + Depth. Because these two variants share the same depth and calibration and differ only in whether robot-frame geometry is pre-computed, the 3.1-point gap isolates the benefit of pre-computation. These results suggest that providing depth and calibration as cues is helpful, but directly providing robot-frame geometry is more effective than leaving it for the policy to infer. We therefore adopt the pre-computed robot-frame pointmap as the input for the remaining experiments.

4.2 Image-form pointmaps outperform point clouds

Given the same robot-frame 3D points, we next ask how to provide them to the VLA: as an image-form pointmap that preserves the RGB pixel grid ($H \times W$), or as a point cloud that does not. A point cloud carries the same 3D robot-frame points, but discards the image-form pixel grid at two costs. First, it no longer aligns with the image-based visual pathway of a pretrained VLA, so it requires a dedicated point cloud encoder and a separate fusion mechanism, adding a 3D feature stream outside the visual tokens used during VLA pretraining. Second, point cloud encoders typically subsample the scene to a fixed number of points, yielding a sparser input that may omit fine-grained details. A

Table 2: **Image form vs. point cloud.** All variants add the same robot-frame 3D points to an RGB baseline, but differ in how the points are encoded and fused. SR denotes success rate.

Input	3D encoder	Fusion	SR
RGB	–	–	27.9
RGB + Point cloud	MLP	concat	24.2
RGB + Point cloud	PTv3 [48]	concat	32.8
RGB + Pointmap	-	concat	30.7
RGB + Pointmap	-	add	34.7

Table 3: **End-effector centering improves viewpoint robustness.** Fixed and Rand. indicate whether the evaluation camera viewpoint is fixed or randomized, and Δ denotes the change from Fixed to Rand.

Input	Fixed	Rand.	Δ
RGB	27.9	25.8	-2.1
RGB + Pointmap (base)	34.7	32.7	-2.0
RGB + Pointmap (EE)	36.9	36.6	-0.3

pointmap instead preserves one XYZ value per RGB pixel, so it reuses the VLA’s image-encoder architecture and fuses directly with the corresponding RGB tokens.

Table 2 shows that preserving the RGB pixel grid is important because it enables direct element-wise fusion between pointmap and RGB tokens. As shown in Fig. 4, addition fuses spatially corresponding tokens on the shared image grid, while concatenation appends the pointmap tokens as a separate sequence. This raises success from 30.7 to 34.7. RGB + pointmap with element-wise addition also outperforms both the lightweight point cloud baseline with an MLP encoder (24.2) and the pre-trained point cloud baseline with Point Transformer v3 [48] (32.8). Because a point cloud does not preserve the grid, its features have no spatial correspondence with the RGB tokens and cannot use element-wise fusion. Overall, preserving the image grid is central to why pointmaps integrate well with pretrained VLAs.

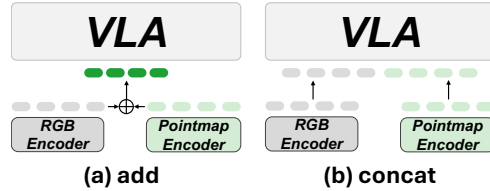


Figure 4: **Pointmap-RGB fusion.** Addition preserves spatial correspondence, while concatenation treats pointmap tokens as a separate sequence.

4.3 End-effector centering yields a viewpoint-robust representation

The last remaining design choice is the coordinate origin of the pointmap. The origin governs how much the input depends on where a target sits in the workspace, a factor prior work shows strongly affects 3D manipulation policies [49–52]. We examine this for pointmap-based VLAs by comparing two origins: the robot base and the current end-effector position. We refer to these as the robot-base-centered pointmap and the end-effector-centered pointmap, respectively (Eq. 3).

A robot-base origin encodes each point by its absolute location in the workspace, which obscures what the action actually depends on: the position of the target relative to the end effector. Consider two tasks that require the same motion, bringing the gripper to a target to grasp it. Grasping near the sink and grasping near the microwave demand a nearly identical gripper-to-target approach, yet the target sits at unrelated robot-base coordinates, $((0.4, 0.5, 0.3))$ and $((0.6, 0.1, 0.8))$ (Fig. 5b). The robot-base frame thus casts two equivalent interactions as unrelated geometry. End-effector centering removes this mismatch: it expresses every point relative to the current end-effector position, so the target’s coordinate is exactly the displacement the gripper must cover. At grasp, that displacement is small in both tasks, and the two situations collapse near $((0, 0, 0))$. Targets that are scattered across the robot-base frame therefore cluster tightly around the end-effector origin (Fig. 5a), giving the policy a consistent local geometry for interactions that share the same motion.

End-effector centering raises success from 34.7 to 36.9 under the fixed evaluation viewpoint (Table 3). The advantage persists when the evaluation viewpoint is randomized. The robot-base-centered pointmap drops by 2.0 points $(34.7 \rightarrow 32.7)$, whereas the end-effector-centered pointmap drops by only 0.3 points $(36.9 \rightarrow 36.6)$. This smaller drop is consistent with end-effector center-

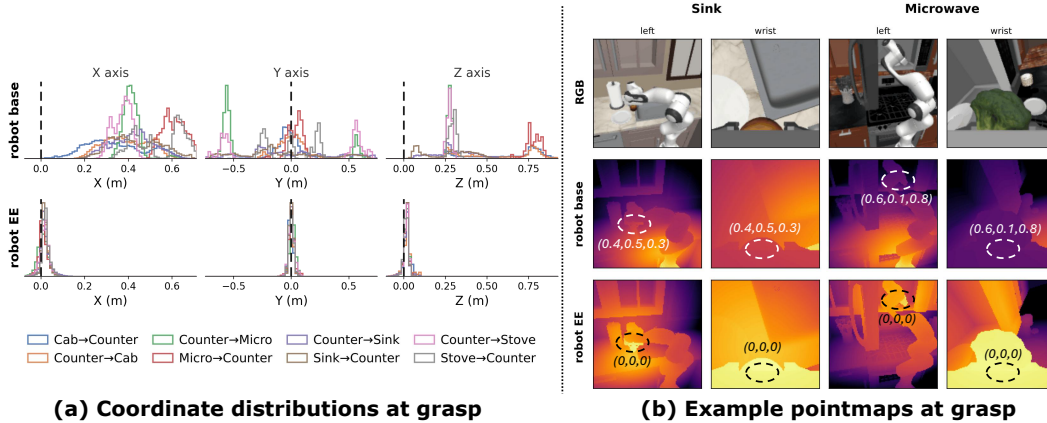


Figure 5: **End-effector centering concentrates interaction targets near a common origin.** (a) Target coordinates are broadly distributed in the robot-base frame but cluster near the origin in the end-effector-centered frame. (b) Two example tasks show that targets with different robot-base coordinates lie near the end-effector-centered origin at grasp.

ing making the input less dependent on the target’s absolute workspace location. We therefore use end-effector-centered pointmaps in the remaining experiments.

