Language-Conditioned Robotic Vial Sorting with a Vision–Language–Action Policy

Sari Abdan

October 2, 2026

Abstract

Vision–language models understand scenes and instructions remarkably well, yet turning that understanding into precise physical manipulation remains an open problem. Vision–language–action (VLA) models address it by adding an action generating component to a pretrained vision–language model, but their demonstrations typically involve coarse tasks, such as moving a large object into a bowl, that tolerate millimetres of error. We study a more demanding case: sorting small, translucent plastic vials of three colours between two racks of narrow slots and a discard bin with a low-cost SO-101 arm, where each command names an exact source and destination position. Success requires an accurate grasp of a thin tube, a precise placement into a small hole, and a policy that handles many positions through many phrasings of the prompt. We fine-tune the $\boldsymbol{\pi_{0.5}}$ **VLA policy**, keeping its vision–language backbone frozen and training only its **action expert**, on **150 teleoperated demonstrations** that we release publicly, and we add a deterministic release step for the bin. In 79 autonomous, human-scored rollouts across 15 commands, the policy completes the full pick-and-place in 25% of cases. It acts on the wrong vial or slot in only 8% of rollouts, which shows that the prompt are generally understood/followed, while the initial grasp of the vial is missed in 54% and the vial is released off the destination target in 19%. Precise physical execution is therefore the main limit, and more grasp demonstrations per position together with closed-loop grasp correction are the most direct ways to raise it.

1 Introduction

Foundation models trained on web-scale data have changed how machines process images and language. Vision–language models (VLMs) such as PaliGemma [1], which combine an image encoder [18] with a language model, recognize objects, describe spatial layouts, answer questions about a scene, and follow detailed instructions written in natural language. This breadth makes them attractive for robotics: a robot that shares a model's understanding of the world could be told what to do in plain language and could recognize objects it has never seen. Understanding a scene, however, is a different capability from acting in it. To manipulate an object, a robot has to turn what it perceives into a precise sequence of motor commands, covering where to move, how to approach, and exactly when to close and open the gripper. Web data contains little of this information, so a VLM alone cannot control a robot arm.

Vision–language–action (VLA) models close this gap by pairing a pretrained VLM with a way of producing robot actions and by training the combination on robot demonstrations. Two families have emerged. RT-2 [4] and OpenVLA [8] express each action as a short sequence of discrete tokens, so that the language model predicts motion in the same way it predicts words. The π_0 family [2, 13] instead adds a dedicated component that generates continuous actions, which allows smooth motion

at high control rates. Both families are trained on large collections of robot data, for example the multi-robot Open X-Embodiment dataset [12], and they follow language commands for a range of manipulation skills.

The policy studied in this report is $\pi_{0.5}$, so its structure is worth describing in some detail (Figure 1). The first part is a VLM backbone, PaliGemma [1]. A SigLIP image encoder [18] converts each camera image into a set of visual tokens, and a Gemma language model of about two billion parameters reads these tokens together with the written command. In $\pi_{0.5}$ the robot's own state, meaning the current end-effector pose and gripper opening, is also converted into numbers and written into the text prompt, so the backbone sees the images, the command, and the state in one sequence. The result is a representation of what is in the scene and what the user is asking for. The second part is the *action expert*, a considerably smaller transformer. We know it is a Gemma model of about 300 million parameters because the LeRobot configuration of $\pi_{0.5}$ that we use specifies it as the action expert (`gemma_300m`). It takes no images itself. It attends to the backbone's representation and generates the robot's motion with flow matching [9]: starting from random noise, it refines a whole chunk of future actions, 50 time steps in our configuration, in ten small denoising steps, and the chunk is sent to the arm as a smooth trajectory. The division of labour is clear. The backbone supplies the understanding of the scene and the command, and the action expert supplies the motor skill. The model is pretrained on a large and varied mixture of robot and web data so that this combination works across many scenes [13], and the released weights can be fine-tuned to a new robot with a modest amount of data.

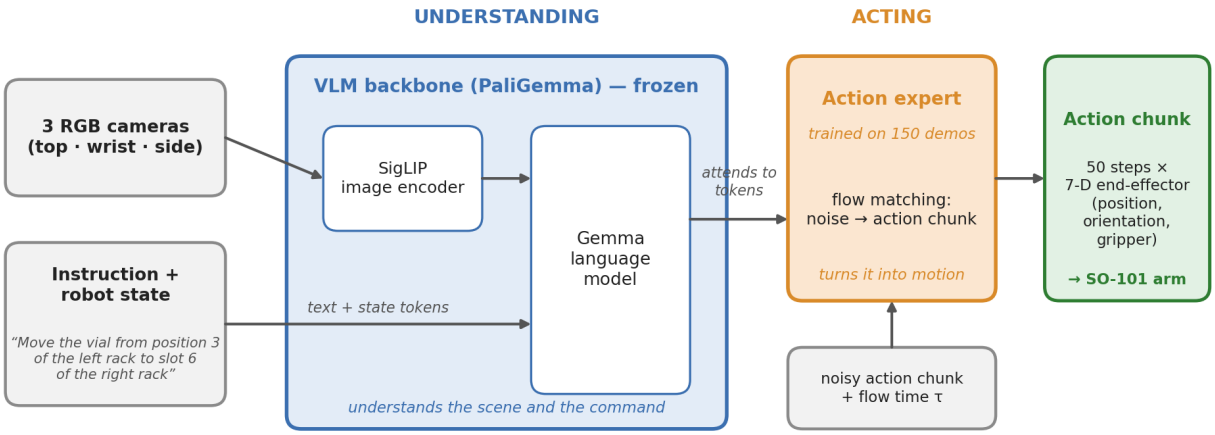


Figure 1: The $\pi_{0.5}$ vision–language–action policy used in this work. A frozen VLM backbone (SigLIP image encoder and Gemma language model) reads the three camera images and a text prompt holding the instruction and the discretized robot state. An action expert, the only part we train, attends to the backbone's tokens and, by flow matching, turns noise into a 50-step chunk of seven-dimensional end-effector actions for the SO-101 arm.

This division of labour suggests a direct question. The backbone brings a broad understanding of scenes and commands, and the action expert is trained on our demonstrations to produce the motion. How accurate is that motion? A task that demands a precise grasp and a precise placement measures exactly this, because every inaccuracy shows up as a missed grasp, a knocked neighbour, or a vial in the wrong slot. This report therefore studies how accurately a $\pi_{0.5}$ policy can grasp and place small objects in a demanding sorting task, and which stage of the motion limits its success.

Freezing the backbone has practical advantages as well. With 150 demonstrations recorded in

a fixed workspace, updating billions of parameters would risk overfitting the recorded scenes and erasing the language grounding learned in pretraining, whereas training the action expert alone keeps that grounding and regularizes the training. It also lowers the cost of training considerably: because the frozen backbone needs no gradients, the whole recipe fits on a single workstation GPU using about 19 GB of memory, which makes it realistic for a small laboratory.

Many demonstrations of VLA models use tasks with generous tolerances. The RT-1 evaluation [3] centres on picking up an object, moving it near another object, placing it into a receptacle, and opening or closing a drawer, and RT-2 [4] is tested with commands such as putting a strawberry into the correct bowl or picking up a bag that is about to fall. A bowl, a counter, or a drawer offers a large target, so a placement a few centimetres off can still succeed.

Sorting vials with an SO-101 arm places much higher demands on the policy, in three ways. The first is accuracy. The vials are thin plastic tubes, so the gripper has to close on a narrow cylinder, and each tube stands in a narrow rack slot with neighbouring slots close by, so the vial has to be lowered into exactly the right hole without touching its neighbours. The SO-101 is a 3D-printed arm driven by low-cost hobby servos, which typically have more play and less repeatability than industrial arms, and the policy has to compensate with accurate motion. The second is visual variety. The vials come in three colours that differ in how translucent they are, from lighter tubes that are partly see-through to darker ones that hide what lies behind them, and the scene holds several look-alike vials among which the commanded one has to be found. The third is positional variety. The policy has to reach 8 source slots on two racks and place into any of 7 destinations, giving 46 distinct source and destination pairs in our data, and each of these is requested through a natural-language prompt that is phrased in many different ways. Together these requirements make vial sorting a stringent, practical test of whether a VLA's understanding translates into precise manipulation. They also match a real need, since sorting and moving sample vials is a repetitive laboratory task, and an open-source arm such as the SO-101 makes automating it widely accessible.

Language plays a central role in the third requirement. A command names the exact source and destination, for example *"move the vial from position 3 of the left rack to slot 6 of the right rack"*, and the colour of the vial is deliberately left out, so the policy has to read the positions from the text and find them in the image. The prompts are composed from independent pools of verbs, object nouns, slot phrasings, rack phrasings, and sentence templates, which yields tens of thousands of distinct wordings of a single command (Section 5). The policy therefore has to interpret each command, since no fixed template is available to memorize.

