

CaPtain: A Pipeline for Post-Training Code-as-Policy Robot Agents

Yijia Fan, Xilin He, Yilei Jiang, Dongming Wu, Jiaxu Wang, Xiangyu Yue[†]

MMLab, The Chinese University of Hong Kong NovaXBot

[†]Corresponding author: xyyue@ie.cuhk.edu.hk

CAPTAIN-4B

(every tool is a low-level API, down to IK motion control)

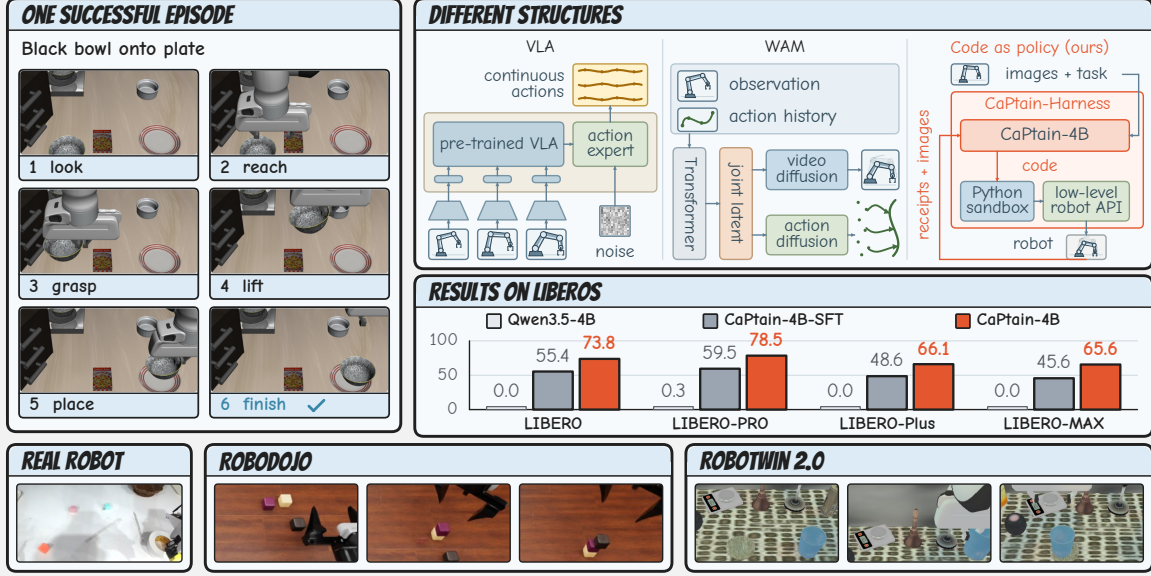


Figure 1 CaPtain-4B, a 4B open model post-trained with our pipeline CaPtain, controls a robot by writing code against low-level primitives. It improves over SFT across LIBERO and three perturbation suites, reaches 78.5 on LIBERO-PRO, and transfers zero-shot.

[Project Page](#) · [Code](#) · [Models & Data](#) · [Video](#)

Abstract

Across coding, web, and tool-use tasks, language-model agents are now routinely trained with the same recipe: supervised fine-tuning (SFT) to learn the interface, then multi-turn reinforcement learning (RL) to master it. Code-as-policy (CaP) puts robot manipulation in the same form, as the policy writes code against robot APIs and reads back the results, so the same recipe should apply. Yet, existing CaP agents sidestep training: they either prompt closed frontier models or delegate manipulation to pre-trained vision-language-action models (VLAs) and high-level skills, while small open vision-language models (VLMs) fail outright. We present CaPtain, an open post-training pipeline that brings this recipe to CaP and turns a small open VLM into a capable CaP agent, with no VLA, frontier model, or privileged state at test time. CaPtain trains and deploys the policy inside CaPtain-Harness, which exposes low-level perception, geometry, and guarded-motion primitives across simulation and real hardware. Within this harness, we first align the base model via SFT on synthetic trajectories generated with GPT-6 Astra, and then apply multi-turn RL. Multi-turn RL, however, must credit tens of requests from the agent, whose physical effects often surface several turns later, with a single outcome observed only at the end of the episode. Since training runs in simulation, we introduce a physical critic that reads the privileged physical state in simulator, which the policy never sees, and credits each request by how much it changes the predicted chance of success. Starting from Qwen3.5-4B (0% zero-shot), CaPtain-4B achieves 55.4% held-out LIBERO success after SFT and 73.8% after RL. On LIBERO-PRO, it reaches 78.5%, surpassing several frontier-model agents, and its RL gains persist across three perturbation suites (+17.6 to +21.2 points). Without target-domain training, it attains 83.3% on SimplerEnv WidowX, exceeds frontier models on CaP-Bench, and runs on a real robot after simulation-only adaptation. We release all code, data, and model weights.

1 Introduction

Language-model agents have improved rapidly in coding, web navigation, and tool use, and much of this progress follows a common recipe: the agent acts inside a fixed execution harness, supervised fine-tuning (SFT) teaches it the interface, and multi-turn reinforcement learning (RL) refines its behavior from the outcomes of its own interactions [19, 25, 26, 72]. Code-as-policy (CaP) agents [27, 43, 58] give robot manipulation the same form: the policy observes camera images, writes programs against perception and motion APIs, and reads back what happened. Since the program is composed at test time, a CaP agent inherits much of the generalization of the language model that writes it, and frontier models have made this approach strikingly capable. Used directly as a robot policy, GPT-6 Astra ranks above every public entry on the RoboDojo leaderboard [90], and with a coding harness it solves 94.0% of LIBERO-PRO episodes [67]. This raises a natural question: *Can a small open model be trained into such a CaP agent with the same SFT-and-RL recipe, rather than relying on a frontier model at test time?*

In practice, existing CaP agents sidestep training, and in doing so hand the manipulation itself to components the agent cannot learn to improve. One line of work prompts closed frontier models without weight updates [13, 39, 40, 50, 93]. These systems come with high finical cost of calling frontier models in each episode [13, 39, 93], cannot be fine-tuned for new robots or tasks, and cannot run on onboard compute. Another line delegates manipulation to pre-trained vision-language-action models (VLAs) or high-level skills. PyRUA trains a 4B coder over a frozen VLA and a motion planner [67], AgenticRoboRL trains a planner around a frozen LIBERO-trained VLA [48], SenseNova-RoboRSI builds a 27B agent around reusable skills and optional VLA calls [64], and Gemini 4 Argon exposes code-as-policies through skills such as grasp and place [22]. In these systems, the code decides *which* skill to call, but the skill decides *how* to grasp, approach, and place. Therefore, the agent stays bounded by the data on which the skill was trained, and RL can at best learn to sequence it. Small open vision-language models (VLMs), meanwhile, fail outright when asked to control a robot directly. Qwen3.5-4B solves none of the LIBERO episodes in our harness, and the few efforts that train such models rely mainly on SFT [45] or single-turn text-only RL [20]. What is missing is a complete SFT-and-RL recipe that teaches a small open VLM to manipulate through low-level primitives, with every manipulation decision made by the policy itself.

Applying the recipe at this low level is harder than at the skill level, for a reason that does not arise for language agents. Without semantic skills such as grasp and place, the policy must itself decide where to grasp, from which direction to approach, and when to close the gripper, and these decisions are spread over tens of requests per episode, most of which only perceive or verify. Multi-turn RL must then credit these requests, whose physical effects often surface several turns later, with a single outcome observed only at the end of the episode. Outcome-only RL assigns the same credit to every request, while a language-model critic would have to infer physical progress indirectly from text and images. Robot learning, however, offers an asymmetry that language agents lack: training runs in simulation, where exact object poses, grasp states, and sub-goal progress are known, even though the deployed policy must act from its cameras alone. A critic can thus read the physical state that the policy cannot see, and credit each request by how much it moves the task toward success, while the policy itself still acts from its cameras at test time.

We present **CaPtain**, an open post-training pipeline built on these two observations. CaPtain trains and deploys the policy inside **CaPtain-Harness**, an execution environment that exposes 22 low-level perception, geometric measurement, and guarded-motion primitives and nothing higher. There are no grasp, place, or open-drawer skills and no VLA. Perception returns masks and points, and motion takes an explicit end-effector pose or gripper command. Every call returns a structured text receipt reporting what physically happened, such as measured poses, contact forces, or why a motion stopped, and the policy-facing API is unchanged across LIBERO [44], RoboTwin 2.0 [11], RoboDojo [10], SimplerEnv [38], CaP-Bench [20], and a physical robot. We first apply SFT on in-harness trajectories generated with GPT-6 Astra, rendered in exactly the multi-turn format the policy sees at deployment. We then run multi-turn RL with a simulator critic that reads privileged physical state hidden from the policy and credits each request by how much it changes the predicted chance of success, combining the streaming RL formulation of SAO [25] with asymmetric actor-critic training [56]. Since this change is measured at a fixed time index, requests that only perceive or verify receive near-zero credit, and credit concentrates on grasps and placements. The critic is used only during training. At test time, the policy relies on camera images and receipts alone.

Applying CaPtain to Qwen3.5-4B yields **CaPtain-4B**, which raises the held-out LIBERO success from 0% to 55.4% after SFT and to 73.8% after RL. On LIBERO-PRO, CaPtain-4B reaches 78.5%, above frontier-model agents such as K1 [40] (77.2%), WAA [93] (75.6%), and ASPIRE [50] (71.7%). RL also gains more when it acts on low-level primitives: across the four LIBERO-PRO suites it adds 19.4 points to our SFT model, whereas RL over a frozen VLA adds 5.9 points in AgenticRoboRL [48], although the two start from different baselines. These gains persist under perturbation, ranging from +17.6 to +21.2 points across LIBERO-PRO [97], LIBERO-Plus [18], and LIBERO-MAX [95], with all seven LIBERO-Plus perturbation dimensions improving significantly; by contrast, reported results for end-to-end policies show $\pi_{0.5}$ [55] falling from 97% on LIBERO to 12.8% on LIBERO-PRO [97] and the RL-tuned RIPT-VLA [68] scoring no higher than its base model on LIBERO-Plus [18]. Without target-domain training, CaPtain-4B reaches 83.3% on SimplerEnv WidowX, scores 75.0 on CaP-Bench, above frontier models and a 7B coder trained with RL on that benchmark, and completes tasks on a physical robot after adaptation in a coarse simulated twin alone. Transfer remains limited on RoboTwin 2.0 and RoboDojo (22.6% and 0.8% success), which require dual-arm coordination and unfamiliar arm kinematics. Together, these results suggest that robot manipulation over a low-level execution harness can be post-trained like any other language agent.

