

A Simulation Benchmark for Bimanual Manipulation

We chose RL Bench [14] as our choice of simulator since it is well-maintained by the research community and has been used in a number of prior works [13, 12, 67, 68, 69, 16], including PerAct [11], which is a core component of VoxAct-B.

A.1 Additional Simulation and Task Details

We extend RL Bench to support bimanual manipulation by incorporating an additional Franka Panda arm into the RL Bench’s training and evaluation pipelines. Importantly, we do not modify the underlying APIs of CoppeliaSim (the backend of RL Bench) to control the additional arm; consequently, the robot arms cannot operate simultaneously, resulting in a delay in their control. However, this limitation is acceptable as our tasks do not require real-time, dual-arm collaboration.

Moreover, we modify `Open Jar`, `Open Drawer`, and `Put Item in Drawer` to support bimanual manipulation: (1) adding an additional Franka Panda arm with a wrist camera; (2) adding new waypoints for the additional arm; (3) adjusting the front camera’s position to capture the entire workspace; (4) removing the left shoulder, right shoulder, and overhead cameras. The new tasks use a three-camera setup: front, left wrist, and right wrist. We also modify the data generation pipeline to use motion planning with the new waypoints, process RGB-D images and the new arm’s proprioception data (joint position, joint velocities, gripper open state, gripper pose), and include the $[x, y, z]$ position (world coordinates) of the object of interest.

The success conditions of these tasks have also been modified: for `Open Jar`, we define a proximity sensor in the jar bottle to detect whether an arm has a firm grasp of the jar (the gripper’s opening amount is between 0.5 and 0.93); for `Open Drawer`, we define a proximity sensor on the top of the drawer to detect whether an arm is stabilizing the drawer. While a robot arm could still “open” the drawer without the other arm’s stabilization, we would not classify it a success in `Open Drawer`.

B Real-World Experimental Details

Hardware Setup. An overview of the hardware setup is described in Section 5.3. Our perception system utilizes the D415 camera to capture RGB and depth images at a resolution of 1280×720 pixels, where the depth images contain values in meters. We apply zero-padding to these images, resulting in a resolution of 1280×1280 pixels. Hand-eye calibration is performed to determine the transformation matrices between the camera frame and the left robot base frame, as well as between the camera frame and the right robot base frame, using the [MoveIt Calibration](#) package. We use the [python-urx](#) library to control the robot arms. Additionally, I/O programming is employed to control the Robotiq grippers, as CB2 UR5 robots do not support URCaps.

Data Collection. We utilize the [GELLO teleoperation framework](#) to collect real-world demonstrations. Due to the lack of Real-Time Data Exchange (RTDE) protocol support in CB2 UR5s, a noticeable lag is present when operating the GELLO arms. For `Open Jar`, a dedicated function controls the gripper’s counterclockwise rotations for unscrewing the lid and lifting it into the air, mitigating the instability caused by latency. This function is triggered when the operator activates the GELLO arm’s trigger. Additionally, we found that fixing the stabilizing arm while the acting arm is in motion is crucial for effective policy learning, as it eliminates noise introduced by unintentional, slight movements of the stabilizing arm. Observations are recorded at a frequency of 2 Hz.

Training and Execution. For training, we use a higher value for `stopped_buffer_timesteps`, a hyper-parameter that determines how frequently keyframes are extracted from the continuous actions based on how long the joint velocities have been near 0 and the gripper state has not been changed, in PerAct’s keyframe extraction function to account for the slower movements of the robot arms due to latency compared to simulation. We apply the inverse of the transformation matrices obtained from hand-eye calibration to project each arm’s gripper position to the camera frame. Using the camera’s intrinsics and an identity extrinsic matrix, we construct the point cloud in the

Hyperparameter	ACT Value	Diffusion Policy Value
learning rate	3e-5	1e-4
weight decay (for transformer only)	-	1e-3
# encoder layers	4	-
# decoder layers	7	-
# layers	-	8
feedforward dimension	3200	-
hidden dimension	512	-
embedding dimension	-	256
# heads	8	4
chunk size	100	100
beta	10	-
dropout	0.1	-
attention dropout probability	-	0.3
train diffusion steps	-	100
test diffusion steps	-	100
ema power	-	0.75

Table 3: Combined hyperparameters of ACT and Diffusion Policy. A dash (“-”) indicates the absence of a hyperparameter for a given method.

camera frame, allowing both arms’ gripper positions and the voxel grid to reside in the same reference frame. For evaluation, we multiply the transformation matrices from hand-eye calibration by the policy’s predicted left and right gripper positions to obtain the tool center point positions in their respective robot base frames. We visualize each robot arm’s predicted gripper position in Open3D before executing it on the robot. Additionally, we conduct real-world experiments using a Ubuntu laptop without a GPU, resulting in significantly slower policy inference and robot execution times—from capturing an image observation to moving the robot arms—compared to a GPU setup. Consequently, this results in longer pauses between each robot execution and extended real-world videos, as demonstrated on our website.

C Additional Implementation Details

C.1 Baselines

We carefully tune the baselines and include the hyperparameters used in Table 3. We report the results of the best-tuned baselines in Table 1. For our three tasks, we found that for ACT, a chunk size of 100 worked well, consistent with the findings reported in [3]. The temporal aggregation technique did not improve performance in our tasks, so we disabled this feature. For Diffusion Policy, lower values (e.g., 16) of the action prediction horizon were inadequate, leading to agents getting stuck at certain poses and failing to complete the tasks, so we used an action prediction horizon of 100. We found the Time-series Diffusion Transformer to outperform the CNN-based Diffusion Policy on Open Drawer and Open Jar, while both of them achieved comparable success rates on Put Item in Drawer. We use a batch size of 32 for both methods, and the observation resolution is 128×128 (same as VoxAct-B). For Diffusion Policy, we use the same image augmentation techniques as in [3]. As shown in Table 4, the performance of ACT and Diffusion Policy progressively improves as more environment variations are removed. For VoxPoser, we modified the LLM prompts to work with our bimanual manipulation tasks. See our [VoxPoser prompts](#) for details. For Bimanual PerActs, we deliberately chose to use 100^3 voxels instead of the 50^3 voxels used in VoxAct-B. The increased number of voxels provides higher voxel resolution, which is essential for fine-grained bimanual manipulation. This is demonstrated in the VoxAct-B w/o VLM ablation, which only utilizes 50^3 voxels and shows a huge drop in performance compared to VoxAct-B.

Method	Open Jar		Open Drawer		Put Item in Drawer	
	FAS	FAS+NSV	FAS	FAS+NSV	FAS	FAS+NSV
Diffusion Policy	21.3	46.7	28.0	56.0	14.7	22.7
ACT w/Transformers	30.7	57.3	32.0	37.3	36.0	82.7

Table 4: Ablation results of ACT and Diffusion Policy trained on 100 demonstrations. “FAS” refers to the demonstrations with fixed acting and stabilizing arms (i.e., right acting and left stabilizing), while “FAS+NSV” refers to fixed acting and stabilizing and without size variation in the environment. We use the same validation and test data as the baseline comparison.