Figure 2 gives an overview of the planned system. A sorting goal is to be broken into atomic source-to-destination commands, such as *"move the vial from position 3 of the left rack to slot 6 of the right rack"*, and a $\pi_{0.5}$ policy executes each command on the SO-101 arm while observing the scene through three RGB cameras. The high-level planner shown in the figure, a VLM orchestrator that would break a sorting goal into such commands, is our plan for future work and is not part of this study. Here the commands are entered by the user, and the study covers the policy that executes them. We fine-tune $\pi_{0.5}$ on 150 teleoperated demonstrations, keeping its vision–language backbone frozen and training only the action expert, and we evaluate the deployed policy in 79 human-scored autonomous rollouts. The evaluation consists of 15 source and destination commands, each run five to seven times on the real robot, and an observer records for every rollout whether the gripper missed the pickup, released the vial away from the commanded slot, collided with a neighbour, or completed the move. Two of the commands start from source slots that never appear in the training data, which tests whether the position grammar extends to unseen positions. This protocol separates errors of understanding, such as going to the wrong vial or slot, from errors of execution, such as a missed grasp or an imprecise release. The policy goes to the wrong vial or slot in only 8%

of these rollouts, so the commands are understood. The physical execution is the demanding part: the initial grasp is missed in 54% of rollouts, the vial is released away from the commanded slot in 19%, and the full pick-and-place completes in 25%. Source slots that were held out of training perform comparably to trained ones, although the sample is too small to support firm conclusions.

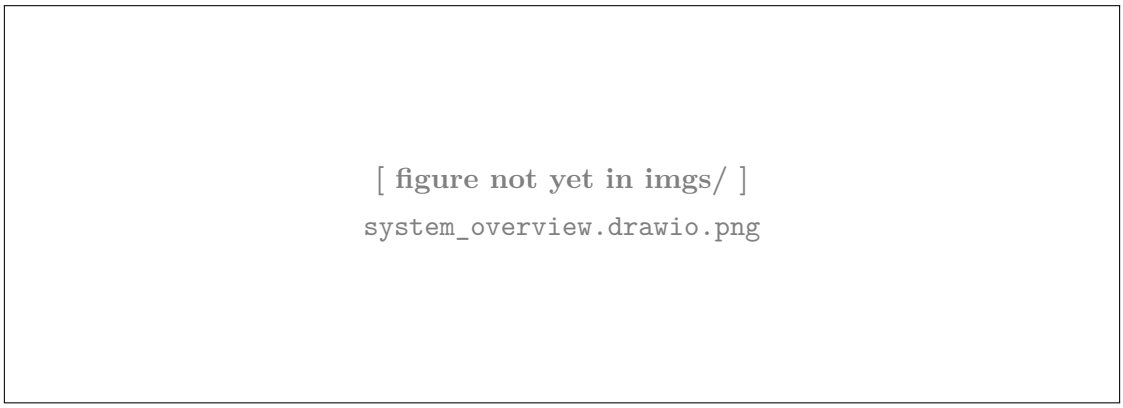


Figure 2: System overview. A high-level planner (a VLM orchestrator) is planned to ground object identity and emit a source-to-destination position command. It is not implemented in this work and is shown as our future plan; here the commands are entered by the user. The low-level VLA policy executes the spatial pick-and-place on the SO-101 arm.

Contributions. This report makes three contributions:

- **A low-cost, language-conditioned vial-sorting system.** An SO-101 rig with three cameras and a client–server control interface that runs a $\pi_{0.5}$ policy on an NVIDIA Jetson Thor, including a deterministic release step that completes discards into the bin.
- **A controlled demonstration dataset and training recipe.** 150 teleoperated demonstrations in which colours and destinations are fully crossed, instructions are paraphrased in a source-to-destination position grammar, and some slots are held out for testing; the dataset is publicly available (Section 5) and is used to fine-tune $\pi_{0.5}$ with a frozen backbone.
- **A quantitative evaluation.** 79 human-scored autonomous rollouts showing that the policy rarely acts on the wrong vial or slot but frequently fails the physical grasp and release, which locates the system's limit on the action side.

The remainder of this report is organised as follows. Section 2 reviews related work, Section 3 describes the hardware setup, and Section 4 the control architecture. Section 5 presents the dataset, Section 6 the policy and its training, and Section 7 the experiments and results. Section 8 discusses limitations and future work, and Section 9 concludes. The SO-101 arm was built and the full rig assembled and calibrated as part of this work; data collection, policy training, and evaluation were carried out collaboratively.

2 Related Work

Vision–language–action models for manipulation. A growing line of work trains a single policy to grasp and move many different objects from language commands. RT-1 [3] learned such a policy from a large set of real-robot demonstrations and could pick up, move, and place a wide

variety of everyday objects on command. RT-2 [4] went a step further by starting from a vision–language model pretrained on web data and fine-tuning it to output robot actions, which let the robot act on objects and instructions it had never seen in its robot data. To pool data across laboratories, the Open X-Embodiment collaboration gathered demonstrations from many different robots into a single dataset [12], on which open generalist policies such as Octo [11] and OpenVLA [8] were trained and released so that others could fine-tune them to their own robots; GR00T N1 [10] pursues a similar open foundation model for humanoid robots. In all of these works, grasping is not programmed separately: the policy learns where and how to close the gripper directly from demonstrations, and RT-1, RT-2, and OpenVLA predict the robot's actions as discrete tokens, one step at a time. These systems are trained on very large robot datasets; our work asks how accurately such a model grasps and places small objects in a precise sorting task.

The π_0 family. The π_0 model [2] combines a pretrained vision–language model with a separate action expert that produces smooth, continuous motions instead of discrete steps. Trained on demonstrations from several different robots, it performs dexterous, multi-stage tasks such as folding laundry and clearing tables, and it can be adapted to new tasks by fine-tuning. $\pi_{0.5}$ [13] keeps this design but is trained on much more varied data—demonstrations from other robots, web data, and high-level descriptions of the task—so that it can carry out tasks in environments it has never seen, such as cleaning up unfamiliar homes. Both models are trained on large and diverse robot datasets; we fine-tune the released $\pi_{0.5}$ base model on 150 demonstrations of a precise sorting task.

Learning from few demonstrations on low-cost hardware. A second line of work asks how much can be learned from a small number of demonstrations on inexpensive hardware. ACT [19] showed that a low-cost pair of arms can learn fine-grained tasks from a few dozen demonstrations by predicting a short chunk of future actions instead of a single step. Diffusion Policy [6] learns the full range of motions seen in the demonstrations, which helps when several different motions are equally valid. SmolVLA [16] brings language conditioning to this setting with a compact model that can be trained on a single GPU and run on consumer hardware, and open-source frameworks such as LeRobot [5] make recording data, training, and deployment on low-cost arms accessible. Our setting—one low-cost arm and 150 demonstrations—belongs to this regime; unlike ACT and Diffusion Policy, our policy follows language commands, and unlike SmolVLA, it builds on a much larger pretrained model.

Hierarchical instruction following. Open-ended requests are followed more reliably when a high-level model breaks them into simple atomic commands that a low-level policy then executes; Hi Robot [15] uses a vision–language model for the first step and a VLA for the second. Our position grammar is designed for such a split: it leaves the choice of *which* vial to move to a separate step—made by the user in this work—while the VLA executes the spatial pick-and-place from a position command.

Grasping transparent and translucent objects. Transparent and translucent objects are a known difficulty for robot grasping because they corrupt the readings of depth sensors. ClearGrasp [14] and TransCG [7] address this by reconstructing the missing depth so that grasps can be planned, and monocular depth estimators such as Depth Anything V2 [17] predict depth from a single colour image. Our vials are translucent coloured plastic, on which the active depth camera is likewise unreliable; we therefore learn grasping from RGB images alone and leave the integration of monocular depth to future work.

3 System Setup and Assembly

Building the arm. The SO-101 is an open, 3D-printed six-DOF design. We printed the structural parts, assembled the six Feetech STS3215 serial-bus servomotors into the linkage, and built a matched pair of arms—a *leader* for teleoperation and a *follower* that executes—wiring each as a daisy-chained serial bus and calibrating every joint's range and zero position.

The manipulator is an SO-101 six-DOF arm operated in a leader–follower configuration for teleoperation: a human moves the leader arm and the follower mirrors it. All computation—camera capture, policy inference, and motor control—runs on an NVIDIA Jetson Thor. Three cameras record at 640×480 and 30 fps: an overhead *top* camera, a *wrist* camera, and a front-facing Intel RealSense D435i *side* camera.