Contributions.

1. **CaPtain**, an open, end-to-end SFT-and-RL pipeline that turns a small open VLM into a standalone CaP agent. We release the SFT data, the training code and the CaPtain-4B weights.
2. **CaPtain-Harness**, a portable execution harness with low-level primitives only, exposing 22 perception, measurement, and guarded-motion capabilities with grounded text receipts.
3. **A simulator critic** for multi-turn VLM agents that converts privileged physical state transitions into dense per-request credit, with analyses showing that credit concentrates on physical manipulation decisions.
4. **A comprehensive evaluation** showing that RL gains persist across three perturbation suites, that a 4B policy surpasses several frontier-model agents at a small fraction of their cost, and that it transfers zero-shot across benchmarks and embodiments and to a physical robot without real-world data.

2 Related Work

Code-as-policy robot agents. Code as Policies [43] and VoxPoser [27] showed that a language model can control a robot by writing programs over perception and control APIs. Recent agents scale this idea with frontier models and richer harnesses. GPT-6 Astra, used directly as a robot policy, ranks above every public entry on RoboDojo [90], and Gemini 4 Argon exposes code-as-policies as a developer workflow built around skills such as **grasp** and **place** [22]. Many systems reach their headline results with frontier models and no weight updates, including ASPIRE [50], RATs [88], RHO [16], RACaP [39], OpenRUA [13], RoboDawn [24], Encore [28] and RoboRSI [77]; OpenRUA, for example, reaches 91.8 on LIBERO-PRO at \$8–22 per episode. Others orchestrate a frozen VLA as a skill, including Harness VLA [94], PyRUA [67], Pigey [21] and AgenticRoboRL [48, 81], or give the agent privileged simulator state or a success oracle at test time [40, 69, 93]. Training the code-writing model itself has received less attention. CaP-X [20] applies GRPO to a single-turn, text-only 7B coder on three Robosuite tasks. Guava [45] trains Qwen3.5-4B mainly by SFT and adds GRPO on two MuJoCo tasks. PyRUA trains a 4B coder with GRPO on top of VLA and motion-planning primitives [67], SenseNova-RoboRSI releases a 27B agent that improves itself through reusable skills [64], and AgenticRoboRL trains a 9B planner by SFT and multi-turn RL around a VLA trained on LIBERO [48]. CaPtain instead post-trains a small open VLM end-to-end directly over low-level perception and motion primitives, operating at test time from camera observations and execution receipts alone.

End-to-end robot policies. Vision-language-action models (VLAs) such as OpenVLA [30], OpenVLA-OFT [31], π_0 [2], $\pi_{0.5}$ [55], GR00T [52, 53] and RDT [47] learn from large demonstration corpora and reach 94–98% on standard LIBERO. World-action models (WAMs) build on video-generation pretraining and predict future observations together with actions, e.g. OpenWAM- α [76], the composable OpenWAM framework of Yu et al. [83], and others [73, 79]. A growing line of work post-trains VLAs with RL [9, 36, 46, 49, 68, 85], which raises in-distribution success. Perturbation suites such as LIBERO-PRO, LIBERO-Plus and LIBERO-MAX [18, 95, 97] test whether these abilities generalize, and they show that end-to-end policies remain

TOP: INFERENCE IN CAPTAIN-HARNESS • BOTTOM: TRAINING AND TEST

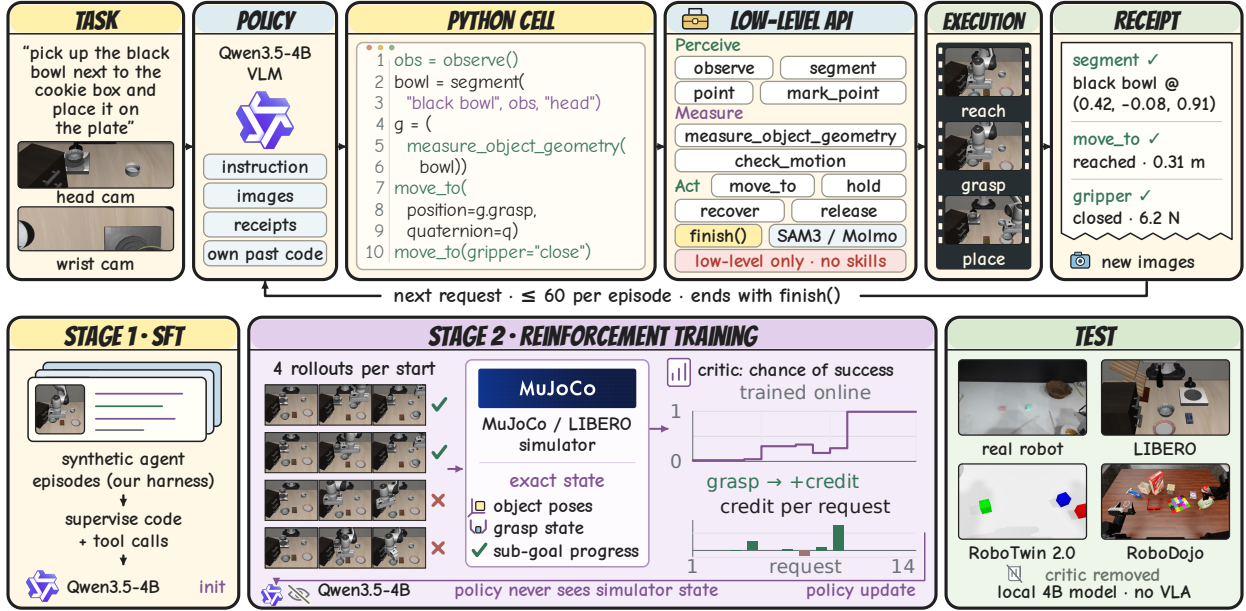


Figure 2 Overview of CaPtain. *Top:* inference, one episode in CaPtain-Harness: at each request the policy writes a Python cell against a low-level API and receives text receipts and new images, for up to 60 requests. *Bottom:* training and test: SFT on synthetic agent trajectories, then RL in which a simulator critic turns privileged physical state into per-request credit; the critic is removed at test time.

sensitive to changes in objects, layouts and instructions. For example, $\pi_{0.5}$ drops from 97% on standard LIBERO to 12.8% under the LIBERO-PRO position and task perturbations [97], and on LIBERO-Plus, RL post-training with RIPT-VLA scores 68.4 against 69.6 for its base OpenVLA-OFT [18, 68]. We study the same question for a policy that writes code, and find that its RL gains persist across all three suites (§4.1).

Credit assignment and privileged critics. PPO [63] with GAE [62] and critic-free GRPO [65] are the standard tools for LLM RL. For long multi-turn episodes, turn-level credit [3, 8, 19, 37, 42, 71, 74, 78, 87, 91, 98] and value estimation [29] are active topics, and SAO [25] shows that streaming RL with a co-trained language-model critic scales to frontier LLM agents. The works closest to our credit mechanism turn a state-value or progress difference into turn-level credit, using an exact game solver (CAST [75]), hand-written acceptance checks (ProCredit [51]), a frozen distance estimator (T2SPO [86]) or rule-based predicates over simulator state (DLS [12]). Privileged critics for LLMs appear in SWEET-RL [99], Le Critique [70], KAT-Coder [26] and Odysseus [66], but none of them reads physical state. In robotics, asymmetric actor-critic [56] gives the critic full simulator state while the actor sees images, with analyses of its bias under partial observability [1, 15], privileged teachers are distilled into sensorimotor students [7, 33, 35], and privileged critics have been used with non-language actors on LIBERO [89, 92]. CaPtain carries the asymmetric template to a language-model actor that writes code: since continuous manipulation offers neither a solver nor a checklist, it learns a value over privileged physics and turns its changes into per-request credit.

3 CaPtain

CaPtain converts a small open vision-language model into a code-as-policy agent by coupling a fixed, portable execution harness (§3.1) with in-harness supervised fine-tuning on synthetic trajectories (§3.2) and multi-turn reinforcement learning guided by a simulator critic (§3.3), as illustrated in Figure 2. The harness defines the low-level action space and receipt feedback loop, SFT aligns the model to this interaction protocol, and RL optimizes closed-loop manipulation and recovery within the same environment. Hyperparameters and infrastructure details are provided in Appendix A.

3.1 CaPtain-Harness

A ReAct loop over requests. A task is given as a natural-language instruction and a `TASK.md` file that describes the API, the coordinate frames and the standard by which the final scene is judged. The agent then runs a ReAct-style loop [82] in which actions are code. An episode is a sequence of *requests* $t = 1, \dots, T$ with $T \leq 60$. At request t the policy π_θ , a vision-language model, observes the history:

$$h_t = (\ell, o_1, a_1, o_2, \dots, a_{t-1}, o_t), \quad (1)$$

where ℓ is the task description with the initial head and wrist images, a_k is the output of request k , and o_k is what the harness returned. Each request is a single model call. The policy first reasons in its thinking channel and then emits one or more calls to the `python` tool, so $a_t \sim \pi_\theta(\cdot | h_t)$ contains both the reasoning and the code. The harness executes the calls in order, appends their outputs (cell results, receipts and images) as o_{t+1} , and issues the next request with the updated history. The loop ends when a reply contains no tool call, normally right after `finish()`, or when a budget on requests, physical actions or wall time is exhausted. The overall design of CaPtain-Harness is summarized in Figure 7 (Appendix A).