4.4 Robustness to increasing training-time viewpoint variation

The previous sections establish our pointmap design. We now test whether this design remains robust as training-time camera viewpoint variation increases. To isolate this factor, we vary only the level of camera randomization in RoboCasa while holding all other factors fixed. The *No variation* setting uses a fixed training camera viewpoint across demonstrations, *Low* applies RoboCasa’s standard camera randomization, and *High* widens the randomization range. Additional details and visualizations are provided in Appendix A.

Fig. 6 shows that increasing training-time viewpoint variation makes RGB-only learning harder. RGB falls 9.6 points (34.5% → 24.9%) from *no* to *high* variation, whereas RGB + pointmap keeps the policy robust to training-time viewpoint variation.

Together, these controlled studies determine the design we use in the rest of the paper. We precompute robot-frame geometry, keep it in image form as a pointmap, and re-center it at the current end-effector position. In the next section, we evaluate this design with pretrained VLA backbones, compare it against prior camera-viewpoint and 3D-aware methods, and test it on a real robot under seen and unseen camera placements.

5 Experiments

We now ask whether this design remains effective when fine-tuning VLAs already pretrained on large-scale robot data, how it compares with prior 3D-aware and camera-viewpoint methods, and whether its benefits hold under camera variation in simulation benchmarks (Section 5.1) and real-world tasks (Section 5.2).

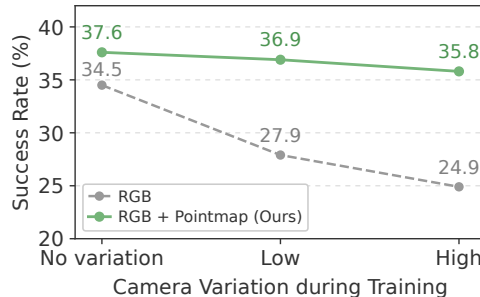


Figure 6: **Effect of training-time camera viewpoint variation.** RGB performance drops as viewpoint variation increases, while RGB + Pointmap remains stable.

Table 4: RoboCasa results (success rate %, fixed viewpoint) over 24 tasks in five categories. All methods except FP3 (standalone point cloud) use the $\pi_{0.5}$ backbone. Shaded rows add our pointmap.

Method	Avg.	Doors	Drawers	Coffee	Pick-and-place	Turn objects
<i>point cloud policy</i>						
FP3 [54]	42.8	79.0	74.0	54.7	21.8	32.0
<i>Camera-aware VLAs</i>						
OC-VLA [19]	56.3	80.0	80.0	42.0	50.5	48.9
KYC [16]	59.1	86.5	86.0	51.3	48.2	51.4
<i>3D-augmented VLAs</i>						
GeoVLA [26]	57.1	81.0	80.0	45.3	49.8	50.3
PointVLA [27]	57.3	87.5	85.0	44.0	46.3	50.3
<i>Ours</i>						
$\pi_{0.5}$ [4]	55.3	79.5	83.0	40.7	46.0	50.6
+ pointmap	62.9	90.0	90.0	58.0	52.8	53.4
SmolVLA [53]	37.2	68.0	63.0	39.3	6.5	46.6
+ pointmap	41.4	80.0	77.0	38.0	12.8	43.4

5.1 Simulation Experiments

Experimental Setup We evaluate on RoboCasa across 24 tasks in five categories, with 50 human demonstrations and 50 evaluation episodes per task. In RoboCasa H50, third-person camera viewpoints are randomized across demonstrations during training and fixed to a single reference viewpoint during evaluation. We fine-tune two pretrained VLAs that differ in scale and action expert, $\pi_{0.5}$ [4] and SmolVLA [53]; for each backbone, the RGB and RGB+Pointmap variants differ only in their visual input.

Baselines All baselines except FP3 share the $\pi_{0.5}$ backbone, so differences isolate how each injects viewpoint or 3D information rather than the backbone. We compare three groups. **(i) Camera-aware VLAs:** KYC [16] feeds the policy per-pixel Plücker rays from the camera intrinsics and extrinsics, and OC-VLA [19] predicts actions in the camera frame. **(ii) 3D-augmented VLAs:** GeoVLA [26] and PointVLA [27] attach point clouds through a dedicated 3D module. **(iii) point cloud policy:** FP3 [54] is a DROID-pretrained policy that consumes point clouds instead of RGB. We reimplement KYC, OC-VLA, GeoVLA, and PointVLA on $\pi_{0.5}$, as official code is unavailable. Full implementation and training details are given in the supplementary material (Appendix B.4).

Results Adding pointmaps improves both pretrained backbones (Table 4). For $\pi_{0.5}$, the 24-task average rises from 55.3 to 62.9, with gains across all five task categories. For SmolVLA, it rises from 37.2 to 41.4. The two backbones differ in scale and action expert, so the improvement is not tied to a single architecture. The camera-aware and 3D-augmented baselines share the $\pi_{0.5}$ backbone with our method. The strongest camera-aware baseline (KYC, 59.1) and the strongest 3D-augmented baseline (PointVLA, 57.3) both remain below $\pi_{0.5}$ with pointmaps (62.9). Conditioning the policy on camera parameters or attaching a separate 3D module is thus less effective than expressing the geometry in the robot frame and fusing it on the RGB grid, where the pretrained visual pathway absorbs it directly. FP3, the only baseline that does not share the $\pi_{0.5}$ backbone, trails all $\pi_{0.5}$ -based methods at 42.8. Rather than building on a 2D VLA, it uses a separately pretrained point-cloud policy and does not inherit the large-scale vision-language pretraining a 2D VLA provides. In addition, its sparse point-cloud input (about 4k points per view) may be less suited to tasks needing precise placement. Full per-task results are in Table 8.

5.2 Real-world Experiments

Experimental Setup We evaluate on a Franka Research 3 with two RealSense cameras (Fig. 7): a wrist-mounted D405 and a single external D435i. During data collection, the wrist camera stays fixed

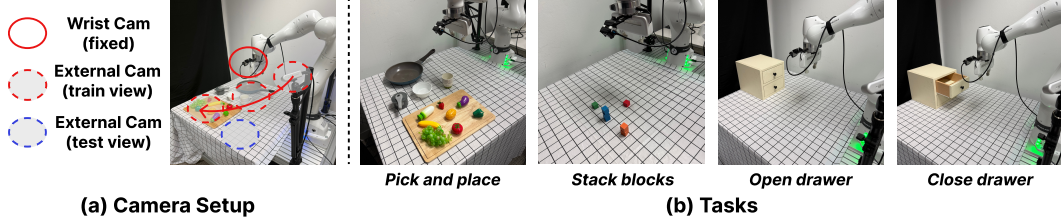


Figure 7: **Real robot setup.** (a) A Franka Research 3 with a fixed wrist camera and one external camera, placed at three training positions (red) and one held-out test position (blue). (b) Tasks: pick-and-place, stack blocks, open/close drawer.

while we physically reposition the D435i to three placements, giving three camera configurations (Fig. 7a). Under each configuration, we collect 15 demonstrations per task on four tasks (pick-and-place, stack blocks, open drawer, close drawer), for 60 demonstrations per configuration and 180 in total. To measure robustness to camera-viewpoint change, we use a seen/unseen protocol at evaluation. The *seen* setting places the external camera at one of the three training placements, and the *unseen* setting uses a held-out placement displaced along the direction shown in Fig. 7. We report success on the four tasks under both placements in Table 5.