The workspace holds two six-slot tube racks arranged diagonally to the arm, with a bin placed to the left for discards (Figure 7). To remove ambiguity, all spatial references are defined from the side-camera image: the *left* rack is the one on the left of that image (nearest the bin), the *right* rack is on the right, and slots are numbered 1 to 6 from left to right in the image. Assembly included mounting and pinning the three cameras, fixing the rack and bin footprints, calibrating the leader and follower arms, and verifying the serial motor bus. A safety watchdog monitors per-motor temperature and current and cuts torque after repeated limit violations. A wider third-person view of the assembled rig—with the leader and follower arms, the cameras, the racks, and the bin labeled—is shown in Figure 3.

Sensing. The side RealSense additionally provides a metric depth stream, recorded alongside RGB. Because the vials are translucent coloured plastic—the lighter colours more see-through than the darker ones—active-stereo depth is unreliable on them, so the policy is trained from the RGB streams.

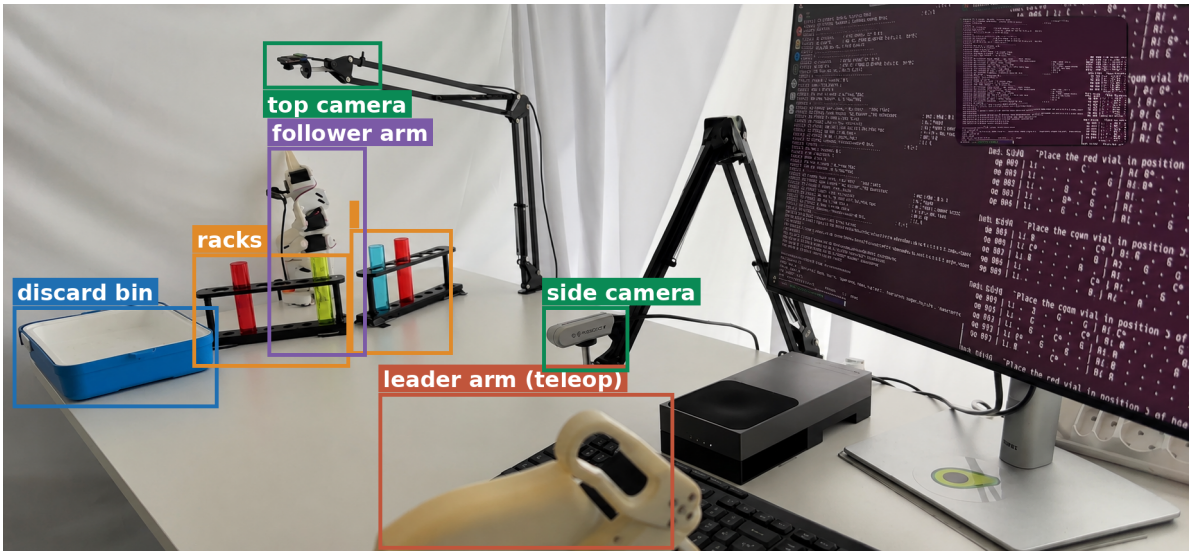


Figure 3: An additional, wider perspective of the rig with the key objects labeled: the teleoperation *leader* arm (foreground), the executing *follower* arm at the racks, the overhead *top* and front-facing *side* cameras, the two tube racks, and the discard bin.

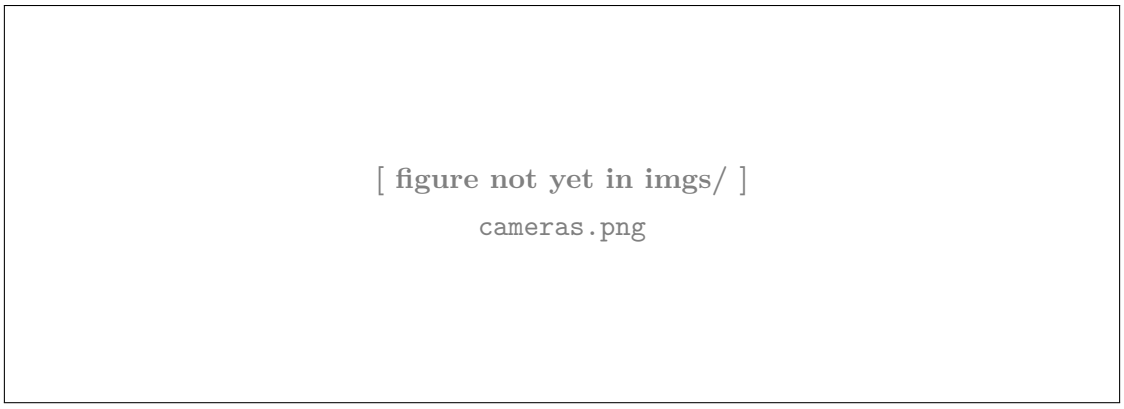


Figure 4: The three camera streams served to the policy (overhead top, wrist, and front-facing side/Intel RealSense D435i); each teleoperated demonstration logs these views together with the synchronized joint trajectory and language instruction—the format the policy is trained on.

4 System Architecture and Control Interface

The system follows a client–server design (Figure 5). The robot, the three cameras, and the policy all reside on the Jetson Thor, wrapped by an inference server (`vla_server.py`) that exposes a small HTTP interface: `/execute` runs a commanded move for a fixed duration, `/home` returns the arm to the training start pose, `/observe` and `/state` stream the live camera frames and status, and `/stop` halts motion immediately. Because every hardware interaction sits behind this interface, any client can drive the robot without touching the hardware directly.

The operator-facing client is a browser control panel (`cv_run.py`) that runs on the workstation and communicates with the Jetson only over HTTP. It shows the three live camera feeds, overlays the policy's motion path on each feed as it moves, and includes a classical computer-vision "slot reader" that detects which vial sits in which slot and pre-builds the corresponding command; Home and Execute controls issue the matching HTTP calls. The inference server additionally serves a lightweight built-in dashboard for direct control. A representative interaction is summarized as a use case in Figure 6: the operator monitors the feeds, optionally analyzes the scene, enters a source-to-destination command, homes the arm, and executes the move, with an emergency stop available at all times.

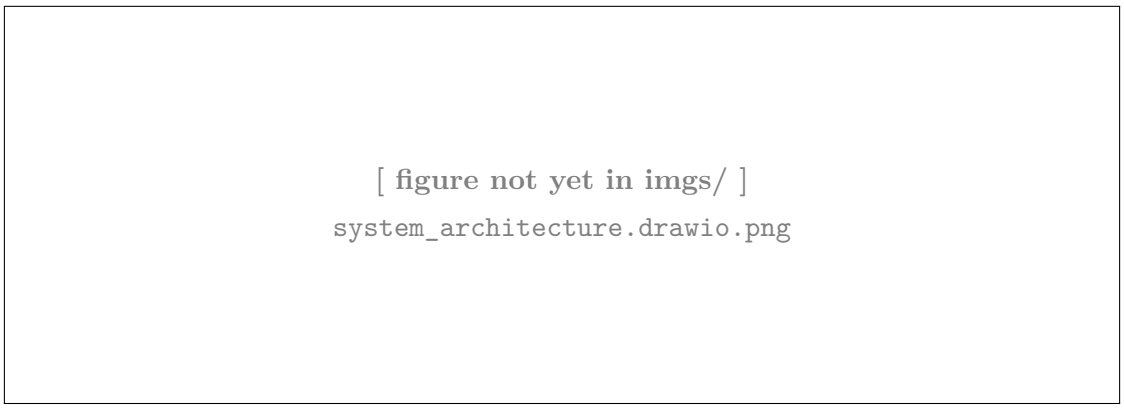


Figure 5: System architecture. The workstation control panel talks to the Jetson-hosted inference server only over HTTP; the server holds the policy, the cameras, and the SO-101 arm.

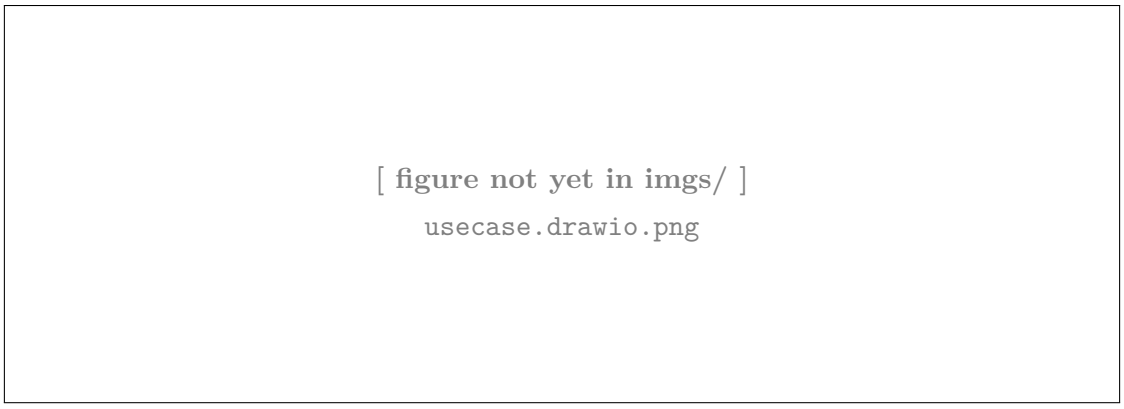


Figure 6: Use-case view of the control panel: monitoring, scene analysis, command entry, homing, execution, and emergency stop.