Context management. A long episode does not fit in the context window of a small model, so the harness manages the history explicitly. Tool outputs are truncated to a fixed token budget, at most 20 receipt lines are printed per cell, and only the 12 most recent images are kept. When the history approaches the context limit, the harness compacts it turn by turn. It keeps the initial task context and the most recent turns verbatim, and replaces the removed turns with a short notice that lists the commands they ran and warns that the scene may have changed since. Because the agent’s variables live in the Python session rather than in the prompt, detections, poses and helper functions survive compaction. Compaction behaves identically during RL rollouts and at test time, so the policy is trained on the inputs it will see.

A persistent Python session. The only tool is `python`, a sandboxed Python session with a pre-imported `robot` module. Variables persist across requests, so the agent can store detections, poses and its own helper functions and reuse them later. Robot calls do not raise exceptions; each returns a structured result with a success flag, a failure category, a message and a value. This lets the agent write ordinary control flow (if, loops, retries) around robot calls within a single request.

Low-level primitives only. The `robot` module exposes 22 capabilities plus `finish()` (Appendix B):

- **perception:** `observe` (head and wrist RGB-D), `segment` (text-prompted masks from SAM 3 [6]), `point` (pointing with Molmo2 [14]), `mark_point` (back-projects a pixel the model chose itself) and `track_target`;
- **measurement:** `measure_object_geometry` (center, extents, yaw and the offset to a destination), contact and release geometry checks, and comparisons of an object across observations;
- **motion:** `move_to` (an absolute end-effector pose, or a single jaw command), `hold`, `recover` (a bounded reversal along the last trajectory) and `release`;
- **planning and state:** read-only reachability and timing checks, robot state and pose arithmetic.

There are no grasp, place or open-drawer skills and no VLA. The agent must decide where to grasp, from which direction to approach and when to close the jaw. The harness turns each `move_to` into a straight Cartesian motion, solves inverse kinematics, checks self-collision and monitors force at every control substep, stopping at first contact unless the agent requests a manipulation mode that tolerates load. Because every approach vector, grasp pose, and jaw command is written explicitly in Python, the full manipulation policy is exposed to language-model post-training.

Receipts. After each robot call, the harness appends a short *receipt* to the cell output that reports what physically happened: for motions, the stop reason, the executed distance, the peak force and the jaw opening before and after; for perception, the detection identity, its metric position and its offset to the destination. The agent can also attach up to four camera images per cell. Receipts give the policy dense, grounded feedback in text that complements the images.

Self-verification and success. A completion gate on the host checks the agent’s own recorded measurements and may reject `finish()` twice before accepting it. *Host success* is the benchmark’s goal predicate, evaluated by the environment host on the final state. *Strict success* additionally requires that the agent ended the episode with `finish()`, since a robot that does not declare completion cannot be deployed.

Portability. The policy-facing API is unchanged across LIBERO, RoboTwin 2.0, RoboDojo, SimplerEnv, CaP-Bench and the real robot; backend adapters, some of which add benchmark-specific safeguards, are listed in Appendix L.

3.2 Stage 1: SFT in the harness

Format alignment. Small open models fail zero-shot in CaPtain-Harness, so we start with supervised fine-tuning on synthetic agent trajectories collected in the harness itself (data generation is described in Appendix C). Each trajectory is rendered with the same chat template, system prompt, `TASK.md`, tool schema and receipt format that the policy later sees, so SFT examples, RL rollouts and test-time requests share one input format. We start from the released instruction-tuned Qwen3.5-4B [59].

Objective. Given a trajectory $\tau = (h_1, a_1, \dots, h_T, a_T)$, we supervise only the tokens the agent produced:

$$\mathcal{L}_{\text{SFT}}(\theta) = - \sum_{t=1}^T w_t \sum_{j=1}^{|a_t|} \log \pi_{\theta}(a_{t,j} \mid h_t, a_{t,<j}), \quad (2)$$

where $w_t = 0$ masks out turns with repeated execution failures or recovery loops and $w_t = 1$ otherwise, and visual question-answering examples are added with weight 0.5. Each a_t contains the agent’s reasoning as well as its code, so the model learns to plan, check receipts and decide on the next call, not only to emit code. Harness outputs (cell results, receipts and images) are context only.

Forcing the model to look. Because many episodes share similar layouts, a model can learn to copy pixel coordinates from earlier `mark_point` calls instead of reading the image. During SFT we therefore replace earlier coordinates in the history with a mask token with probability 0.2, which forces the model to ground new calls in the current image. We fine-tune all language-model weights and keep the vision tower frozen.

3.3 Stage 2: RL with the simulator critic

RL maximizes the expected return $J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau) + \sum_t c_t]$, where R is a terminal return and $c_t \leq 0$ are small per-request costs. The difficulty is assigning this sparse, delayed signal to the requests that caused it.

Privileged state. Before each request, the simulator records a hidden snapshot s_t that the policy cannot access. From it we compute 36 features $x(s_t)$ in five groups: end-effector and jaw state; goal bookkeeping (number of goals, fraction satisfied, active goal); distances between end-effector, target and destination; target lift and displacement; and articulated-joint positions together with held and ever-held flags (Appendix A). We deliberately exclude counters of the policy’s own behavior, such as the number of perception or failed calls, which otherwise act as proxies for elapsed time and absorb credit from grasping (§4.5).

The simulator critic. The critic predicts the probability of host success from the physical state, the request index, and the task:

$$V_{\phi}(s_t, t) = \sigma \left(\frac{1}{5} \sum_{k=1}^5 f_{\phi_k}([x(s_t); t; e_{\text{task}}]) \right), \quad (3)$$

an ensemble of five small MLPs whose logits are averaged. It is warm-started on episodes of an earlier policy and then *co-trained* online, as in SAO [25]: after each round of rollouts it takes a few gradient steps on the binary cross-entropy loss:

$$\mathcal{L}_V(\phi) = - \sum_i \sum_t \left[y_i \log V_{\phi}(s_{i,t}, t) + (1 - y_i) \log (1 - V_{\phi}(s_{i,t}, t)) \right], \quad (4)$$

where $y_i \in \{0, 1\}$ is the host success of episode i . Co-training keeps the value calibrated to the improving policy. The advantages of a round always use the critic from before that round’s update.

Per-request credit. For episode i we define the temporal-difference credit of each request as the change in predicted success it causes, plus its immediate cost:

$$\delta_t = c_t + V_\phi(s_{t+1}, t) - V_\phi(s_t, t) \quad (t < T), \quad \delta_T = c_T + R_i - V_\phi(s_T, T). \quad (5)$$

Both values in each difference use the same index t , so the passage of time alone produces no credit. Because physical state changes only through physical actions, $\delta_t \approx 0$ for requests that only perceive or verify, regardless of how many tokens they contain. Such requests still receive credit through generalized advantage estimation [62], $A_t^{\text{GAE}} = \sum_{k \geq 0} \lambda^k \delta_{t+k}$ with $\lambda = 0.9$, from the physical actions they enable. Each start state is rolled out $G=4$ times, and we add a group-relative outcome term:

$$A_{i,t} = (G_i - \overline{G}_{\text{group}}) + (A_{i,t}^{\text{GAE}} - \overline{A_{i,\cdot}^{\text{GAE}}}), \quad G_i = R_i + \sum_t c_{i,t}, \quad (6)$$

which ranks rollouts from the same start, while the critic term ranks requests within a rollout. Advantages are divided by a running estimate of their standard deviation, and all tokens of request t , including its reasoning, share $A_{i,t}$. A request whose cells only produced Python errors receives no positive credit, and its tokens receive a small additional penalty.

Returns and costs. The terminal return separates the outcomes that matter for deployment:

$$R_i = \begin{cases} 1 & \text{success with } \texttt{finish}(), \\ 0.25 & \text{success without } \texttt{finish}(), \\ 0 & \text{other failure,} \\ -0.1 & \text{budget exhausted,} \\ -0.3 & \text{false finish (} \texttt{finish}() \text{ on a failed state).} \end{cases} \quad (7)$$

The costs c_t penalize failed or rejected physical actions, physical actions well beyond the typical count for the task, and any actions or requests after the scene is already solved, which pushes the policy to declare completion. If all goals have held for 15 consecutive requests without `finish()`, the episode is stopped and settled as a missed finish. For the hardest long-horizon tasks, a failed episode also receives a small one-shot reward for each sub-goal it reached (grasp, lift, place, joint opened), bounded below the return of a missed finish (Appendix E).

Policy update. We follow the streaming, single-pass update of SAO [25] with double-sided token masking. For token j of request t , let $r_{t,j} = \pi_\theta(a_{t,j} \mid \cdot) / \pi_{\text{old}}(a_{t,j} \mid \cdot)$ be the ratio to the rollout policy. The policy loss is given by:

$$\mathcal{L}_\pi(\theta) = - \sum_{i,t,j} m_{t,j} r_{t,j} A_{i,t} + \beta \text{KL}_{k3}(\pi_\theta \parallel \pi_{\text{SFT}}), \quad m_{t,j} = \mathbf{1}[0.2 < r_{t,j} < 4], \quad (8)$$

so tokens whose ratio has drifted too far in either direction are dropped rather than clipped, and the KL term keeps the policy close to the SFT model. Rollouts are generated asynchronously while the trainer updates, so a request may come from a policy up to two versions old; the ratio mask bounds the effect of this staleness. To bound the cost of long episodes, at most eight requests per trajectory are used for the policy gradient, sampled uniformly and reweighted by their inclusion probability; the critic still uses every request. The gradient-noise scale of this estimator is in the thousands of episodes (Appendix H), so we accumulate gradients over 1,024 episodes per optimizer step; without accumulation, consecutive updates were nearly orthogonal.

Training distribution. RL samples fresh start states across all 40 LIBERO tasks, mostly wide-layout scenes with randomized lighting, camera pose and materials, plus official initial states. The 240 official states used for evaluation are excluded from sampling. To save rollout time, an episode is stopped early once the critic predicts near-certain failure for several consecutive requests, except in a small control slice. Algorithm 1 summarizes one RL round.