Baselines We compare $\pi_{0.5}$ with pointmap against two references, both trained on the same demonstrations. RGB-only $\pi_{0.5}$ [4] shares our backbone and differs only in the visual input, isolating the effect of adding the pointmap. DP3 [55] is a point cloud diffusion policy trained from scratch on 3D points instead of RGB. We include it because it already receives metric 3D geometry that can be expressed in the robot frame, yet obtains it without a pretrained VLA backbone and in point cloud rather than image form. It therefore tests whether our gains require the specific design of expressing robot-frame geometry as an image-form pointmap fused into a pretrained VLA, or whether access to 3D geometry alone is sufficient. As a policy that operates natively in 3D, DP3 also serves as a reference for how well a 3D input resists camera-viewpoint shift, which is central to our seen/unseen comparison.

Table 5: **Real-world camera generalization.** All models are evaluated 15 times under *seen* and *unseen* camera configurations across four manipulation tasks (success rate, %).

Eval. camera	Model	Avg.	Pick-and-place	Stack blocks	Open drawer	Close drawer
Seen	DP3 [55]	63.3	60.0	40.0	60.0	93.3
	$\pi_{0.5}$ [4]	73.3	80.0	53.3	73.3	86.7
	$\pi_{0.5}$ + pointmap	78.3	86.7	60.0	73.3	93.3
Unseen	DP3 [55]	48.3	33.3	33.3	40.0	86.7
	$\pi_{0.5}$ [4]	55.0	40.0	26.7	66.7	86.7
	$\pi_{0.5}$ + pointmap	66.7	53.3	46.7	73.3	93.3

Results When the external camera is placed at one of the training placements (*Seen* viewpoint), pointmaps improve $\pi_{0.5}$ from 73.3 to 78.3, a +5.0 margin over the RGB-only backbone. Both VLA variants also outperform the point cloud policy DP3 (63.3), so at a seen placement the pretrained RGB backbone alone is already competitive, and the pointmap adds a further gain. At a held-out camera placement (*Unseen* viewpoint), the advantage grows. RGB-only $\pi_{0.5}$ drops from 73.3 to 55.0, whereas $\pi_{0.5}$ with pointmap holds at 66.7, so the pointmap degrades by only 11.6 points against 18.3 for RGB-only. Its margin over RGB therefore widens from +5.0 at the seen placement to +11.7 at the unseen one. DP3 shows the same effect from the opposite direction. The RGB-only lead over DP3 narrows from +10.0 to +6.7, and DP3 overtakes RGB-only on Stack blocks (33.3 vs. 26.7), showing that a 3D input already resists viewpoint shift better than RGB.

These results point to a broader advantage. Pointmaps are particularly useful when the training data spans many camera viewpoints. Because each scene point is assigned a fixed robot-frame coordinate, that coordinate stays the same as the camera is repositioned across demonstrations. An RGB-only policy must instead learn how each viewpoint maps to robot-frame actions, so wider viewpoint coverage makes this observation-to-action relation harder to fit. Pointmaps supply the robot-frame geometry directly, letting the policy learn from viewpoint-diverse demonstrations without absorbing that variation itself. This matches the simulation trend in Section 4.4, where RGB-only success falls as training-time viewpoint variation increases while RGB + Pointmap stays nearly flat, and it explains why the real-robot advantage widens at the held-out placement.

6 Limitations

Our study leaves several questions open. We do not ablate how the pointmap is injected relative to the action expert, nor how it interacts with the pretraining recipe, so the best way to combine the two remains unsettled. The comparison against a point cloud uses a single sampling budget, and a larger budget could narrow the gap we report. In addition, pointmaps require calibrated camera intrinsics and extrinsics at both training and test time, which restricts them to setups where calibration is available. Finally, our camera-variation results focus on changes in camera placement and extrinsics, and do not yet cover changes in the number of cameras or their fields of view, which we leave to future work.

7 Conclusion

We studied how VLAs should use 3D observations when demonstrations are collected under various camera viewpoints. In this setting, the same task can be observed from different camera viewpoints during training, requiring the policy to relate image observations to the robot’s action frame. Rather than leaving this camera-to-robot relationship for the policy to infer from RGB, depth, and calibration inputs, we express each observed scene point directly in the robot frame as a robot-centric pointmap. The pointmap preserves the dense $H \times W$ image grid, allowing robot-centric 3D coordinates to be fused with RGB tokens through the existing visual pathway of a pretrained VLA.

Our experiments point to a simple and effective input: a robot-frame, image-form pointmap centered on the end effector. Robot-centric pointmaps improve pretrained VLAs in both simulation and real-robot experiments. These results suggest a general principle for 3D-aware VLAs: when calibration is available, express observations in the coordinate system where the robot acts before passing them to the policy. Robot-centric pointmaps implement this principle while retaining the image-form structure that pretrained VLAs already use.