5 The Vial-Sort Dataset

The core of this work is a teleoperated demonstration dataset recorded with the LeRobot framework [5]. As the operator drives the leader arm and the follower mirrors it, every timestep is logged with the three RGB camera streams (a colorized depth channel is stored alongside), the joint states, and the language instruction. The dataset contains **150 demonstrations**, each a single atomic pick-and-place of roughly 23 s (∼700 frames) at 30 Hz.

5.1 The sorting task

Every demonstration realizes one atomic skill: pick a designated *target* vial from its source slot and place it at a commanded destination—either a slot in one of the two six-slot racks or the discard bin (Figure 7). Targets appear in three colors (red, cyan, dark green), and colors are *fully crossed* with destinations: every color is demonstrated to every destination. Each scene additionally holds two to four *distractor* vials of other colors, so the policy must localize the commanded slot in clutter rather than reach for the only object present.

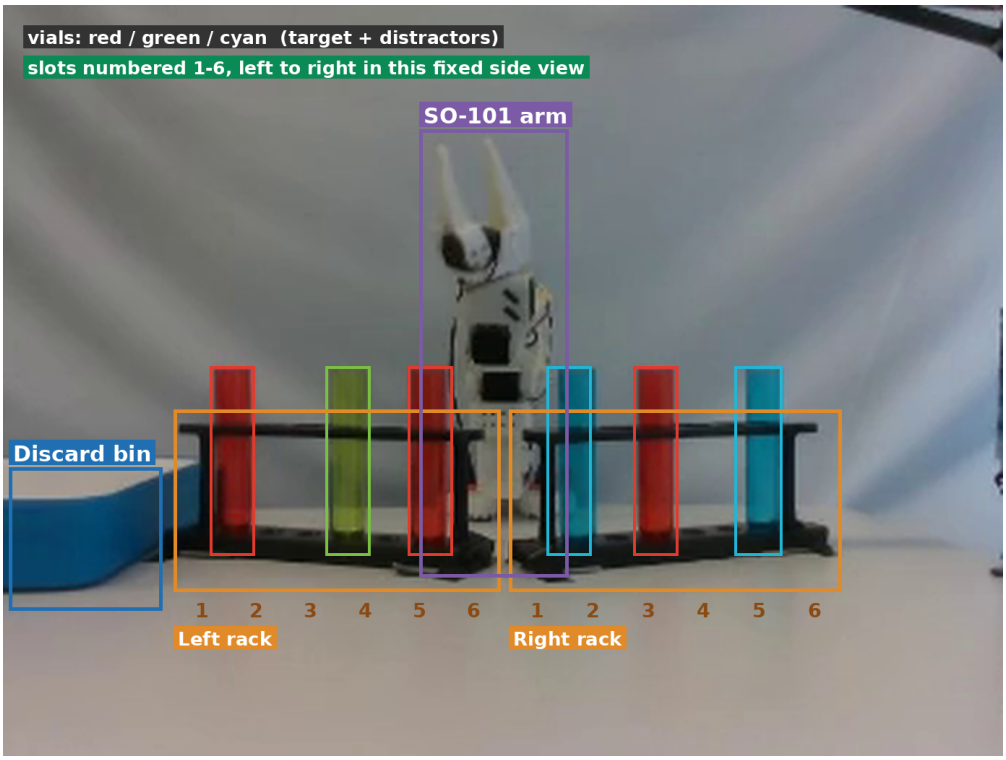


Figure 7: The vial-sorting scene, viewed from the side camera. Two fixed six-slot racks flank the SO-101 arm and the discard bin sits at the far left; vials come in three colors (red, cyan, dark green), one of which is the commanded target while the rest are distractors.

Coverage. The 150 demonstrations cover 46 distinct source→destination pairs—on average 3.3 episodes per pair (range 1–8)—split 120 rack placements to 30 bin discards (Figure 8). The coverage per pair is uneven because the target's source slot varies from scene to scene and the layout constraints below forbid some combinations.

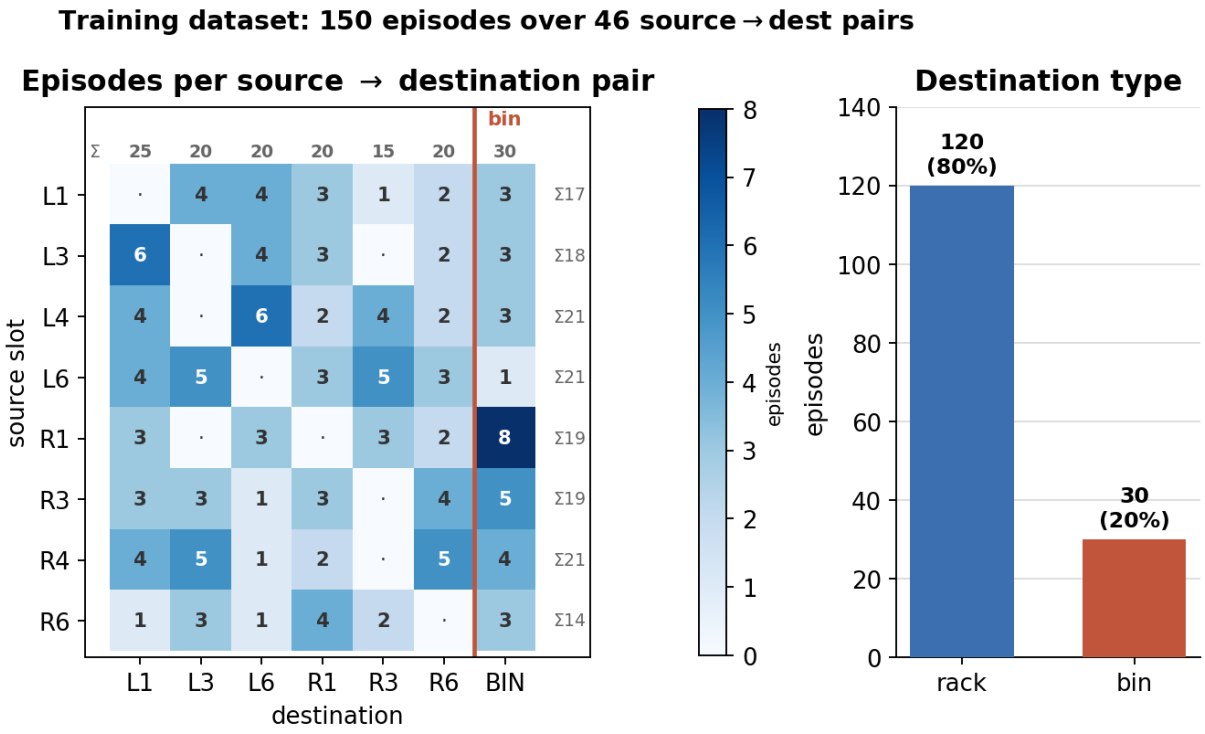


Figure 8: Distribution of the 150 training episodes. *Left:* episode count for every source→destination pair—rows are the eight source slots ($\{1, 3, 4, 6\}$ on each rack), columns the seven destinations, with the bin separated at right; Σ gives each source's total (right margin) and each destination's total (bottom margin). *Right:* the 120 rack versus 30 bin split.

5.1.1 Constraints

Scenes are generated from a fixed, seeded layout sheet under a set of constraints, each chosen for a concrete reason—to keep every episode physically executable, to keep the target unambiguous, and to turn the layout into a controlled test of generalization.

- **Spacing — no two vials in adjacent slots.** The parallel-jaw gripper needs finger clearance on *both* sides of a vial; an immediate neighbor blocks the grasp and the release. Enforcing spacing on the target *and* every distractor keeps every vial pickable and every slot droppable.
- **Destination clearance — empty slot, both neighbors free.** A release into a slot needs room on either side; in practice leaving only *one* neighbor free was not enough, so the destination and both its neighbors are kept empty, and the target never starts adjacent to its destination.
- **Capacity and stock — ≤ 3 vials per rack, ≤ 3 of any color.** Three vials per rack make the scene look realistically full without becoming infeasible to service; the per-color cap matches the physical stock of three vials per color.
- **Color set — three distinct colors, distractors never the target color.** Restricting to three well-separated colors (red, cyan, dark green) yields denser demonstrations per color and unambiguous grounding, and because distractors are never the target's color, the target is the unique vial of its color in the scene.