Algorithm 1 One RL round of CaPtain

- 1: Sample 64 start states; roll out each $G=4$ times with π_θ in CaPtain-Harness, recording the hidden snapshot s_t before every request
 - 2: Compute per-request credit δ_t with the current critic V_ϕ (Eq. 5) and advantages $A_{i,t}$ (Eq. 6)
 - 3: Update the critic on all request states of the round with $\mathcal{L}_V$
 - 4: Accumulate the policy gradient of $\mathcal{L}_\pi$ on up to 8 sampled requests per episode
 - 5: Every 4 rounds, take an optimizer step on θ and synchronize the weights to the rollout servers
-

Table 1 SFT vs. RL across the LIBERO benchmark family. We report host and strict success rates (%) and the paired RL gain Δ (in percentage points) over CaPtain-4B-SFT across n evaluation episodes per suite. Exact McNemar p -values are in Appendix K ($p < 10^{-5}$ on all primary suites).

Benchmark	n	Host Success (%)				Strict Success (%)		
		Base	SFT	CaPtain-4B	Δ	SFT	CaPtain-4B	Δ
LIBERO	240	0.0	55.4	73.8	+18.3	52.1	67.5	+15.4
LIBERO-PRO (3 suites)	600	0.3	59.5	78.5	+19.0	56.7	71.5	+14.8
LIBERO-PRO (All 4 suites)	800	0.2	49.2	68.6	+19.4	45.1	58.9	+13.8
LIBERO-Plus	700	0.0	48.6	66.1	+17.6	44.6	59.0	+14.4
LIBERO-PRO (4 categories)	160	0.0	48.8	68.8	+20.0	48.1	60.6	+12.5
LIBERO-MAX Dynamic	80	0.0	40.0	61.2	+21.2	40.0	50.0	+10.0
LIBERO-MAX Base	80	0.0	51.2	70.0	+18.8	45.0	63.8	+18.8

4 Experiments

Benchmarks. We train only on the 40 original LIBERO tasks [44] and evaluate on LIBERO and three perturbation suites built on them, whose perturbations never appear in training. *LIBERO* uses the official held-out initial states 11–16 of all 40 tasks (240 episodes). *LIBERO-PRO* [97] perturbs object positions and task semantics across the four suites (100 episodes per subset); following prior work, we report the short-horizon 3-suite average (Spatial, Object, Goal; 600 episodes) and the full 4-suite average (adding Long; 800 episodes), as well as its 4-category split (160 episodes). *LIBERO-Plus* [18] is evaluated on a stratified sample of 700 cases across its seven perturbation dimensions. *LIBERO-MAX* [95] uses its 80 Lite cases under Base (static) and Dynamic (in-episode) perturbations. For zero-shot transfer we use RoboTwin 2.0 [11], RoboDojo [10], SimplerEnv [38] and CaP-Bench [20] (§4.3).

Protocol and metrics. All evaluations run in CaPtain-Harness with the settings of RL rollouts: temperature 1, thinking enabled, at most 60 requests, 150 physical actions and 1,080s per episode, and no early stopping. At test time, the simulator critic is removed and all object coordinates are estimated online from RGB-D camera observations and execution receipts. We report *host* success, the benchmark predicate on the final state, and *strict* success, which additionally requires that the agent ended the episode with `finish()`. Comparisons between our models are paired on identical (suite, task, initial-state) triples, with exact two-sided McNemar tests and 95% task-cluster bootstrap intervals.

Models and baselines. CaPtain-4B-SFT is the model after Stage 1, which also initializes RL and serves as the KL reference. We use the checkpoint at round 300 (75 optimizer updates) as CaPtain-4B. Baseline numbers are taken from the respective papers.

4.1 Main results

Table 1 and Figure 3 show the main results. On held-out LIBERO, RL raises host success from 55.4% to 73.8% (55 episodes gained, 11 lost) and strict success from 52.1% to 67.5%. False finishes fall from 7.1% to 2.1% of episodes, and the mean number of requests per episode falls from 39.1 to 32.6. The gains are largest on the goal suite (+26.7 host, $p=1.4\times 10^{-4}$) and the spatial suite (+18.3), and smallest on the long-horizon

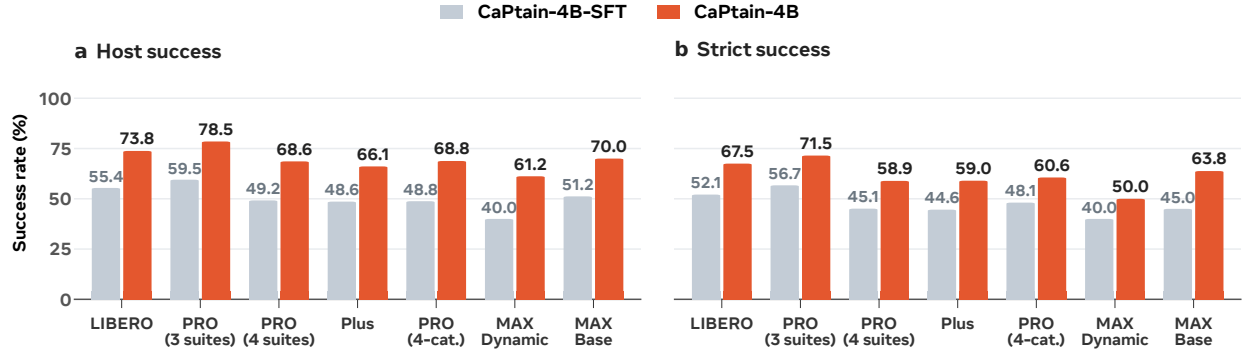


Figure 3 RL gains survive perturbation. SFT vs. RL host and strict success on held-out LIBERO and on four perturbation suites.

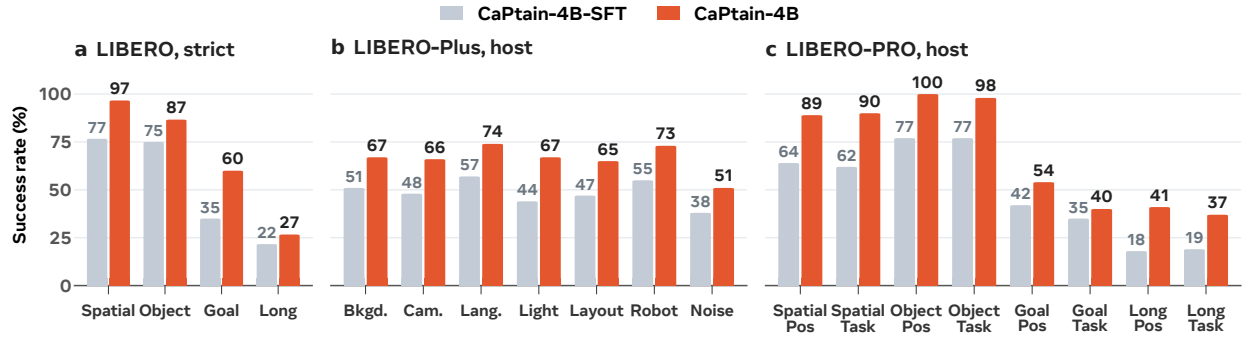


Figure 4 Breakdown, CaPtain-4B-SFT vs. CaPtain-4B. (a) LIBERO strict success by suite. (b) LIBERO-Plus host success by perturbation dimension. (c) LIBERO-PRO host success by suite and perturbation type.

suite (+13.3). Without post-training, the base Qwen3.5-4B solves essentially nothing (at most 0.3% on any suite): it exhausts its request budget misusing the interface.

Robustness to perturbations. The RL gain carries over to every perturbation suite: +19.0 host points on LIBERO-PRO across the 3 short-horizon suites (+19.4 across all 4 suites), +17.6 on LIBERO-Plus, and +21.2 (Dynamic) and +18.8 (Base) on LIBERO-MAX. On LIBERO-Plus, all seven perturbation dimensions improve at $p < 0.05$, and six of seven survive a Bonferroni correction (Figure 4b). The largest gains are for lighting (+23), camera, object layout and robot initial state (+18 each), and the smallest for sensor noise (+13). This is not the case for end-to-end policies trained on LIBERO: $\pi_{0.5}$ falls from 97% on standard LIBERO to 12.8% under the LIBERO-PRO position and task perturbations [97], and the VLA RL method reported on LIBERO-Plus stays at the level of its base model (RIPT-VLA 68.4 vs. OpenVLA-OFT 69.6 [18]), whereas the gains of CaPtain-4B persist on every suite.

4.2 Comparison with published systems

Table 2 compares CaPtain-4B against published LIBERO-PRO systems across three dimensions: whether the agent queries a frontier model at test time, whether it invokes a VLA skill trained on LIBERO demonstrations, and what auxiliary signals or training data it uses. Operating as a standalone 4B policy over low-level primitives, CaPtain-4B reaches 78.5% on the 3-suite average (74.6% on held-out initial state 11; CaPtain-4B-SFT: 57.8%, Appendix K). Among open-weight agents that do not delegate manipulation to a LIBERO-trained VLA, CaPtain-4B outperforms both SPARK (66.8%, which uses a 27B model with ground-truth depth and behavior-tree primitives developed on LIBERO-PRO) and Guava ($\approx 41\%$, a 4B model trained primarily via SFT), while also surpassing several frontier-API systems including K1 (77.2%), WAA (75.6%), and ASPIRE (71.7%). Compared with AgenticRoboRL—which trains a 9B planner around a frozen LIBERO VLA and gains

Table 2 Comparison with published systems on LIBERO-PRO (success %). Systems are grouped by paradigm. *Frontier API*: queries a closed frontier model at test time; *VLA Skill*: invokes a policy trained on LIBERO demonstrations; *In-Domain Data*: uses LIBERO demonstrations or LIBERO-PRO development states. Protocol notes are in Appendix K.