References

- [1] B. Zitkovich et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *CoRL*, 2023. 1
- [2] M. J. Kim et al. Openvla: An open-source vision-language-action model. In *CoRL*, 2024.
- [3] K. Black et al. π_0 : A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024. 4
- [4] K. Black, N. Brown, et al. pi0.5: a vision-language-action model with open-world generalization. In *CoRL*, 2025. 8, 9, 15, 20
- [5] J. Bjorck et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025. 1
- [6] A. O’Neill et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration. In *ICRA*, 2024. 1, 3
- [7] Q. Bu et al. Agibot world colosseo: A large-scale manipulation platform for scalable and intelligent embodied systems. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2025.
- [8] A. Khazatsky et al. DROID: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024. 1, 3
- [9] S. Nasiriany, A. Maddukuri, L. Zhang, A. Parikh, A. Lo, A. Joshi, A. Mandlekar, and Y. Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. In *Robotics: Science and Systems (RSS)*, 2024. 2, 4, 16
- [10] S. Fei, S. Wang, J. Shi, Z. Dai, J. Cai, P. Qian, L. Ji, X. He, S. Zhang, Z. Fei, J. Fu, J. Gong, and X. Qiu. Libero-plus: In-depth robustness analysis of vision-language-action models. *arXiv preprint arXiv:2510.13626*, 2025. 2
- [11] Z. Wang, Z. Zhou, J. Song, Y. Huang, Z. Shu, and L. Ma. Vlatest: Testing and evaluating vision-language-action models for robotic manipulation. *Proc. ACM Softw. Eng.*, 2(FSE), June 2025. doi:10.1145/3729343. URL <https://doi.org/10.1145/3729343>. 2
- [12] H. Walke, K. Black, A. Lee, M. J. Kim, M. Du, C. Zheng, T. Zhao, P. Hansen-Estruch, Q. Vuong, A. He, V. Myers, K. Fang, C. Finn, and S. Levine. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning (CoRL)*, 2023. 3
- [13] A. Xie, L. Lee, T. Xiao, and C. Finn. Decomposing the generalization gap in imitation learning for visual robotic manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024. 3
- [14] J. Morgan, P. Vijay, H. Oh, J. Song, A. Arora, A. Du, G. Sukhatme, J. Thomason, and I. Singh. Colosseum v2: Benchmarking generalization for vision language action models. *arXiv preprint arXiv:2605.27759*, 2026. 3
- [15] H. Lin, H. Yu, J. Huang, H. Zhang, Y. Ling, P. Tan, X. Xue, and Y. Fu. Universal pose pre-training for generalizable vision-language-action policies. In *Robotics: Science and Systems (RSS)*, 2026. 3
- [16] T. Jiang, J. Ji, X. Tan, J. Fang, A. Bhattad, V. Guizilini, and M. R. Walter. Do you know where your camera is? View-invariant policy learning with camera conditioning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2026. 8, 18, 20
- [17] J. Zhang, Y. Chen, Y. Xu, Z. Huang, Y. Zhou, Y.-J. Yuan, X. Cai, G. Huang, X. Quan, H. Xu, and L. Zhang. 4d-VLA: Spatiotemporal vision-language-action pretraining with cross-scene calibration. In *Proc. the Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

- [18] D. Qu, H. Song, Q. Chen, Y. Yao, X. Ye, Y. Ding, Z. Wang, J. Gu, B. Zhao, D. Wang, et al. Spatialvla: Exploring spatial representations for visual-language-action model. *arXiv preprint arXiv:2501.15830*, 2025. 3
- [19] T. Zhang, H. Duan, H. Hao, Y. Qiao, J. Dai, and Z. Hou. Grounding actions in camera space: Observation-centric vision-language-action policy. In *Proc. the AAAI Conference on Artificial Intelligence (AAAI)*, 2026. 3, 8, 18, 20
- [20] S. Tian, B. Wulfe, K. Sargent, K. Liu, S. Zakharov, V. C. Guizilini, and J. Wu. View-invariant policy learning via zero-shot novel view synthesis. In *Conference on Robot Learning (CoRL)*, 2024. 3
- [21] L. Y. Chen, C. Xu, K. Dharmarajan, R. Cheng, K. Keutzer, M. Tomizuka, Q. Vuong, and K. Goldberg. Rovi-aug: Robot and viewpoint augmentation for cross-embodiment robot learning. In *Conference on Robot Learning (CoRL)*, 2024. 3
- [22] L. Y. Chen, K. Hari, K. Dharmarajan, C. Xu, Q. Vuong, and K. Goldberg. Mirage: Cross-embodiment zero-shot policy transfer with cross-painting. In *Robotics: Science and Systems (RSS)*, 2024. 3
- [23] H. Heo, S. Woo, S. M. Kim, J. Kim, J. Lee, Y. Lee, and Y. M. Kim. Anycamvla: Zero-shot camera adaptation for viewpoint robust vision-language-action models, 2026. 3
- [24] F. Liu, F. Yan, L. Zheng, Y. Huang, C. Feng, and L. Ma. Robouniview: Visual-language model with unified view representation for robotic manipulation. *arXiv preprint arXiv:2406.18977*, 2024. 3
- [25] A. Wilcox, M. Ghanem, M. Moghani, P. Barroso, B. Joffe, and A. Garg. Adapt3r: Adaptive 3d scene representation for domain transfer in imitation learning. In *Conference on Robot Learning (CoRL)*, 2025. 3
- [26] L. Sun, B. Xie, Y. Liu, H. Shi, T. Wang, and J. Cao. Geovla: Empowering 3d representations in vision-language-action models. In *arXiv preprint arXiv:2508.09071*, 2025. 3, 8, 17, 20
- [27] C. Li, J. Wen, Y. Peng, Y. Peng, F. Feng, and Y. Zhu. Pointvla: Injecting the 3d world into vision-language-action models. *IEEE Robotics and Automation Letters*, 11(3):2506–2513, 2026. 8, 18, 20
- [28] V. Bhat, Y.-H. Lan, P. Krishnamurthy, R. Karri, and F. Khorrami. 3d cavla: Leveraging depth and 3d context to generalize vision language action models for unseen tasks. In *arXiv preprint arXiv:2505.05800*, 2025.
- [29] T. Yuan, Y. Liu, C. Lu, Z. Chen, T. Jiang, and H. Zhao. Depthvla: Enhancing vision-language-action models with depth-aware spatial reasoning. In *arXiv preprint arXiv:2510.13375*, 2025.
- [30] X. Fan, S. Deng, X. Wu, Y. Lu, Z. Li, M. Yan, Y. Zhang, Z. Zhang, H. Wang, and H. Zhao. Any3d-vla: Enhancing vla robustness via diverse point clouds. *arXiv preprint arXiv:2602.00807*, 2026.
- [31] S. Chen, P. Pacaud, and C. Schmid. Pointact: Vision-language-action models with multi-scale point-action interaction, 2026. URL <https://arxiv.org/abs/2605.21414>. 3
- [32] P. Li, Y. Chen, H. Wu, X. Ma, X. Wu, Y. Huang, L. Wang, T. Kong, and T. Tan. Bridgevla: Input-output alignment for efficient 3d manipulation learning with vision-language models. In *Proc. the Advances in Neural Information Processing Systems (NeurIPS)*, 2025. 3
- [33] I. Singh, A. Goyal, S. Birchfield, D. Fox, A. Garg, and V. Blukis. Og-vla: Orthographic image generation for 3d-aware vision-language action model. *arXiv preprint arXiv:2506.01196*, 2025. 3