- **Held-out positions.** Targets originate only from slots $\{1, 3, 4, 6\}$ and trained destinations are only $\{1, 3, 6\}$; slots $\{2, 4, 5\}$ are never used in training and are reserved to test interpolation to unseen positions at evaluation.
- **Full crossing.** Because every color is paired with every destination, no color is correlated with a fixed slot; the *instruction*, not a memorized color-to-slot shortcut, must determine where the vial goes.

Fixed objects are constraints too. The cameras, the two racks, and the bin are all held in fixed positions, and the entire left/right/slot naming convention is read off the fixed side-camera image (Figure 7, whose slot numbering 1–6 is read directly off that fixed view). This is not incidental: the spatial grammar only means something if the viewpoint that defines it does not move.

Why a fixed camera. A fixed camera maps each physical slot to the *same* region of the image in every episode, so the policy can reliably associate "slot n" with a stable visual location. A camera that drifts or is repositioned shifts the whole observation distribution, and a policy trained under one viewpoint degrades under another. We saw this directly: when the top camera drifted from its mounted pose, a π_0 policy that had previously worked stopped working—its inputs had moved outside the distribution it was trained on. We therefore keep the rig rigid and re-anchor each camera to a saved reference frame before recording, treating viewpoint stability as a first-class data constraint.

5.2 Prompt design

Commands use a source-to-destination *position* grammar—e.g. *"Move the vial from position 3 of the left rack to slot 6 of the right rack"* or *"…to the bin"*—and the color of the vial is deliberately never named. Addressing the source slot keeps the instruction unambiguous even when several vials share a color: choosing which vial to move (the color-to-position grounding) is left to a separate step—made by the user in this work—so the policy performs a purely spatial pick-and-place, mirroring the high-level/low-level split of hierarchical VLAs [15].

A single command is realized under a large space of surface forms. Every instruction is composed from independent pools—the verb, the object noun, the phrasing of each slot, the phrasing of each rack, the phrasing of the bin, and the sentence template—listed in Table 1. Multiplying these out gives on the order of $6 \times 4 \times (5 \times 4) \times (5 \times 4) \times 4 \approx 3.8 \times 10^4$ distinct phrasings of a single rack-to-rack command (about 10^4 for the bin variant). The dataset draws from this space deterministically, so consecutive demonstrations of the *same* underlying command are phrased differently; this forces the policy to parse the language rather than latch onto one fixed template. Table 2 shows representative realizations.

Table 1: Prompt-design pools. Each instruction is one draw from each row; the color of each dimension is the key for the colored spans in Table 2.

Dimension	Values
Verb (6)	Move, Take, Transfer, Bring, Relocate, Carry
Object noun (4)	the vial, the tube, the test tube, the sample tube
Slot phrasing (5)	position n, slot n, the nth slot, hole n, spot n
Rack phrasing (4)	the left rack, the rack on the left, the left-side rack, the left tube rack
Bin phrasing (5)	the bin, the trash, the waste bin, the trash can, the rubbish bin
Template (4)	"…from S to D", "…in S to D", "…from S over to D", "…at S to/into D"

Table 2: Example prompts drawn from the pools of Table 1; spans are colored by the dimension they realize (verb, object, slot, rack, bin, template).

Move the vial from position 3 of the left rack to slot 6 of the right rack.
Take the tube in slot 3 of the rack on the left to the sixth slot of the right rack.
Transfer the test tube from the third slot of the left-side rack over to hole 6 of the right tube rack.
Bring the sample tube at spot 1 of the left tube rack to position 3 of the right rack.
Relocate the vial from hole 4 of the right rack to the first slot of the left rack.
Carry the tube from position 6 of the right-side rack to slot 1 of the left rack.
Move the test tube from position 4 of the right rack to the trash can.
Take the sample tube in slot 1 of the left rack into the waste bin.

5.3 Recording and data fields

Representation. Actions and proprioceptive state are expressed in the end-effector frame: recorded joint positions are mapped to a seven-dimensional end-effector pose (position, orientation, gripper) by forward kinematics from the arm's URDF.

Recording. Demonstrations were recorded in short runs against a deterministic, seeded scene sheet; the arm returns to a common home pose before each episode, with a brief reset interval between recordings. Instructions were authored programmatically and expanded into many surface forms per command—varying the verb, the slot phrasing, and the rack phrasing—so the policy sees diverse language for the same underlying move.

Data fields. Following the LeRobot dataset layout, each timestep stores the three RGB images (`top`, `wrist`, `side`) and a colorized depth image, a seven-dimensional end-effector state (`observation.state`), the seven-dimensional end-effector action (`action`), the natural-language task string, and episode/frame indices. Table 3 summarizes the dataset and its scene grammar.

Table 3: Summary of the vial-sort dataset and its scene grammar.

Property	**Value**
Demonstrations	150 (one atomic pick-and-place each)
Frame rate / length	30 Hz, ∼23 s (∼700 frames) per episode
Modalities	3 RGB cameras (top, wrist, side); colorized depth stored
Colors	red, cyan, dark green
Racks / slots	2 racks × 6 slots, plus a discard bin
Destinations	rack slots $\{1, 3, 6\}$ (left/right) + bin, crossed with color
Source slots	$\{1, 3, 4, 6\}$
Held-out slots	source $\{2, 5\}$, destination $\{2, 4, 5\}$ (interpolation test)
Distractors per scene	2–4, spaced, never the target color
Per-rack capacity	≤ 3 vials; ≤ 3 of any color
Command grammar	source → destination, paraphrased
Action / state	7-D end-effector (padded to 32)

5.4 Data and code availability

The teleoperated dataset used to train the deployed policy is released on the Hugging Face Hub in the LeRobot format as `sari-abdan/vial-sort-v5-ee-rgb`,[1] with the RGB streams, the colorized depth channel, seven-dimensional end-effector state and action, and the paraphrased language instruction stored per timestep as described above. Training, deployment, and evaluation were built on the open-source LeRobot framework [5].

6 Policy Learning

The policy is $\pi_{0.5}$ [13], a vision–language–action flow model of roughly three billion parameters in the π_0 family [2], fine-tuned from the released base weights. We train in a *frozen* regime: the vision encoder and the vision–language backbone are held fixed and only the action expert—the flow-matching head that produces motions—is updated.

What $\pi_{0.5}$ is. The structure of $\pi_{0.5}$ is described in the introduction (Figure 1): a pretrained vision–language backbone reads the images, the instruction, and the discretized robot state, and a smaller action expert generates continuous action chunks from the backbone's representation [2, 13]. We rely on the language grounding and visual semantics that the backbone brings from large-scale pretraining, which let a single policy parse the source-to-destination position grammar, while fine-tuning only the action expert adapts the motions to our SO-101 arm.

Training objective (flow matching). The action expert is trained by conditional flow matching [9], the objective used throughout the π_0 family [2]. Let o denote an observation (the three RGB images and the proprioceptive state) and ℓ the language instruction, and let $A = (a_1, \ldots, a_H)$ be the demonstrated action chunk of horizon H=50. We draw a flow time $\tau \sim \mathcal{U}(0, 1)$ and Gaussian noise $\epsilon \sim \mathcal{N}(0, I)$, form the linear interpolant $A^\tau = \tau A + (1 - \tau)\epsilon$, and regress the network's predicted velocity field v_θ onto the target flow $A - \epsilon$:

$$\mathcal{L}(\theta) = \mathbb{E}_{A,\epsilon,\tau,o,\ell} \left\| v_\theta(A^\tau, \tau \mid o, \ell) - (A - \epsilon) \right\|_2^2. \tag{1}$$

[1] `https://huggingface.co/datasets/sari-abdan/vial-sort-v5-ee-rgb`

At inference the learned field is integrated from noise back to a clean action chunk. Only the action-expert parameters θ receive gradients in Equation (1); the VLM weights are frozen, which is what makes the small-data fine-tune both cheap ($\sim$19 GB) and stable.

Why freeze. With only 150 demonstrations, updating billions of vision parameters overfits the recorded scenes and erodes the pretrained language grounding that makes the model follow instructions; freezing preserves those priors and regularizes the small-data regime, the standard transfer-learning outcome for VLAs [4, 8]. Training such a model from scratch is infeasible at this data scale. The choice is empirical as well: an unfrozen variant of the comparison model overfit and failed to grasp, while the frozen policy trained cleanly. Table 4 lists the configuration.