		Frontier	VLA	In-Domain	LIBERO-PRO (%)	
System	Policy Model	API	Skill	Data	3 Suites	All 4 Suites
<i>End-to-end VLAs & VLA-assisted agents</i>						
Harness VLA [94]	Claude Opus 4.7 + $\pi_{0.5}$	✓	✓	✓	87.7	82.4
AgenticRoboRL [48]	Qwen3.5-9B + RL-tuned $\pi_{0.5}$	✗	✓	✓	81.3	71.0
RLinf $\pi_{0.5}$ [94]	$\pi_{0.5}$ (RL fine-tuned)	✗	✓	✓	56.2	50.0
$\pi_{0.5}$ [97]	$\pi_{0.5}$ (SFT)	✗	✓	✓	12.8	10.8
<i>Closed frontier-model CaP agents</i>						
OpenRUA [13]	Claude Opus 5	✓	✗	✗	91.8	87.0
K1 [40]	Gemini 3.7 Flash	✓	✗	✓	77.2	–
WAA [93]	Gemini 3.7 Flash	✓	✗	✗	75.6	–
ASPIRE [50]	Claude Opus 4.6	✓	✗	✓	71.7	–
RACaP [39]	GPT-5.5	✓	✗	✓	45.0	–
RATs [88]	Gemini 3.1 Pro	✓	✗	✓	43.8	–
CaP-Agent0 [20]	3-model ensemble	✓	✗	✗	18.2	–
<i>Open-weight CaP agents (no VLA skill)</i>						
SPARK [23]	Qwen3.8-27B	✗	✗	✓	66.8	–
Guava [45]	Qwen3.5-4B (SFT+GRPO)	✗	✗	✗	≈41.0	–
CaPtain-4B-SFT (ours)	Qwen3.5-4B (SFT)	✗	✗	✗	59.5	49.2
CaPtain-4B (ours)	Qwen3.5-4B (SFT+RL)	✗	✗	✗	78.5	68.6

+5.9 points from RL on the 4-suite average (65.1%→71.0%)—multi-turn RL in CaPtain yields a +19.4-point gain on the same suites (49.2%→68.6%) directly over low-level motion primitives.

Cost. A CaPtain-4B episode on LIBERO uses on average 32.6 requests (median 26), 16.6 physical actions and 7.3k output tokens, and takes 417s of wall time on a GPU shared by eight episodes. This amounts to about 52 GPU-seconds of policy compute, or an estimated \$0.03–0.06 per episode at \$2–4 per GPU-hour, excluding perception services. Published code agents report \$0.18–0.24 (SPARK, WAA, RACaP) to \$8–22 (OpenRUA) per episode. Appendix Q compares CaPtain-4B with published VLAs, world-action models and code agents on all four LIBERO-family benchmarks.

4.3 Zero-shot transfer

We evaluate CaPtain-4B, trained only on LIBERO, on four benchmarks whose scenes and tasks differ from LIBERO, without further training; RoboDojo also changes the embodiment to two 6-DoF ARX X5 arms. On dual-arm robots, every motion tool accepts an optional **arm** argument. When it is omitted, the harness routes the call to the arm whose base is nearer the target, unless the active arm holds an object, and a **move_both** tool moves both arms together. Because the policy acts with one arm at a time, we also report a single-arm subset fixed by a mechanical rule: RoboTwin tasks whose expert program moves one arm at a time and holds each object with a single arm (33 of 50), and RoboDojo tasks whose specification requires no simultaneous two-arm action, handover or navigation (32 of 42). Table 3 compares CaPtain-4B against representative VLAs, world-action models (WAMs), and CaP agents across the four benchmarks (protocol details in Appendix K).

RoboTwin 2.0. Under the official Hard protocol (100 randomized trials per task, 5,000 episodes), CaPtain-4B succeeds in 22.6% of episodes on all 50 tasks and in 30.0% on the single-arm subset. Policies co-trained on the clean demonstrations of all 50 tasks reach 46.0% ($\pi_{0.5}$) and 48.7% (OpenWAM- α). On the 250 randomized scenes used by PyRUA-Lean, RL improves over CaPtain-4B-SFT (21.6% vs. 16.8%) and reduces false finishes from 45 to 16; PyRUA-Lean reaches 52.4% there with a 4B planner, using the native embodiment, a RoboTwin-trained VLA tool and a collision-aware motion planner.

Table 3 Zero-shot cross-benchmark transfer. Trained exclusively on LIBERO, CaPtain-4B is evaluated zero-shot across four unseen benchmarks and compared against in-domain VLAs/WAMs (trained on target-benchmark demonstrations) and CaP agents. RoboDojo reports official leaderboard Score and success rate (SR %); all other benchmarks report success rate (%).

(a) RoboTwin 2.0 (Randomized Hard)					(b) RoboDojo-Sim (Dual 6-DoF ARX X5)			
Method	Type	All	1-Arm		Method	Backbone / Type	Score	SR
PyRUA-Lean Coder-4B [67]	CaP+VLA	52.4	–		RoboDojo Agent [90]	GPT-6 Astra	29.0	22.5
OpenWAM- α [76]	WAM	48.7	–		RoboDojo Agent [90]	DeepSeek-Flash	3.0	1.9
$\pi_{0.5}$ [55]	VLA	46.0	45.8		RoboDojo Agent [90]	GPT-5.5	1.1	0.9
X-VLA [96]	VLA	20.9	–		RDT [47]	In-domain VLA	0.5	0.1
Xiaomi-Robotics-0 [5]	VLA	18.2	–		OpenVLA-OFT [31]	In-domain VLA	0.2	0.0
π_0 [2]	VLA	16.3	18.3		CaPtain-4B (ours, all 42)	Qwen3.5-4B	1.7	0.8
RDT [47]	VLA	13.7	14.0		CaPtain-4B (ours, 1-arm 32)	Qwen3.5-4B	2.0	1.0
CaPtain-4B (ours, zero-shot)	CaP	22.6	30.0					

(c) SimplerEnv (WidowX Visual Matching)				(d) CaP-Bench (Cube Lift & Stack)		
Method	Backbone / Type	SR (%)		Method	Backbone	SR (%)
ManiAgent [80]	GPT-5	86.8		CaP-X (multi-turn) [20]	Gemini-3-Pro	73.5
ManiAgent [80]	Grok-4	83.0		CaP-RL (in-domain RL) [20]	Qwen2.5-7B	62.0
PhysBrain 1.0 [41]	Bridge VLA	80.2		CaP-X (multi-turn) [20]	GPT-5.2	55.5
SoFar [57]	Modular Agent	58.3		CaP-X (multi-turn) [20]	Claude Opus 4.5	54.0
$\pi_{0.5}$ [55]	Bridge VLA	57.1		CaP-X (single-turn) [20]	Qwen2.5-7B	14.5
π_0 [2]	Bridge VLA	27.1		Base Qwen3.5-4B	Qwen3.5-4B	5.0
OpenVLA-OFT [31]	Bridge VLA	23.4		CaPtain-4B-SFT (ours)	Qwen3.5-4B	55.0
CaPtain-4B (ours, zero-shot)	Qwen3.5-4B	83.3		CaPtain-4B (ours, zero-shot)	Qwen3.5-4B	75.0

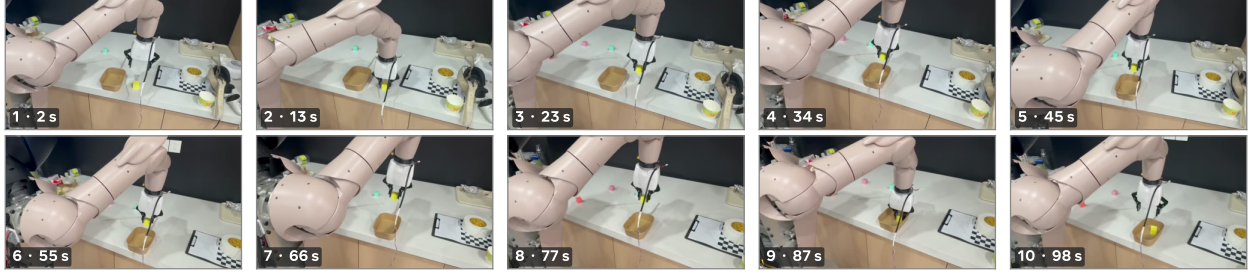
RoboDojo. RoboDojo evaluates zero-shot cross-embodiment transfer from LIBERO’s single 7-DoF Panda to dual 6-DoF ARX X5 arms across 42 tasks. We compare against the RoboDojo agent driven by GPT-6 Astra, DeepSeek-Flash, or GPT-5.5, as well as two VLAs (RDT and OpenVLA-OFT) fine-tuned directly on the RoboDojo training set. While GPT-6 Astra reaches a leaderboard Score of 29.0, the benchmark remains challenging for both smaller frontier models (DeepSeek-Flash at 3.0, GPT-5.5 at 1.1) and in-domain VLAs (RDT at 0.5, OpenVLA-OFT at 0.2). Without any RoboDojo training data, CaPtain-4B achieves a Score of 1.7 (0.8% success overall; 2.0 Score and 1.0% success on the 32 single-arm tasks), solving tasks in the Generalization and Open splits. The primary bottleneck for CaPtain-4B on the X5 hardware is kinematic reach, as the 6-DoF arm’s top-down workspace extends only ≈ 0.5 m from its base (Appendix J).

SimplerEnv. On the four WidowX tasks of SimplerEnv [38] (24 visual-matching episodes each), CaPtain-4B succeeds in 83.3% of episodes on average (spoon on towel 75.0, carrot on plate 79.2, block stacking 91.7, eggplant in basket 87.5). This is close to ManiAgent with GPT-5 (86.8), above ManiAgent with Grok-4 and SoFar, agents that also use depth and require no robot-policy training, and above VLAs fine-tuned on Bridge demonstrations such as PhysBrain 1.0 (80.2) and $\pi_{0.5}$ (57.1).