- [34] H. Zhen, X. Qiu, P. Chen, J. Yang, X. Yan, Y. Du, Y. Hong, and C. Gan. 3d-vla: 3d vision-language-action generative world model. *Proc. the International Conference on Machine Learning (ICML)*, 2024. 3
- [35] T. Lin, G. Li, Y. Zhong, Y. Zou, and B. Zhao. Evo-0: Vision-language-action model with implicit spatial understanding. *arXiv preprint arXiv:2507.00416*, 2025. 3
- [36] F. Li, W. Song, H. Zhao, J. Wang, P. Ding, D. Wang, L. Zeng, and H. Li. Spatial forcing: Implicit spatial representation alignment for vision-language-action model. In *Proc. the International Conference on Learning Representations (ICLR)*, 2026. URL <https://openreview.net/forum?id=euMVC1D04k>. 3
- [37] S. Wang, V. Leroy, Y. Cabon, B. Chidlovskii, and J. Revaud. Dust3r: Geometric 3d vision made easy. In *Proc. of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 20697–20709, June 2024. 3
- [38] V. Leroy, Y. Cabon, and J. Revaud. Grounding image matching in 3d with mast3r. In *Proc. of the European Conference on Computer Vision (ECCV)*, 2024.
- [39] J. Wang, M. Chen, N. Karaev, A. Vedaldi, C. Rupprecht, and D. Novotny. Vggt: Visual geometry grounded transformer. In *Proc. of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 5294–5306, June 2025.
- [40] R. Wang, S. Xu, C. Dai, J. Xiang, Y. Deng, X. Tong, and J. Yang. Moge: Unlocking accurate monocular geometry estimation for open-domain images with optimal training supervision. In *Proc. of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 5261–5271, 2025.
- [41] R. Wang, S. Xu, Y. Dong, Y. Deng, J. Xiang, Z. Lv, G. Sun, X. Tong, and J. Yang. Moge-2: Accurate monocular geometry with metric scale and sharp details. In *Proc. the Advances in Neural Information Processing Systems (NeurIPS)*, 2025.
- [42] Q. Wang*, Y. Zhang*, A. Holynski, A. A. Efros, and A. Kanazawa. Continuous 3d perception model with persistent state. In *Proc. of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2025.
- [43] J. Yang, A. Sax, K. J. Liang, M. Henaff, H. Tang, A. Cao, J. Chai, F. Meier, and M. Feiszli. Fast3r: Towards 3d reconstruction of 1000+ images in one forward pass. In *Proc. of the IEEE conference on computer vision and pattern recognition (CVPR)*, June 2025.
- [44] Y. Wang, J. Zhou, H. Zhu, W. Chang, Y. Zhou, Z. Li, J. Chen, J. Pang, C. Shen, and T. He. π^3 : Permutation-equivariant visual geometry learning. In *Proc. the International Conference on Learning Representations (ICLR)*, 2026. 3
- [45] X. Jia, Q. Wang, A. Wang, H. A. Wang, B. Gyenes, E. Gospodinov, X. Jiang, G. Li, H. Zhou, W. Liao, X. Huang, M. Beck, M. Reuss, R. Lioutikov, and G. Neumann. Pointmappolicy: Structured point cloud processing for multi-modal imitation learning. In *Proc. the Advances in Neural Information Processing Systems (NeurIPS)*, 2025. 3
- [46] X. Yang, R. Wu, J. Liu, and X. Li. Remap-dp: Reprojected multi-view aligned pointmaps for diffusion policy, 2026. 3
- [47] L. Beyer et al. Paligemma: A versatile 3b vlm for transfer. *arXiv preprint arXiv:2407.07726*, 2024. 4
- [48] X. Wu, L. Jiang, P.-S. Wang, Z. Liu, X. Liu, Y. Qiao, W. Ouyang, T. He, and H. Zhao. Point transformer v3: Simpler, faster, stronger. In *Proc. of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2024. 6, 19

- [49] M. Liu, X. Li, Z. Ling, Y. Li, and H. Su. Frame mining: a free lunch for learning robotic manipulation from 3d point clouds. In *Conference on Robot Learning (CoRL)*, 2022. 6
- [50] L. Ke, J. Wang, T. Bhattacharjee, B. Boots, and S. Srinivasa. Grasping with chopsticks: Combating covariate shift in model-free imitation learning for fine manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, page 6185–6191. IEEE Press, 2021. doi:10.1109/ICRA48506.2021.9561662. URL <https://doi.org/10.1109/ICRA48506.2021.9561662>.
- [51] K. Rana, J. Abou-Chakra, S. Garg, R. Lee, I. Reid, and N. Suenderhauf. Learning from 10 demos: Generalisable and sample-efficient policy learning with oriented affordance frames. In *Conference on Robot Learning (CoRL)*, 2025. URL <https://openreview.net/forum?id=1K3kjo91Q1>.
- [52] Z. Zhang, Z. Xu, J. N. Lakamsani, and Y. She. Canonical policy: Learning canonical 3d representation for equivariant policy. *arXiv preprint arXiv:2505.18474*, 2025. 6
- [53] M. Shukor, D. Aubakirova, F. Capuano, P. Kooijmans, S. Palma, A. Zouitine, M. Aractingi, C. Pascal, M. Russi, A. Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025. 8, 15, 20
- [54] R. Yang, G. Chen, C. Wen, and Y. Gao. Fp3: A 3d foundation policy for robotic manipulation, 2025. URL <https://arxiv.org/abs/2503.08950>. 8, 18, 20
- [55] Y. Ze, G. Zhang, K. Zhang, C. Hu, M. Wang, and H. Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. In *Robotics: Science and Systems (RSS)*, 2024. 9, 19
- [56] X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 11975–11986, 2023. 17

Supplementary Material

A Details of Training-Time Camera Viewpoint Variation

Section 4.4 of the main paper varies how much the third-person camera poses change across training demonstrations, using three settings: *No variation*, *Low*, and *High*. This section specifies these settings and visualizes them.

RoboCasa mounts three cameras: two third-person cameras providing a left and a right view, and a wrist camera. At the start of each demonstration, we jitter the position and orientation of the two third-person cameras with zero-mean Gaussian noise. *No variation* applies no jitter, so every demonstration shares the canonical camera pose. *Low* uses a position and orientation standard deviation of (5cm, 3°), which matches RoboCasa’s default per-demonstration camera randomization. *High* doubles the jitter to (10cm, 6°). The wrist camera is mounted on the end-effector and is never jittered, and the evaluation cameras are held fixed, so the per-demonstration jitter of the two third-person cameras is the only factor that changes across the training sets. Each setting thus defines a distinct training set, on which we train a separate policy.

Fig. 8 visualizes the camera-pose distributions as frustums in the robot-base frame. Fig. 9 shows qualitative examples of an episode rendered under each variation level.