Table 4: Training configuration for the deployed $\pi_{0.5}$ policy (from the checkpoint's `train_config.json`).

Setting	**Value**
Base model	`lerobot/pi05_base`
Trainable parameters	action expert only (vision + VLM frozen)
Objective	conditional flow matching (Eq. 1)
Precision	bfloat16
Optimizer	AdamW (wd = 0.01)
Learning rate	1×10^{-4} peak, cosine decay to 2.5×10^{-6}
Warmup	1,000 steps
Batch size	16
Steps	30,000 (checkpoint every 2,000; deploy 12,000)
Action chunk (H)	50
Action / state space	7-D end-effector (padded to 32)
Image augmentation	enabled
Cameras	RGB (top, wrist, side)
Training dataset	`sari-abdan/vial-sort-v5-ee-rgb`
GPU	1$\times$ NVIDIA RTX PRO 6000 Blackwell (96 GB)

Hardware. Training ran on a single NVIDIA RTX PRO 6000 Blackwell (96 GB). Because only the action expert receives gradients, the frozen regime used roughly 19 GB—far below a full fine-tune—since the frozen majority of the network performs only a forward pass with no gradient, optimizer, or activation memory.

Other policies. Only $\pi_{0.5}$ was carried through to deployment and evaluation in this work. A few other policies were set up on the same platform but were not taken to a deployable, command-following state; they are listed for context in Table 5 and are not evaluated in Section 7.

Table 5: Policies set up on this platform. Only $\pi_{0.5}$ was actually trained and deployed for the general command-following system; the others were exploratory and not carried through.

Policy	Language-cond.	Deployed	Reason not deployed
$\pi_{0.5}$ [13]	yes	**yes**	— (deployed policy)
π_0 [2]	yes	no	only a single fixed command tried, not the full crossed task
GR00T N1.5 [10]	yes	no	exploratory setup; not carried through to deployment
ACT [19]	no	no	no language conditioning; motion baseline only

7 Experiments and Results

Deployment. A chosen checkpoint is served on the Jetson by an inference server that exposes a browser control panel with the three live camera feeds and Home/Execute controls (Figure 9). Before every command the arm homes to the start pose of the demonstrations; we found that a mismatch here produced an out-of-distribution transient (the arm flailed before recovering) and corrected it by setting the home pose directly from the recorded data.

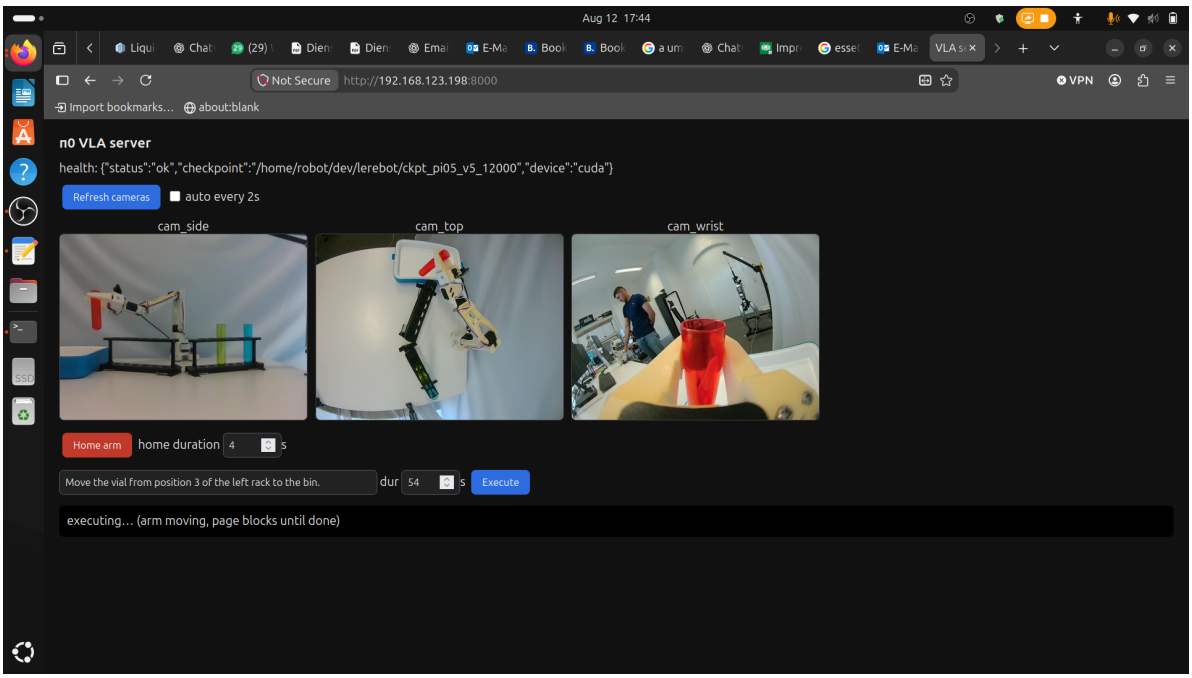


Figure 9: The deployment control panel. The three camera feeds are shown live; a position command is entered and executed on the arm, which first homes to the training start pose.

Checkpoint study. Table 6 summarizes the behavior across training checkpoints. At 8k steps the policy does not yet follow the command (its language grounding is under-trained); at 12k it follows commands, grasps, and completes placements; at 16k it over-fits and loses the grasp. This is the expected data-density/over-fitting curve, and 12k is the operating point we deploy.

Table 6: Checkpoint study for the $\pi_{0.5}$ policy (epochs = steps × batch / 104,421 frames). The 12k checkpoint is the deployed operating point.

Checkpoint	Epochs	Follows command	Grasps	Completes rack placement
8k	≈ 1.2	no	–	–
12k	**≈ 1.8**	**yes**	**yes**	**yes**
16k	≈ 2.5	yes	no	no

Command following. We verify instruction following by holding the scene fixed and changing only the destination clause of the command; the arm then proceeds to different targets accordingly. In the quantitative study below, gross targeting errors—reaching the wrong vial or releasing into the wrong slot—account for only 8% of rollouts, so when the policy grasps the vial it generally carries out the commanded source-to-destination move. Command following is thus *not* the system's bottleneck.

Grasping and placement. Grasping is the limiting factor. Across the evaluation the gripper misses the target pickup on 54% of rollouts (Section 7.1); the two outermost slots (1 and 6) lie at the extremes of the arm's reach and are the hardest to grasp cleanly, and collisions with neighboring vials or the rack occur on 13% of rollouts, concentrated at those extremes. Placement, by contrast, is comparatively reliable: once a grasp succeeds the policy usually lowers the vial into the commanded slot and releases it, completing an autonomous rack-to-rack sort (Figure 10). Some rollouts miss the first grasp and then recover and complete the placement within the time budget.



Figure 10: A complete autonomous rack-to-rack sort under a single position command: approach, grasp at the source slot, transport, and release into the destination slot.

Bin placement. Bin discards are completed by a *deterministic release primitive*. When a command is a discard (its text mentions the bin or trash), the inference server, once the policy move has ended, opens the gripper *in place*—holding every other joint at its measured position and driving only the gripper to its fully-open value—to drop the vial, after which the arm homes to the start pose as it does after every command. The primitive is triggered only for bin commands, so rack-to-rack sorts are unaffected.

Robustness to the environment. The deployed policy tolerates substantial variation in its surroundings. During inference the background was not held constant—people moved through the scene and the surrounding room changed—and the demonstrations themselves were recorded under both lights-on and lights-off conditions. These background and lighting changes did not materially affect the policy's behavior: the arm kept attending to the racks and vials and carried out the commanded move. We attribute this to the frozen, web-pretrained vision–language backbone,

whose features are largely invariant to such nuisance variation, and to the fixed foreground geometry (racks, bin, camera) that anchors the spatial grammar.

7.1 Quantitative evaluation

We evaluate the deployed policy (the 12k checkpoint) on the physical rig by issuing 15 distinct source-to-destination commands—the evaluation *scenarios* S1–S15 (Figure 12)—and running the policy autonomously 5–7 times per scenario, for a total of 79 **rollouts**. A human observer scores each rollout on four labels: *missed pickup* (the gripper failed to grasp the target on its attempt), *missed drop* (the vial was released away from the commanded slot), *collision* (the arm struck another vial or the rack), and *complete success* (the commanded vial was picked and placed at the commanded destination with no error). A rollout that misses the first grasp but recovers and completes the placement within the time budget is counted a success. The commands span both racks and source slots $\{1, 3, 4, 5, 6\}$ with destinations $\{1, 3, 6\}$ and the bin; two commands originate from *held-out* source slots (5 and 2) never seen in training, testing spatial interpolation.