CaP-Bench. Following CaP-X [20], which reports per-model breakdowns on Cube Lift and Cube Stack, we evaluate CaPtain-4B zero-shot under CaP-Gym’s standard step horizon. CaPtain-4B reaches a score of 75.0, outperforming both closed frontier models in CaP-X’s multi-turn visual setting—including Gemini-3-Pro (73.5), GPT-5.2 (55.5), and Claude Opus 4.5 (54.0)—and the open 7B coder trained with CaP-RL directly on CaP-Bench (62.0). Multi-turn RL raises the horizon score from 55.0 (CaPtain-4B-SFT) to 75.0, compared with 5.0 for the untrained Qwen3.5-4B in the same harness. When allowed to run to the full episode horizon, CaPtain-4B completes 20 of 20 trials on Cube Lift and 19 of 20 on Cube Stack (97.5% mean success), showing that the gap between 75.0 and 97.5% reflects execution speed relative to CaP-Gym’s step cutoff rather than task failure. On the remaining three tasks of the five-task suite, CaPtain-4B solves Re-stack in 7 of 20 episodes, does not solve Nut insertion, and cannot execute Wipe because the task requires sustained downward pressing that our guarded-contact motion primitives prevent.

Across the transfer benchmarks, the most common failures are tasks that need simultaneous two-arm motion and the reach limits of unfamiliar arms (Appendix J). Appendix P shows one successful episode each from

a “Put the yellow cube in the wooden tray.”



b “Put the cube that has the color of the sky into the container that is not round.”

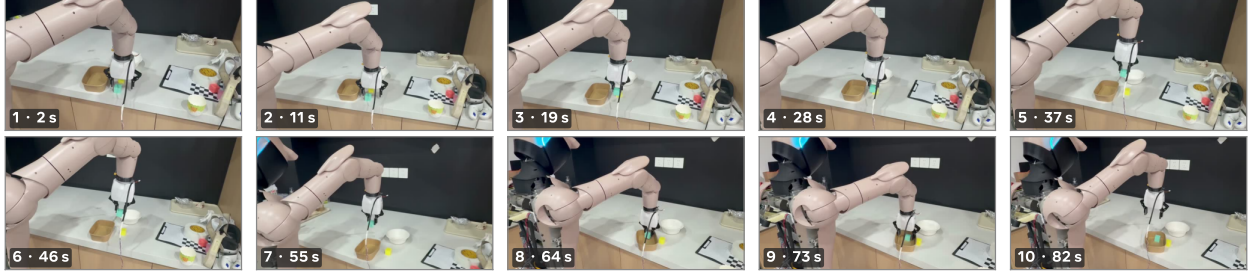


Figure 5 Real-robot episodes. Two tasks with new objects and layouts; ten frames per episode, with time stamps. (b) requires reasoning about the instruction: the agent must infer that the cube with the color of the sky is the light-blue one and that the container that is not round is the rectangular tray.

RoboTwin 2.0 and RoboDojo, with the head-camera frames and the code CaPtain-4B wrote.

4.4 Transfer to a real robot

We also test whether the pipeline carries over to a real robot through CaPtain-Harness. The simulator is replaced by a hardware backend for an Astrabot XR1 arm with a head RGB-D camera, a wrist camera, a kinematic twin updated from the measured joints, and safety guards. The policy-facing API and receipts are unchanged, and the critic is not used. Because the XR1 arm differs from the simulated Panda in kinematics, reach, gripper geometry and camera placement, we use a checkpoint fine-tuned on episodes from a coarse twin simulation of the XR1, which teaches the policy the conventions of the new embodiment. The twin uses primitive objects and untextured surfaces and does not reconstruct the real scene (Figure 6); the policy is adapted with simulated XR1 episodes only and uses no real-world data.

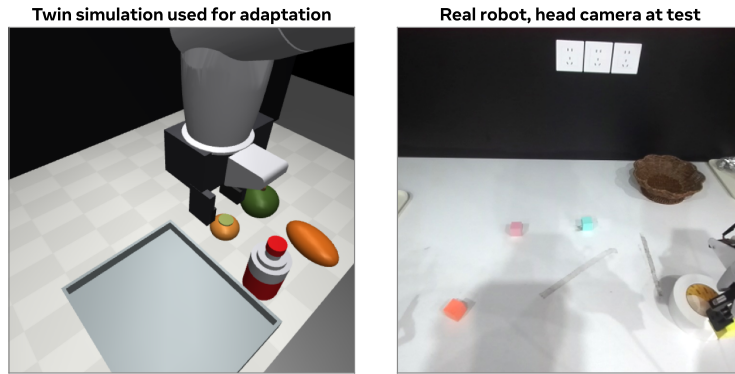


Figure 6 The twin simulation used to adapt the policy to the XR1 arm (left) and the real scene at test time (right).

Figure 5 shows two tasks with new objects and layouts. In the first, the agent puts a yellow cube into a wooden tray. In the second, the instruction refers to the objects only indirectly, so the agent must first resolve which

cube has the color of the sky and which container is not round before it can act. Both tasks are completed with the same low-level primitives used in simulation. Appendix O gives step-by-step traces of both episodes.

4.5 Analysis of the simulator critic

To understand what the simulator critic learns and why it stabilizes multi-turn RL, we analyze offline rollouts from an earlier 9B policy (15k episodes, 573k request states) alongside training-time logs.

Value ranking and credit concentration. The critic predicts final episode success with an overall AUC of 0.970, a within-group AUC of 0.904 across rollouts sharing the same start state, and a calibration error of 0.006 (the warm-start critic used for 4B RL achieves 0.907 held-out AUC). Because the one-step TD difference (Eq. 5) holds the request index fixed, credit concentrates sharply on turns that alter the physical scene: successful grasps receive a median ΔV of +0.058 (IQR [0.036, 0.090]) and failed grasps receive -0.0075 , whereas perception-only requests receive -0.0002 . Aggregated over whole trajectories, 65% of the total $|A|$ mass is assigned to motions, 14% to grasps, and 14% to releases, while perception-only turns account for just 1.3%.

Why policy-side features and LM hidden states are excluded. When counters of the policy’s own actions (e.g., cumulative perception or failed calls) are provided to the critic, within-group AUC on pre-grasp states rises from 0.773 to 0.862, but these counters act as proxies for elapsed time and leak negative credit onto perception turns. Similarly, a linear probe on the language model’s hidden state decodes the request index with $R^2=0.80$; feeding hidden states to the critic shifts 35% of the total $|A|$ mass onto perception requests. Restricting the critic inputs to privileged physical state and evaluating ΔV at a fixed time index prevents this credit leakage and reduces the gradient-noise scale on 9B SFT rollouts from 9.0k episodes under outcome-only GRPO to 6.7k episodes ($\lambda=0.9$; Appendix H).

5 Limitations

- **Single-benchmark RL training and bimanual control.** Our RL stage is trained exclusively on the 40 LIBERO tasks with a single 7-DoF arm. While CaPtain-4B transfers zero-shot to unseen single-arm tasks and embodiments, cross-benchmark performance remains below in-domain accuracy, and tasks requiring simultaneous bimanual coordination or tight kinematic reach on unfamiliar 6-DoF arms are not yet addressed (§4.3).
- **Privileged state during RL.** Training the simulator critic relies on exact simulator state. In our offline sensitivity study, replacing ground-truth object poses with noisy estimates degrades correlation with the privileged ΔV (0.49 under 2 cm pose noise, 0.36 under 5 cm, and 0.21 using the policy’s own perception), indicating that online RL directly on physical hardware would require an external state estimator.
- **Interface and SFT coverage.** The agent’s action space is bounded by the 22 capabilities of CaPtain-Harness and the behaviors demonstrated in the synthetic SFT trajectories (Appendix C). Tasks that require continuous force-controlled compliance—such as Wipe on CaP-Bench, which demands sustained downward pressing—cannot be expressed by guarded Cartesian position primitives.

6 Conclusion

We presented CaPtain, an open post-training pipeline that turns a 4B vision-language model into a standalone code-as-policy robot agent operating directly over low-level perception and motion primitives. By pairing a portable execution harness (CaPtain-Harness) with synthetic SFT trajectories and multi-turn RL guided by a privileged simulator critic, CaPtain converts sparse episode outcomes into dense per-request credit concentrated on physical manipulation decisions, while requiring no privileged information, demonstration-trained VLA skills, or frontier API calls at test time. Across held-out LIBERO and three perturbation suites, CaPtain-4B improves substantially over SFT, matches or surpasses several frontier-model agents at a small fraction of their inference cost, transfers zero-shot to unseen benchmarks and embodiments, and deploys on a physical robot through simulated embodiment adaptation alone. Extending the simulator critic to value

termination decisions, bimanual coordination, and multi-embodiment workspaces offers a direct path toward closing the remaining gap to frontier systems.