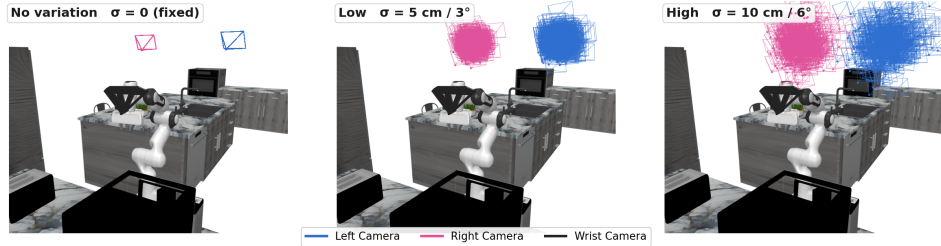


Figure 8: **Camera-pose distributions across variation levels.** Per-demonstration poses of the two third-person cameras, drawn as camera frustums in the robot-base frame (left camera in blue, right in pink; the end-effector wrist camera, in black, is never jittered). From *No variation* to *High* the position and orientation jitter grows from zero to (10cm, 6°), widening the range of viewpoints the policy sees.

B Implementation Details and Experimental Setup

B.1 Backbones and training

We study two pretrained VLAs that differ in scale and action expert, $\pi_{0.5}$ [4] and SmolVLA [53]. For each backbone, we train two policies that are identical except for the visual input: RGB alone versus RGB with the robot-centric pointmap, so every reported difference is attributable to the pointmap. Fig. 10 shows the full model architecture. Both are fine-tuned with AdamW at a peak learning rate of 10^{-4} under a cosine schedule with 5% linear warmup, in bfloat16 at an effective batch size of 64. We train $\pi_{0.5}$ for 20k steps and SmolVLA for 60k steps. Each policy reads RGB at its backbone’s native resolution and predicts a chunk of end-effector deltas. Table 6 lists the full configuration.

Controlled-analysis protocol. For the controlled studies in Section 4, we use a single π -style architecture initialized from a base PaliGemma checkpoint with an action expert trained from scratch, rather than the robot-pretrained π checkpoint, isolating the effect of the input design. All variants share the same RoboCasa training recipe: 24 atomic tasks, 50 human demonstrations per task, 30k training steps, and 50 evaluation episodes per task.

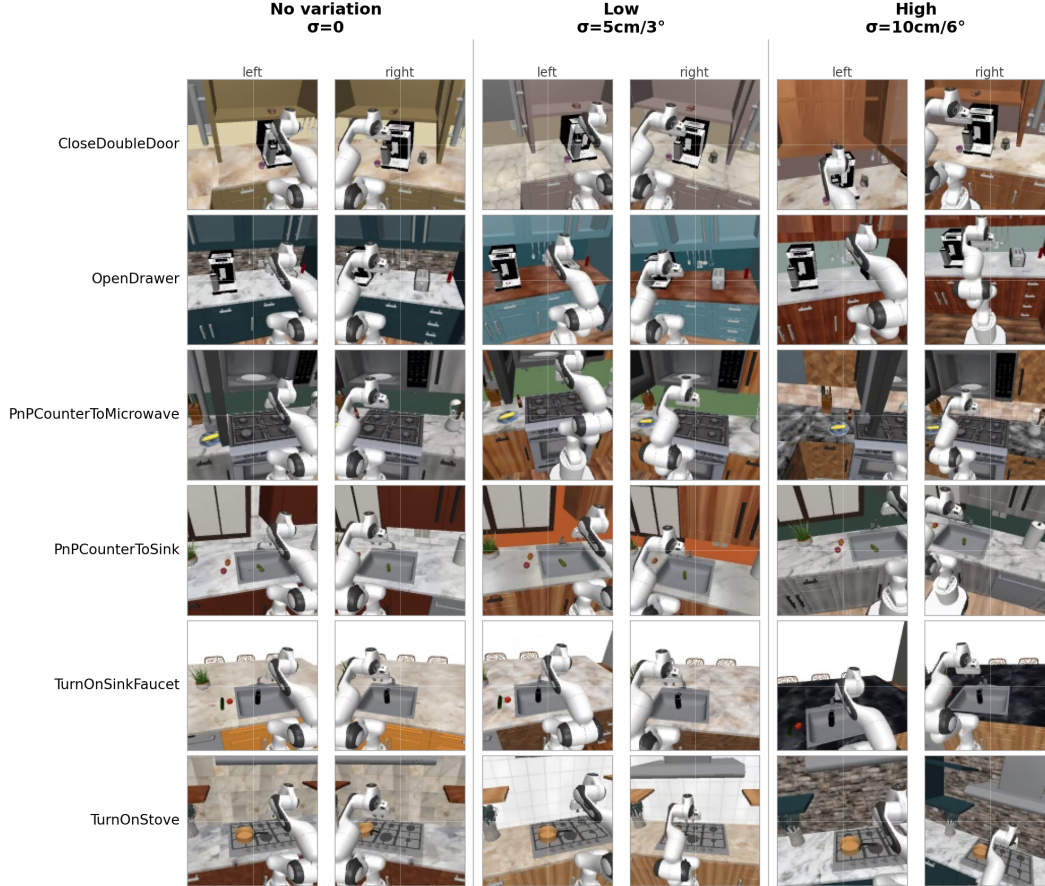


Figure 9: **Qualitative examples across variation levels.** For six tasks, the same frame is rendered under each training level for the left and right third-person cameras. Under *No variation*, the viewpoint is identical across demonstrations; under *Low* and *High*, growing per-demonstration jitter shifts and rotates the scene within the image. The robot-centric pointmap reduces this camera-induced variation by expressing the observed geometry in the robot frame.

B.2 Simulation setup (RoboCasa)

We follow the RoboCasa [9] atomic-task protocol with 24 environments, grouped into five categories for reporting: doors, drawers, coffee, pick-and-place, and turn objects. Each environment provides 50 human demonstrations and is evaluated over 50 episodes. The policy reads three cameras: two third-person cameras and one wrist camera. During training, the third-person camera viewpoints are randomized across demonstrations. Unless otherwise stated, evaluation uses a fixed reference viewpoint. Appendix D additionally reports a separate randomized-viewpoint evaluation.