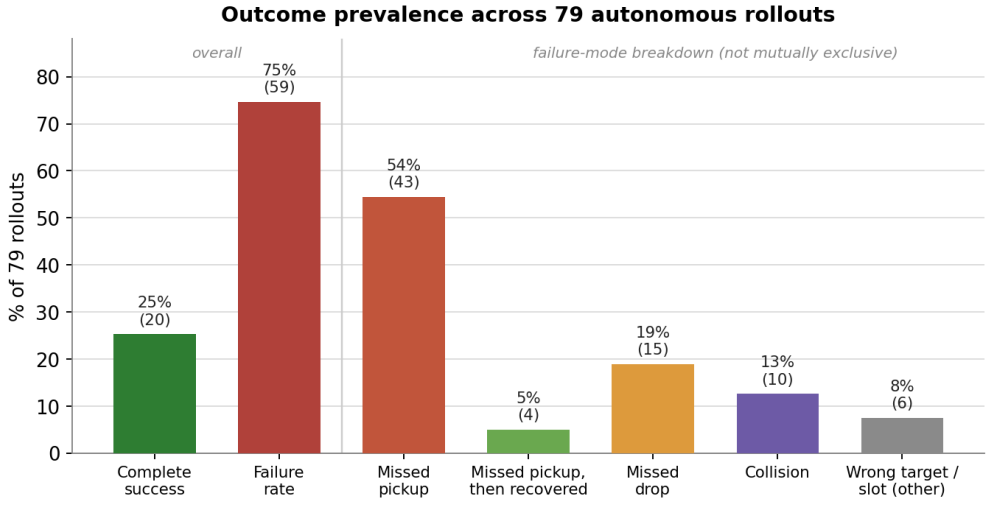


Figure 11: Outcome prevalence across the 79 autonomous rollouts. The first two bars give the overall outcome—complete success (25%) versus failure (75%); the rest break the failures down by mode. A missed pickup occurs on 54% of rollouts, though 4 of those recover and still complete the placement. The failure-mode categories are not mutually exclusive.

Overall outcomes. The policy completes the full pick-and-place on 20 of 79 rollouts (25%), so about three rollouts in four fail (Figure 11). The most common failure by far is the *grasp*: the gripper misses the target pickup on 54% of rollouts. Releasing the vial is the second weak point—it is released away from the commanded slot (missed drop) on 19% of rollouts—whereas a gross targeting error (wrong vial or wrong slot, with no grasp or collision flagged) occurs on only 8%. The policy therefore usually heads for the right vial and the right slot; its failures lie in the physical execution, above all the grasp. Collisions, at 13%, never stand alone: they accompany a missed grasp or drop at the arm's reach extremes. The "missed pickup, then recovered" bar (5%) shows the policy can sometimes re-attempt a missed grasp and still finish within the time budget.

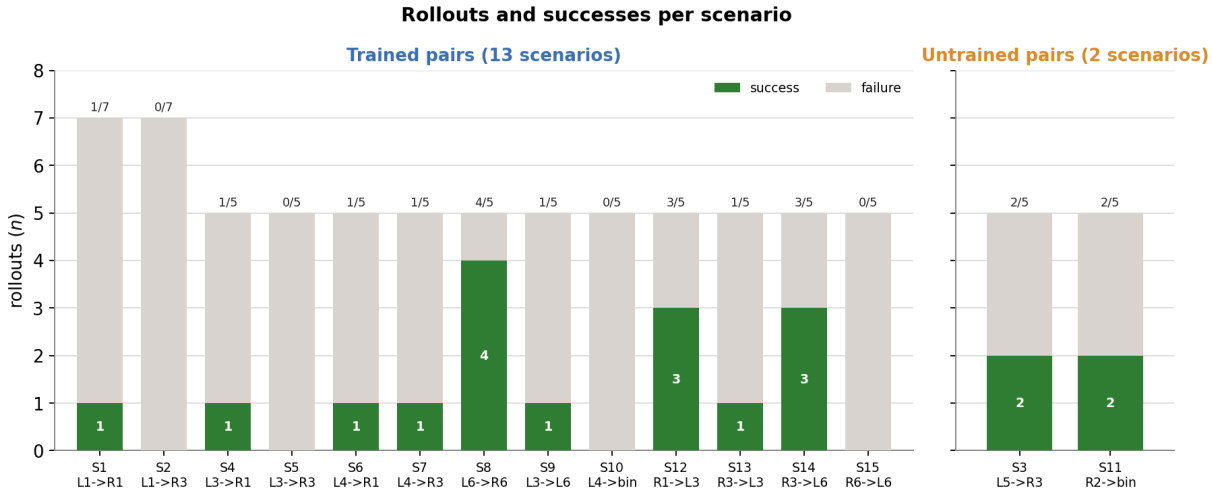


Figure 12: Rollouts per evaluation scenario, split into the 13 trained pairs (left) and the 2 untrained pairs whose source slot (2 or 5) was held out of training (right). Each bar's height is the number of rollouts (5, or 7 for S1–S2); the green portion is the successes, with s/n printed above.

Per-scenario spread. Broken out by scenario (Figure 12), success is very uneven. Each scenario contributes only five rollouts (seven for S1–S2), and outcomes range from no successes on four commands (L1 → R3 at 0/7; L3 → R3, L4 → bin, and R6 → L6 at 0/5) to 4/5 on the best (L6 → R6). Three of the four complete failures come mostly from a missed pickup of the source vial, whereas L1 → R3 fails mainly by releasing the vial off target; the successful scenarios are those whose source slot the arm grasps cleanly. Per-scenario success thus tracks how reliably the arm grasps at the *source* slot—and, for some commands, how precisely it releases—rather than the language.

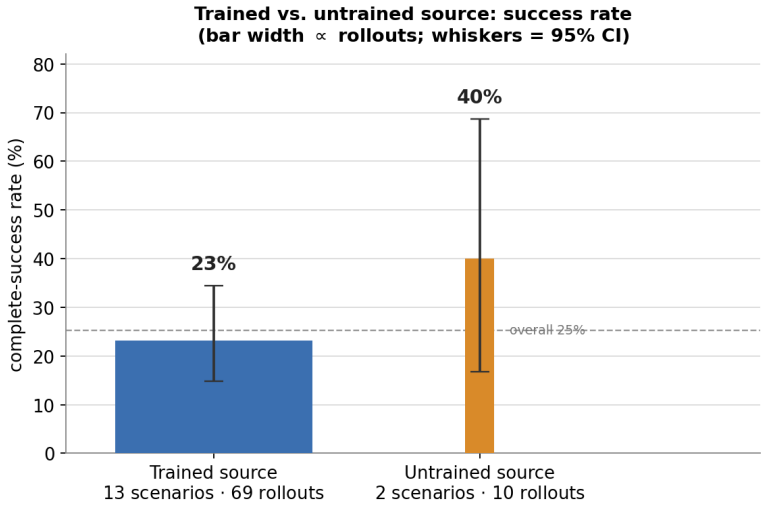


Figure 13: Trained versus untrained source slots. Bar height is the pooled success rate; bar *width* is proportional to the number of rollouts (trained 69, untrained 10) and the whiskers are 95% Wilson confidence intervals. The dashed line marks the overall 25%.

Trained versus untrained source. Pooling the rollouts by whether the source slot appeared in training (Figure 13) gives 23% (16/69) for trained sources and 40% (4/10) for the two untrained ones. The figure encodes the *weight* of each estimate as bar width: the untrained bar is a thin sliver of only ten rollouts, and its confidence interval (17–69%) is very wide and overlaps the trained interval (15–34%). On this sample the apparent untrained advantage is therefore not statistically meaningful; we return to it after the last figure.