References

- [1] Andrea Baisero and Christopher Amato. Unbiased Asymmetric Reinforcement Learning under Partial Observability. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2022.
- [2] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, et al. π_0 : A Vision-Language-Action Flow Model for General Robot Control. In *Robotics: Science and Systems (RSS)*, 2025.
- [3] George Bredis, Stanislav Dereka, Viacheslav Sinii, Ruslan Rakhimov, and Daniil Gavrilov. Enhancing Vision-Language Model Training with Reinforcement Learning in Synthetic Worlds for Real-World Success. *arXiv preprint arXiv:2508.04280*, 2025.
- [4] Qingwen Bu, Yanting Yang, Jisong Cai, Shenyuan Gao, Guanghui Ren, Maoqing Yao, Ping Luo, and Hongyang Li. UniVLA: Learning to Act Anywhere with Task-centric Latent Actions. In *Robotics: Science and Systems (RSS)*, 2025.
- [5] Rui Cai, Jun Guo, Xinze He, et al. Xiaomi-Robotics-0: An Open-Sourced Vision-Language-Action Model with Real-Time Execution. *arXiv preprint arXiv:2602.12684*, 2026.
- [6] Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, et al. SAM 3: Segment Anything with Concepts. In *International Conference on Learning Representations (ICLR)*, 2026.
- [7] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by Cheating. In *Conference on Robot Learning (CoRL)*, 2019.
- [8] Hanyang Chen, Mark Zhao, Rui Yang, Qinwei Ma, Ke Yang, et al. ERA: Transforming VLMs into Embodied Agents via Embodied Prior Learning and Online Reinforcement Learning. *arXiv preprint arXiv:2510.12693*, 2025.
- [9] Kang Chen, Zhihao Liu, Tonghe Zhang, Zhen Guo, Si Xu, et al. π_{RL} : Online RL Fine-tuning for Flow-based Vision-Language-Action Models. *arXiv preprint arXiv:2510.25889*, 2025.
- [10] Tianxing Chen, Yue Chen, Zixuan Li, Junyuan Tang, Kailun Su, et al. RoboDojo: A Unified Sim-and-Real Benchmark for Comprehensive Evaluation of Generalist Robot Manipulation Policies. *arXiv preprint arXiv:2607.04434*, 2026.
- [11] Tianxing Chen, Zanzin Chen, Baijun Chen, Zijian Cai, Yibin Liu, et al. RoboTwin 2.0: A Scalable Data Generator and Benchmark with Strong Domain Randomization for Robust Bimanual Robotic Manipulation. In *International Conference on Machine Learning (ICML)*, 2026.
- [12] Yanzhe Chen, Zhijun Cao, and Mike Zheng Shou. Where Success Breaks: Failure-Boundary Learning for Robust Vision-Language-Action Models. *arXiv preprint arXiv:2609.06114*, 2026.
- [13] Zhaoyang Chu, Earl T. Barr, Claire Le Goues, Peter O’Hearn, Mark Harman, Federica Sarro, and He Ye. OpenRUA: Robot-Use Agents Are Zero-Shot Visuomotor Policies. *arXiv preprint arXiv:2610.02459*, 2026.
- [14] Christopher Clark, Jieyu Zhang, et al. Molmo2: Open Weights and Data for Vision-Language Models with Video Understanding and Grounding. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.
- [15] Daniel Ebi, Damien Ernst, Klemens Böhm, and Gaspard Lambrechts. Informed Asymmetric Actor-Critic: Leveraging Privileged Signals Beyond Full-State Access. In *International Conference on Machine Learning (ICML)*, 2026.
- [16] Karim Elmaaroufi, Justin Svegliato, Sarunas Kalade, Graham Schelle, Sanjit A. Seshia, and Matei Zaharia. RHO: Your Coding Agent is Secretly a Robotocist. *arXiv preprint arXiv:2606.16458*, 2026.
- [17] Haoquan Fang, Jiafei Duan, Donovan Clay, Sam Wang, Shuo Liu, et al. MolmoAct2: Action Reasoning Models for Real-world Deployment. In *Conference on Robot Learning (CoRL)*, 2026.
- [18] Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, et al. LIBERO-Plus: In-depth Robustness Analysis of Vision-Language-Action Models. *arXiv preprint arXiv:2510.13626*, 2025.
- [19] Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-Group Policy Optimization for LLM Agent Training. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

- [20] Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, et al. CaP-X: A Framework for Benchmarking and Improving Coding Agents for Robot Manipulation. In *International Conference on Machine Learning (ICML)*, 2026.
- [21] Liane Galanti, Dhruv Shah, and Tri Dao. Addressing the Orchestration Gap in Generalist Robots via Physical Agency. *arXiv preprint arXiv:2607.21725*, 2026.
- [22] Google DeepMind. Gemini 4 Argon for Robotics. Google Developers Blog, <https://developers.googleblog.com/gemini-4-argon-for-robotics/>, 2026. October 13, 2026.
- [23] Bryce Grant, Aryeh Rothenberg, Logan Senning, Zonghe Chua, Zach Patterson, and Peng Wang. Sequential Planning via Anchored Robotic Keypoints. *arXiv preprint arXiv:2606.30613*, 2026.
- [24] Meng-Hao Guo, Zhe-Han Mo, Jia-Jun Wang, Yi Zhang, Kejin Wang, et al. Transferring the Intelligence of VLMs to Robotic Control. *arXiv preprint arXiv:2609.22966*, 2026.
- [25] Zhenyu Hou, Yujiang Li, Jie Tang, and Yuxiao Dong. Single-Rollout Asynchronous Optimization for Agentic Reinforcement Learning. *arXiv preprint arXiv:2607.07508*, 2026.
- [26] Bo Huang, Fengxiang Li, Hao Xu, Haoyang Huang, Hongyi Fu, et al. KAT-Coder-V2.5 Technical Report. *arXiv preprint arXiv:2607.05471*, 2026.
- [27] Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models. In *Conference on Robot Learning (CoRL)*, 2023.
- [28] Yifan Kang, Zihan Wang, Zhiwen Fan, and Bangya Liu. Encore: Few-Shot Agentic Discovery of Manipulation Strategies. *arXiv preprint arXiv:2609.37359*, 2026.
- [29] Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. VinePPO: Refining Credit Assignment in RL Training of LLMs. In *International Conference on Machine Learning (ICML)*, 2025.
- [30] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, et al. OpenVLA: An Open-Source Vision-Language-Action Model. In *Conference on Robot Learning (CoRL)*, 2024.
- [31] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-Tuning Vision-Language-Action Models: Optimizing Speed and Success. In *Robotics: Science and Systems (RSS)*, 2025.
- [32] Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, and Jinwei Gu. Cosmos Policy: Fine-Tuning Video Models for Visuomotor Control and Planning. In *International Conference on Learning Representations (ICLR)*, 2026.
- [33] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. RMA: Rapid Motor Adaptation for Legged Robots. In *Robotics: Science and Systems (RSS)*, 2021.
- [34] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, et al. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2023.
- [35] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning Quadrupedal Locomotion over Challenging Terrain. *Science Robotics*, 5(47):eabc5986, 2020.
- [36] Haozhan Li, Yuxin Zuo, Jiale Yu, Yuhao Zhang, Zhaohui Yang, et al. SimpleVLA-RL: Scaling VLA Training via Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*, 2026.
- [37] Junbo Li, Peng Zhou, Rui Meng, Meet P. Vadera, Lihong Li, and Yang Li. Turn-PPO: Turn-Level Advantage Estimation with PPO for Improved Multi-Turn RL in Agentic LLMs. In *Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2026.
- [38] Xuanlin Li, Kyle Hsu, Jiayuan Gu, et al. Evaluating Real-World Robot Manipulation Policies in Simulation. In *Conference on Robot Learning (CoRL)*, 2024.
- [39] Zexi Li, Yehang Zhang, Haojian Huang, Bohan Zhou, Wenqian Li, et al. RACaP: Agentic Reasoning, Acting, and Coding as Policies for Evolvable Robot Learning. *arXiv preprint arXiv:2609.29394*, 2026.
- [40] Zexi Li, Yehang Zhang, Wenqian Li, Haojian Huang, Chenxu Wang, et al. Robo-Harness K1: Harnessing Robot-Use Agents via Perception Augmentation. *arXiv preprint arXiv:2609.29389*, 2026.
- [41] Shijie Lian, Bin Yu, Xiaopeng Lin, et al. PhysBrain 1.0 Technical Report. *arXiv preprint arXiv:2605.15298*, 2026.

- [42] Dayang Liang, Lang Feng, Bo An, and Yunlong Liu. SAPO: Single-Rollout Autoregressive Policy Optimization for Agentic Reinforcement Learning. *arXiv preprint arXiv:2608.19842*, 2026.
- [43] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as Policies: Language Model Programs for Embodied Control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.
- [44] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking Knowledge Transfer for Lifelong Robot Learning. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2023.
- [45] Haowen Liu, Xirui Li, Shaoxiong Yao, Peng Shi, Tianyi Zhou, Jia-Bin Huang, Furong Huang, and Jiayuan Mao. Guava: Distilling Frontier VLMs into a Compact Agent through a Robotic Manipulation Harness. *arXiv preprint arXiv:2606.18363*, 2026.
- [46] Jijia Liu, Feng Gao, Bingwen Wei, Xinlei Chen, Qingmin Liao, Yi Wu, Chao Yu, and Yu Wang. What Can RL Bring to VLA Generalization? An Empirical Study. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.
- [47] Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, et al. RDT-1B: a Diffusion Foundation Model for Bimanual Manipulation. In *International Conference on Learning Representations (ICLR)*, 2025.
- [48] Xinxin Liu, Bingxin Xu, Wenrui Bao, Ming Li, Wentao Bao, Chen Chen, and Yuzhang Shang. AgenticRoboRL: Bringing Coding-Agent Post-Training to Agentic Robotics. *arXiv preprint arXiv:2610.28589*, 2026.
- [49] Guanxing Lu, Wenkai Guo, Chubin Zhang, Yuheng Zhou, Haonan Jiang, Zifeng Gao, Yansong Tang, and Ziwei Wang. VLA-RL: Towards Masterful and General Robotic Manipulation with Scalable Reinforcement Learning. *arXiv preprint arXiv:2505.18719*, 2025.
- [50] Runyu Lu, Yubo Wu, Ethan Kou, Haotian Lin, Letian Fu, et al. ASPIRE: Agentic /Skills Discovery for Robotics. *arXiv preprint arXiv:2607.00272*, 2026.
- [51] Ming Ma, Yi Zhu, Yiran Zhong, Feida Zhu, Chonghan Liu, et al. ProCredit: From Outcome Rewards to Progress Credit in Agentic Reinforcement Learning. *arXiv preprint arXiv:2609.27532*, 2026.
- [52] NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, et al. GR00T N1: An Open Foundation Model for Generalist Humanoid Robots. *arXiv preprint arXiv:2503.14734*, 2025.
- [53] NVIDIA GEAR Team. GR00T N1.5: An Improved Open Foundation Model for Generalist Humanoid Robots. https://research.nvidia.com/labs/gear/gr00t-n1_5/, 2025. Project page, June 2025.
- [54] Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. FAST: Efficient Action Tokenization for Vision-Language-Action Models. In *Robotics: Science and Systems (RSS)*, 2025.
- [55] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, et al. $\pi_{0.5}$: a Vision-Language-Action Model with Open-World Generalization. In *Conference on Robot Learning (CoRL)*, 2025.
- [56] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric Actor Critic for Image-Based Robot Learning. In *Robotics: Science and Systems (RSS)*, 2018.
- [57] Zekun Qi, Wenyao Zhang, Yufei Ding, et al. SoFar: Language-Grounded Orientation Bridges Spatial Reasoning and Object Manipulation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.
- [58] Chonghan Qin, Dongming Wu, Kaituo Feng, Jiaxu Wang, Xinlei Yu, Peiwen Sun, Yijia Fan, Xilin He, Bingao Chen, Weitong Lian, et al. Towards agentic robots: Model, data, environment, and harness. *arXiv preprint arXiv:2610.20442*, 2026.
- [59] Qwen Team. Qwen3.5: Towards Native Multimodal Agents. <https://qwen.ai/blog?id=qwen3.5>, 2026. Blog post, February 2026.
- [60] RoboDojo Team. RoboDojo Leaderboard. <https://robodojo-benchmark.com/leaderboard>, 2026. Accessed 2026-10-24.
- [61] RoboTwin Team. RoboTwin 2.0 Leaderboard. <https://robotwin-platform.github.io/leaderboard>, 2026. Accessed 2026-10-26.