B.3 Real-robot setup

We use a Franka Research 3 (FR3) with two Intel RealSense cameras, a wrist-mounted D405 and a single external D435i. During data collection the wrist camera stays fixed and the D435i is physically repositioned to three placements, yielding three camera configurations. Under each configuration we collect 15 demonstrations per task across four tasks (pick-and-place, stack blocks, open drawer, close drawer), for 60 demonstrations per configuration and 180 in total. Low-level control runs at 20 Hz, and the policy executes 10 of each 20-step action chunk before replanning. At evaluation we test under one of the three external placements seen during training (*seen*) and under a held-out placement displaced along the direction indicated in Fig. 7 (*unseen*). Demonstrations are collected by human teleoperation with VR controller (Meta Quest 3). The pointmap is built from the RealSense

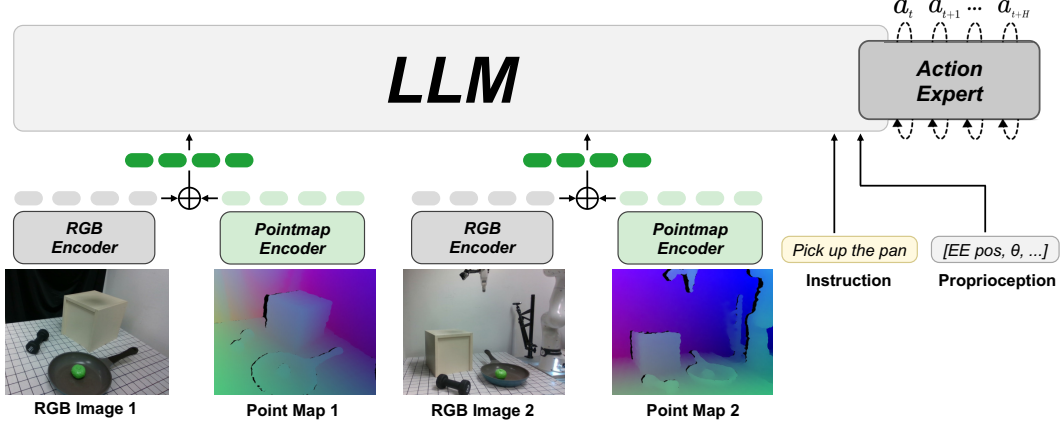


Figure 10: **Model architecture.** Each RGB-D camera observation is converted into an end-effector-centered robot-centric pointmap. RGB and pointmap inputs are encoded by separate SigLIP vision towers [56], and pointmap tokens are added element-wise to the corresponding RGB tokens before entering the pretrained VLA. The VLA predicts an action chunk from the fused visual tokens, language instruction, and proprioception.

Table 6: **Training and deployment hyperparameters**

Setting	$\pi_{0.5}$	SmolVLA
Optimizer	AdamW	AdamW
Peak learning rate	10^{-4}	10^{-4}
LR schedule	cosine, 5% warmup	cosine, 5% warmup
Precision	bfloat16	bfloat16
Effective batch size	64	64
Training steps	20k	60k
Action chunk (sim)	50 (execute 25)	50 (execute 25)
Action chunk (real)	20 (execute 10)	—
Control frequency (real)	20 Hz	—
Pointmap fusion	element-wise add	element-wise add

stereo depth and a one-time hand-eye calibration, so unlike the exact simulation geometry, it carries sensor noise.

B.4 Baseline implementation details

The main paper compares our pointmap against camera-aware, 3D-augmented, and point-cloud baselines on RoboCasa (Table 4). GeoVLA, PointVLA, KYC, and OC-VLA are re-implemented on the $\pi_{0.5}$ backbone, and use the same RoboCasa data (24 tasks, 50 demonstrations each), a batch size of 64, and the same 50-episode evaluation. The one exception is FP3, a standalone, point-cloud based 3D foundation policy that does not share the $\pi_{0.5}$ backbone. We describe each baseline and the implementation choices used in our experiments below.

GeoVLA [26]. GeoVLA augments a VLA with a dedicated 3D branch that encodes the scene point cloud into a geometric anchor token injected into the action expert. Since GeoVLA is not open-sourced, we reimplement its method on our $\pi_{0.5}$ backbone. Following GeoVLA, we convert depth into an end-effector-centered point cloud, encode it with a point-embedding network that uses paired geometric and positional paths, and inject the resulting anchor token into the action expert. The SigLIP vision tower stays frozen, while the action expert, language model, and point encoder are trained from the $\pi_{0.5}$ initialization for the same 20k steps. We sweep three anchor-selection variants:

Table 7: **The 24 RoboCasa environments by category.** Categories match the columns of the main-paper RoboCasa table.

Category	# tasks	Environments
Doors	4	CloseDoubleDoor, CloseSingleDoor, OpenDoubleDoor, OpenSingleDoor
Drawers	2	CloseDrawer, OpenDrawer
Coffee	3	CoffeePressButton, CoffeeServeMug, CoffeeSetupMug
Pick-and-place	8	PnPCounterToCabinet, PnPCabinetToCounter, PnPCounterToSink, PnPSinkToCounter, PnPCounterToMicrowave, PnPMicrowaveToCounter, PnPCounterToStove, PnPStoveToCounter
Turn objects	7	TurnOffMicrowave, TurnOffSinkFaucet, TurnOffStove, TurnOnMicrowave, TurnOnSinkFaucet, TurnOnStove, TurnSinkSpout
Total	24	

argmin, top- K , and ground-truth end-effector. The best configuration uses a top- $K=4$ anchor with a fixed wrist cell.

PointVLA [27]. PointVLA injects point-cloud features into a pretrained VLA through a subset of its action-expert blocks, using a two-stage schedule so the 2D policy is set before 3D is added. PointVLA is likewise unreleased, so we reimplement its mechanism on $\pi_{0.5}$. We encode a point cloud from a fixed exocentric camera with an iDP3-style hierarchical encoder and inject it into a subset of the action-expert blocks, following PointVLA’s skip-block design. The vision tower and the action-expert body are frozen while the language model and the point encoder train. Following PointVLA’s two-stage recipe, we first train the RGB-only $\pi_{0.5}$ backbone and then run 20k additional steps in which the point-cloud encoder is injected while the vision tower and action-expert body stay frozen.

FP3 [54]. FP3 is a 3D foundation policy that acts directly from point clouds and is pretrained on large-scale robot data, rather than augmenting a 2D VLA. Unlike GeoVLA and PointVLA, FP3 is a standalone 3D foundation policy with public weights. Instead, we fine-tune the authors’ DROID-pretrained checkpoint on our RoboCasa data. FP3 pairs a Uni3D point-cloud encoder with a diffusion-transformer head, and we fine-tune it for the same 20k steps. We keep FP3’s two point-cloud streams: the wrist cloud is passed to its hand-camera encoder, and the two third-person clouds are merged into a single cloud for its external-camera encoder. Among the variants we tried, full fine-tuning with learning rate 10^{-4} performed best, so we report this setting.

KYC [16]. KYC conditions the policy on camera geometry using per-pixel Plücker rays. For each camera, we compute a six-channel Plücker map from the camera intrinsics and extrinsics, with ray direction and moment expressed in the robot-base frame. We encode this map with a SigLIP vision tower initialized from the $\pi_{0.5}$ vision encoder, using a learnable 1×1 convolution to map the six Plücker channels into the three-channel patch embedding. The resulting tokens are added element-wise to the RGB tokens, matching the fusion used by our pointmap encoder. All weights, including the ray encoder, are trained for the same 20k steps as the other matched baselines. We report the strongest variant.