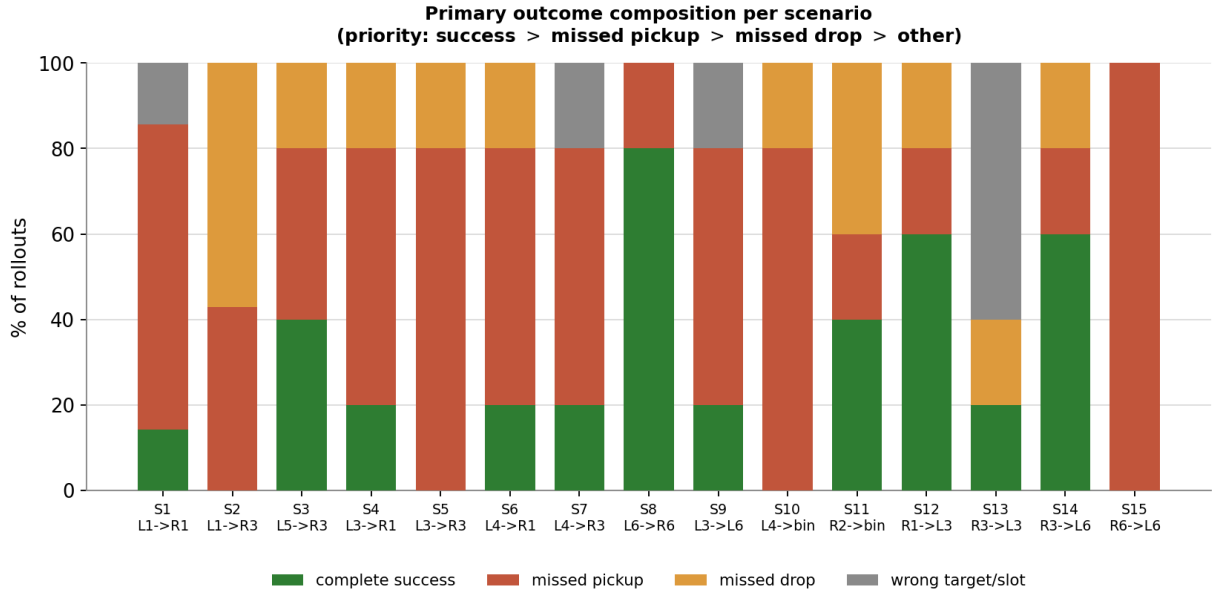


Figure 14: Primary-outcome composition per scenario (each rollout assigned its most salient outcome under the stated priority). Green is complete success; the large terracotta mass is missed pickups. Collision is not shown as it never occurs alone (its overall rate is in Figure 11).

What the composition shows. The stacked view (Figure 14) shows how the failures are distributed across scenarios. A missed pickup (terracotta) is the largest failure in 12 of the 15 scenarios. Missed drops (amber) are more widespread than a single weak stage would suggest: they occur in 10 of the 15 scenarios and are the largest failure in two (L1 → R3 and R2 → bin). The remaining failures (grey), mostly a wrong target or slot, appear in only four scenarios, most visibly R3 → L3, where the vial three times ended in slot 4 of the right rack. Grasping the source vial is therefore the largest single loss, but getting past it is not sufficient: of the 36 rollouts whose first grasp succeeded, only 16 (44%) completed the move, while 14 released the vial away from the commanded slot. Improving grasp reliability would recover the largest share of the failures, and precise release is the next limit behind it.

Trained vs. untrained positions: expected worse, observed better. We expected the two untrained source positions (slots 2 and 5, never seen in training) to be *harder* than trained ones, since the policy had no demonstration of a pick from those slots. Instead they scored higher (40% versus 23%). We do not read this as generalization. The untrained estimate rests on only ten rollouts across two scenarios, with a confidence interval (17–69%) that spans the trained rate, so the gap is within chance. And because per-scenario success is dominated by whether the one source slot is grasped cleanly (some trained scenarios hit 4/5, others 0/5), the two untrained scenarios most plausibly happened to sit at easy-to-grasp positions. Separating luck from a real effect needs

many more rollouts per position; even then, an untrained advantage would most likely mean those slots are simply easier to grasp, not that the policy generalizes better to unseen positions.

8 Limitations and Future Work

Grasp reliability is the dominant limit. A missed initial pickup is the most common outcome (54% of rollouts), and it is worst at the arm's reach extremes (slots 1 and 6). The most direct remedies are (i) more grasp demonstrations per source slot—the per-(source, destination) cell count is thin at 150 demonstrations spread over the crossed set—and (ii) closed-loop grasp correction, e.g. a shorter action horizon or real-time chunking so the policy re-plans on a missed contact instead of committing a full 50-step chunk.

Spatial precision. Gross targeting errors are rare (8%), but releases away from the commanded slot are not (19%), and in one scenario (R3 $\rightarrow$ L3) the vial repeatedly ended in the wrong slot (slot 4 of the right rack). A deterministic, geometry-aware slot verifier read from the fixed side camera could confirm each placement and trigger a retry, complementing the learned policy where it releases imprecisely.

Bin release. The release primitive resolves bin discards at deployment but is an engineered patch outside the learned policy. A small set of consistent controlled-release demonstrations, warm-started onto the 12k checkpoint at a low learning rate, would fold the release into the policy and remove the special case.

Perception. The system is RGB-only because active-stereo depth is unreliable on the translucent plastic vials. Integrating a monocular depth estimator [17] as an additional input channel is a natural next step for improving grasp localization at the reach extremes.

High-level planner. The system overview (Figure 2) includes a high-level planner, a VLM orchestrator that would turn a sorting goal into the atomic source-to-destination commands that the policy executes. It is our planned next step. In this work the commands were entered by the user, so the reported results describe the low-level policy alone.

Evaluation. The reported rates come from 79 human-scored rollouts; the held-out-slot result in particular rests on a small sample. Scaling the evaluation and, potentially, automating scoring with a learned reward model would tighten these estimates and enable a runtime retry-on-failure policy.

Controlled generalization study. The held-out-position result here is confounded by grasp difficulty and a small sample. A cleaner test is to train several policies, each holding out a *different* set of source positions, and evaluate each on its own held-out positions with a large rollout budget. Averaging over many positions and rollouts would separate genuine position generalization from per-slot grasp difficulty and luck; we would then expect the held-out positions to score *lower* on average than the small-sample, likely luck-inflated numbers reported here.

9 Conclusion

We set out to measure how accurately a vision–language–action policy can grasp and place small objects. Sorting small, translucent plastic vials with a low-cost SO-101 arm served as the testbed

because it combines three demanding requirements: an accurate grasp and placement into narrow slots, vials of several colours, and many source and destination positions requested through varied prompts. In this report we have introduced a low-cost, language-conditioned vial-sorting system built around an SO-101 arm and a $\pi_{0.5}$ policy [13, 2]. We have presented a controlled dataset of 150 teleoperated demonstrations and shown that fine-tuning only the policy's action expert on it, with the vision–language backbone frozen, yields a deployable policy that follows source-to-destination commands. Finally, we have shown in 79 human-scored rollouts that the system completes the full pick-and-place in 25% of attempts.

These results answer the question we started from. The commands are followed well: the policy acts on the wrong vial or slot in only 8% of rollouts. The accuracy of the motion has room to grow: the initial grasp is missed in 54% of rollouts, and among the rollouts with a successful first grasp, 44% complete the move. The advantage on held-out source slots rests on too few rollouts to support any claim about generalization. Closing the gap therefore means improving the action side with more grasp demonstrations per source position, closed-loop grasp correction that re-plans after a missed contact, and a controlled, larger study of generalization to unseen positions (Section 8).

Declaration on the Use of AI Tools

In the preparation of this report, an artificial-intelligence large language model (LLM) assistant was used as a writing aid. Its use was limited to refining the language—checking grammar, improving clarity and phrasing—and to assisting with the formatting and preparation of figures and tables. All research contributions—the system design, hardware assembly, data collection, experiments, analysis, and conclusions—are the author's own.

References

[1] Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, et al. Paligemma: A versatile 3b vlm for transfer. *arXiv preprint arXiv:2407.07726*, 2024.

[2] Kevin Black, Noah Brown, Danny Driess, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

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

[4] Anthony Brohan, Noah Brown, Justice Carbajal, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning (CoRL)*, 2023.

[5] Remi Cadene, Simon Alibert, Alexander Soare, et al. Lerobot: State-of-the-art machine learning for real-world robotics in pytorch, 2024. `https://github.com/huggingface/lerobot`.

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

[7] Hongjie Fang, Hao-Shu Fang, Sheng Xu, and Cewu Lu. Transcg: A large-scale real-world dataset for transparent object depth completion and a grasping baseline. *IEEE Robotics and Automation Letters*, 2022. arXiv:2202.08471.

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

[9] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2210.02747.

[10] NVIDIA. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

[11] Octo Model Team et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

[12] Open X-Embodiment Collaboration. Open X-Embodiment: Robotic learning datasets and RT-X models. *arXiv preprint arXiv:2310.08864*, 2023.

[13] Physical Intelligence. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. arXiv:2504.16054.

[14] Shreeyak Sajjan, Matthew Moore, Mike Pan, et al. Cleargrasp: 3d shape estimation of transparent objects for manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[15] Lucy Xiaoyang Shi, Brian Ichter, et al. Hi robot: Open-ended instruction following with hierarchical vision-language-action models. *arXiv preprint arXiv:2502.19417*, 2025.

[16] Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. SmolVLA: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

[17] Lihe Yang, Bingyi Kang, Zilong Huang, et al. Depth anything v2. *arXiv preprint arXiv:2406.09414*, 2024.

[18] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023. arXiv:2303.15343.

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