OC-VLA [19]. OC-VLA expresses the action label in the camera frame instead of the robot-base frame. OC-VLA has no public code, so we reimplement its mechanism on $\pi_{0.5}$. We retarget the predicted end-effector action from the robot-base frame into the frame of the static left agent-view camera, while leaving the RGB input and the $\pi_{0.5}$ architecture unchanged. Since the policy predicts operational-space delta actions, we apply only the rotation between the robot-base and camera frames, without adding a translation offset. The training and evaluation recipe matches the other matched-backbone baselines.

DP3 [55]. On the real robot, we compare against DP3, a point-cloud diffusion policy trained from scratch. We build its input by back-projecting the sensed depth into a robot-frame point cloud and subsampling it to 4096 points using the official codebase. DP3 is trained on the same demonstrations as the VLA baselines and evaluated under the same seen and unseen camera conditions. Like our pointmap, DP3 receives metric 3D geometry in the robot frame, so it tests whether access to robot-frame 3D geometry alone is sufficient without the image-form integration of a pretrained VLA.

Point cloud and Point Transformer v3. In Section 4.2, the point-cloud branch uses either a lightweight MLP set encoder over 1024 points per camera or a pretrained Point Transformer v3 [48] encoder. Both consume the same robot-frame 3D geometry as the pointmap. Since point clouds do not preserve pixel correspondence with RGB, their encoded features are concatenated to the RGB token sequence rather than fused element-wise.

Plücker rays and depth. In Section 4.1, the model is given per-pixel depth together with six-dimensional Plücker rays that encode the camera ray of each pixel from the intrinsics and extrinsics. These channels provide the same depth and calibration information used to construct the pointmap, but the policy must still infer the robot-frame coordinates from them.

B.5 Evaluation metric and protocol

We report the task success rate in percent. In simulation, each task is evaluated over 50 episodes. On the real robot, each task is evaluated over 15 rollouts per camera condition. All results are reported from a single final checkpoint, without selecting the best checkpoint per method.

C Full RoboCasa Results

Table 8: **Full RoboCasa results** (success rate %, fixed evaluation viewpoint). Per-task breakdown of Table 4. All methods except FP3 use the $\pi_{0.5}$ backbone. PMP denotes our pointmap, added to the backbone in the preceding column (shaded).

Category	Task	FP3	OC-VLA	KYC	GeoVLA	PointVLA	$\pi_{0.5}$	+PMP	SmoVLA	+PMP
Doors	CloseDoubleDoor	76	80	84	70	86	76	80	74	92
	CloseSingleDoor	86	74	98	90	90	80	96	76	90
	OpenDoubleDoor	94	88	96	100	100	98	100	72	84
	OpenSingleDoor	60	78	68	64	74	64	84	50	54
Drawers	CloseDrawer	94	96	94	96	94	96	96	86	94
	OpenDrawer	54	64	78	64	76	70	84	40	60
Coffee	CoffeePressButton	80	38	68	58	50	46	76	74	70
	CoffeeServeMug	64	62	62	54	50	48	64	36	34
	CoffeeSetupMug	20	26	24	24	32	28	34	8	10
Pick-and-place	PnPCounterToCabinet	22	46	42	40	42	40	42	2	12
	PnPCabinetToCounter	38	34	26	28	14	16	28	8	6
	PnPCounterToSink	14	64	62	52	54	56	60	6	24
	PnPSinkToCounter	18	58	60	62	70	54	58	18	30
	PnPCounterToMicrowave	36	44	42	40	44	44	50	4	12
	PnPMicrowaveToCounter	28	24	28	38	16	28	38	4	8
	PnPCounterToStove	0	76	66	62	68	60	70	6	6
	PnPStoveToCounter	18	58	60	76	62	70	76	4	4
Turn objects	TurnOffMicrowave	32	60	68	72	70	68	66	72	78
	TurnOffSinkFaucet	56	50	46	46	44	44	60	58	50
	TurnOffStove	8	4	8	6	8	12	8	14	14
	TurnOnMicrowave	34	50	58	62	72	66	70	68	60
	TurnOnSinkFaucet	58	50	42	56	42	44	46	24	26
	TurnOnStove	12	52	58	40	48	54	56	40	24
	TurnSinkSpout	24	76	80	70	68	66	68	50	52
Average		42.8	56.3	59.1	57.1	57.3	55.3	62.9	37.2	41.4

Table 8 breaks the category averages of Table 4 down into the 24 individual RoboCasa environments under the fixed evaluation viewpoint. Adding the pointmap improves or matches the RGB-only $\pi_{0.5}$ on 22 of the 24 tasks, including large gains on tasks that require precise spatial reasoning, such as CoffeePressButton (46 $\rightarrow$ 76) and CoffeeServeMug (48 $\rightarrow$ 64). $\pi_{0.5}$ with pointmap also achieves the highest average in each of the five task categories. The same pattern holds for SmolVLA, where the pointmap improves all six door and drawer tasks and nearly doubles the pick-and-place average (6.5 $\rightarrow$ 12.8).

D RoboCasa Results under Randomized Evaluation Viewpoints

Our main RoboCasa results use a fixed evaluation viewpoint, while the training demonstrations include randomized third-person camera viewpoints. As an additional test, we evaluate the same trained policies with the third-person evaluation viewpoint randomized independently in each episode. Table 9 reports the results.

The conclusions from the fixed-viewpoint evaluation carry over. Adding pointmaps improves both pretrained backbones, raising the 24-task average from 54.2 to 60.3 for $\pi_{0.5}$ and from 35.9 to 38.3 for SmolVLA. Under randomized evaluation viewpoints, $\pi_{0.5}$ with pointmap also remains above all camera-aware and 3D-augmented baselines.

Table 9: RoboCasa results (success rate %, randomized evaluation viewpoint) over 24 tasks in five categories. All methods except FP3 (standalone point-cloud) use the $\pi_{0.5}$ backbone. Shaded rows add our pointmap.

Method	Avg.	Doors	Drawers	Coffee	Pick-and-place	Turn objects
<i>Point-cloud policy</i>						
FP3 [54]	41.1	71.0	80.0	50.7	19.0	34.0
<i>Camera-aware VLAs</i>						
OC-VLA [19]	52.0	76.0	76.0	40.0	46.2	43.1
KYC [16]	51.5	72.0	87.0	32.0	41.8	49.1
<i>3D-augmented VLAs</i>						
GeoVLA [26]	57.3	78.5	86.0	48.7	50.5	48.6
PointVLA [27]	56.8	82.0	87.0	44.7	43.2	54.6
<i>Ours</i>						
$\pi_{0.5}$ [4]	54.2	75.5	87.0	39.3	43.0	51.7
+ pointmap	60.3	83.5	89.0	48.0	51.0	54.9
SmolVLA [53]	35.9	60.0	71.0	41.3	7.8	42.0
+ pointmap	38.3	75.5	72.0	26.0	12.2	42.6