- [62] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *International Conference on Learning Representations (ICLR)*, 2016.
- [63] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [64] SenseNova Team. SenseNova-RoboRSI: Recursive Self-Improvement for Embodied Agents. <https://github.com/WillWangZiHuan/SenseNova-RoboRSI>, 2026. 27B-Preview weights released; technical report forthcoming.
- [65] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, et al. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300*, 2024.
- [66] Chengshuai Shi, Wenzhe Li, Xinran Liang, Yizhou Lu, Wenjia Yang, et al. Odysseus: Scaling VLMs to 100+ Turn Decision-Making in Games via Reinforcement Learning. *arXiv preprint arXiv:2605.00347*, 2026.
- [67] Ruiyang Si, Jianxin Bi, Shunyu Yang, Rui Ni, Wenbo Huang, et al. Fewer Tokens, Better Action: GPT-6 Astra Robot Agents with 14% Higher Success Rate but 65% Fewer Tokens. *arXiv preprint arXiv:2610.01939*, 2026.
- [68] Shuhan Tan, Kairan Dou, Yue Zhao, and Philipp Krähenbühl. Interactive Post-Training for Vision-Language-Action Models. *arXiv preprint arXiv:2505.17016*, 2025.
- [69] Brian Y. Tsui, Alan Y. Fang, and Tiffany J. Hwu. Demonstration-Free Robotic Control via LLM Agents. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2026.
- [70] Siddarth Venkatraman, Matthieu Dinot, and Laurence Aitchison. Le Critique: Privileged Value Functions for LLM Reinforcement Learning. *arXiv preprint arXiv:2608.16739*, 2026.
- [71] Daoyu Wang, Qingchuan Li, Mingyue Cheng, Jie Ouyang, Shuo Yu, et al. CAPO: Critic-Guided Action-Aligned Policy Optimization for Advancing LLM Agent Capabilities. *arXiv preprint arXiv:2604.18401*, 2026.
- [72] Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, et al. UI-TARS-2 Technical Report: Advancing GUI Agent with Multi-Turn Reinforcement Learning. *arXiv preprint arXiv:2509.02544*, 2025.
- [73] Jiayu Wang, Bin Zhu, Yue Yu, et al. MoWAM: Explicit Future Motion Prediction for Efficient World Action Models. *arXiv preprint arXiv:2609.20709*, 2026.
- [74] Kangrui Wang, Pingyue Zhang, Zihan Wang, Yaning Gao, Linjie Li, et al. VAGEN: Reinforcing World Model Reasoning for Multi-Turn VLM Agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.
- [75] Yu Wang, Yi-Kai Zhang, Wentao Shi, Ziang Ye, Yuchun Miao, et al. CAST: Game Solvers as Turn-Level Teachers for LLM Agents. *arXiv preprint arXiv:2607.25308*, 2026.
- [76] Yuran Wang, Siqiao Huang, Mingleyang Li, et al. OpenWAM: An Open, Modular Exploration Towards Systematic World-Action Model Pretraining. *arXiv preprint arXiv:2609.07398*, 2026.
- [77] Zimo Wen, Yijin Chen, Yuxuan Cao, Wendi Chen, Yanwen Zou, et al. RoboRSI: Stable, efficient, and reusable robot self-evolution in complex real-world environments. *arXiv preprint arXiv:2610.12424*, 2026.
- [78] Zhongwen Xu and Zihan Ding. Single-stream Policy Optimization. In *International Conference on Learning Representations (ICLR)*, 2026.
- [79] Haodong Yan, Junfeng Li, Junjie He, et al. Robust-WAM: Bridging Generative Pretraining and Semantic Foresight in World-Action Models. *arXiv preprint arXiv:2608.05903*, 2026.
- [80] Yi Yang, Kefan Gu, Yuqing Wen, et al. ManiAgent: An Agentic Framework for General Robotic Manipulation. *arXiv preprint arXiv:2510.11660*, 2025.
- [81] Zhejian Yang, Yongchao Chen, Xueyang Zhou, Jiangyue Yan, Dingjie Song, Yinuo Liu, Yuting Li, Yu Zhang, Pan Zhou, Hechang Chen, and Lichao Sun. Agentic Robot: A Brain-Inspired Framework for Vision-Language-Action Models in Embodied Agents. *arXiv preprint arXiv:2505.23450*, 2025.
- [82] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR)*, 2023.
- [83] Heng Yu, David D. Yuan, Juze Zhang, et al. OpenWAM: An Open Framework for Composable World-Action Models. *arXiv preprint arXiv:2610.07922*, 2026.

- [84] Tianyuan Yuan, Zibin Dong, Yicheng Liu, and Hang Zhao. Fast-WAM: Do World Action Models Need Test-time Future Imagination? *arXiv preprint arXiv:2603.16666*, 2026.
- [85] Hongzhi Zang, Mingjie Wei, Si Xu, Yongji Wu, Zhen Guo, et al. RLinf-VLA: A Unified and Efficient Framework for Reinforcement Learning of Vision-Language-Action Models. In *Robotics: Science and Systems (RSS)*, 2026.
- [86] Bo-Wen Zhang, Junwei He, Maoqi Liu, Feiran Li, Song-Lin Lv, et al. T2SPO: Trajectory-to-Step Policy Optimization for Agentic Reinforcement Learning. *arXiv preprint arXiv:2610.00388*, 2026.
- [87] Chenchen Zhang. From Reasoning to Agentic: Credit Assignment in Reinforcement Learning for Large Language Models. *arXiv preprint arXiv:2604.09459*, 2026.
- [88] Junyi Zhang, Jiaxin Ge, Hanjun Yoo, Letian Fu, Zihan Yang, et al. Playful Agentic Robot Learning. *arXiv preprint arXiv:2606.19419*, 2026.
- [89] Rui Zhang, Qiwei Wu, Zhengyu Zhang, Tao Li, Hongyu Zhou, et al. A GPU-Parallel Framework for Heterogeneous Multi-Task Reinforcement Learning. *arXiv preprint arXiv:2606.03335*, 2026.
- [90] Wenbo Zhang, Kaixuan Wang, Yutao Ouyang, Xiaoyu Huang, Liyang Li, et al. An Unexpected Robot Policy: Early Evaluations of GPT-6 Astra on RoboDojo and Beyond. *arXiv preprint arXiv:2609.24170*, 2026.
- [91] Wenxuan Zhang, Yuhui Wang, Donggang Jia, Xiaoqian Shen, Jian Ding, Ivan Viola, Jürgen Schmidhuber, and Mohamed Elhoseiny. Hybrid Advantage Estimation with Unified Critic for VLM Agentic Reinforcement Learning. In *European Conference on Computer Vision (ECCV)*, 2026.
- [92] Xiaopeng Zhang, Yueyang Weng, Qi Liu, Yongjin Mu, and Yanjie Li. Learning Robust Execution in Robotic Manipulation with Agentic Reinforcement Learning. *arXiv preprint arXiv:2607.13818*, 2026.
- [93] Yehang Zhang, Haojian Huang, Yifan Chang, Jianchong Su, Bohan Zhou, et al. World Action Agent: Harnessing VLMs for Robot Manipulation via World Action Rehearsal. *arXiv preprint arXiv:2609.29964*, 2026.
- [94] Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, et al. Harness VLA: Steering Frozen VLAs into Reliable Manipulation Primitives via Memory-Guided Agents. *arXiv preprint arXiv:2607.08448*, 2026.
- [95] Yunbei Zhang, Zijian Jin, Yuanzhe Liu, Janet Wang, Xilun Zhang, et al. LIBERO-MAX: Do Robot Policies Adapt When the World Changes? *arXiv preprint arXiv:2609.36518*, 2026.
- [96] Jinliang Zheng, Jianxiong Li, Zhihao Wang, et al. X-VLA: Soft-Prompted Transformer as Scalable Cross-Embodiment Vision-Language-Action Model. In *International Conference on Learning Representations (ICLR)*, 2026.
- [97] Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. LIBERO-PRO: Towards Robust and Fair Evaluation of Vision-Language-Action Models Beyond Memorization. *arXiv preprint arXiv:2510.03827*, 2025.
- [98] Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. ArCHer: Training Language Model Agents via Hierarchical Multi-Turn RL. In *International Conference on Machine Learning (ICML)*, 2024.
- [99] Yifei Zhou, Song Jiang, Yuandong Tian, Jason Weston, Sergey Levine, Sainbayar Sukhbaatar, and Xian Li. SWEET-RL: Training Multi-Turn LLM Agents on Collaborative Reasoning Tasks. *arXiv preprint arXiv:2503.15478*, 2025.

Appendix

A	Implementation details	20
A.1	Policy and harness	20
A.2	RL	21
B	Robot API	22
C	SFT details	22
D	Diversity of tasks and layouts during post-training	23
E	Reward, costs and early stopping	24
F	RL infrastructure	24
G	Lessons from RL on a multi-turn agent	25
H	Gradient-noise analysis	25
I	Host and strict success late in RL	25
J	Failure taxonomy for zero-shot transfer	26
K	Additional LIBERO results	27
L	Harness adaptation per transfer benchmark	27
M	Compute accounting	28
N	Example episode	28
O	Real-robot episode traces	29
P	Zero-shot transfer episodes	29
Q	Full LIBERO-family comparison	32

A Implementation details

A.1 Policy and harness

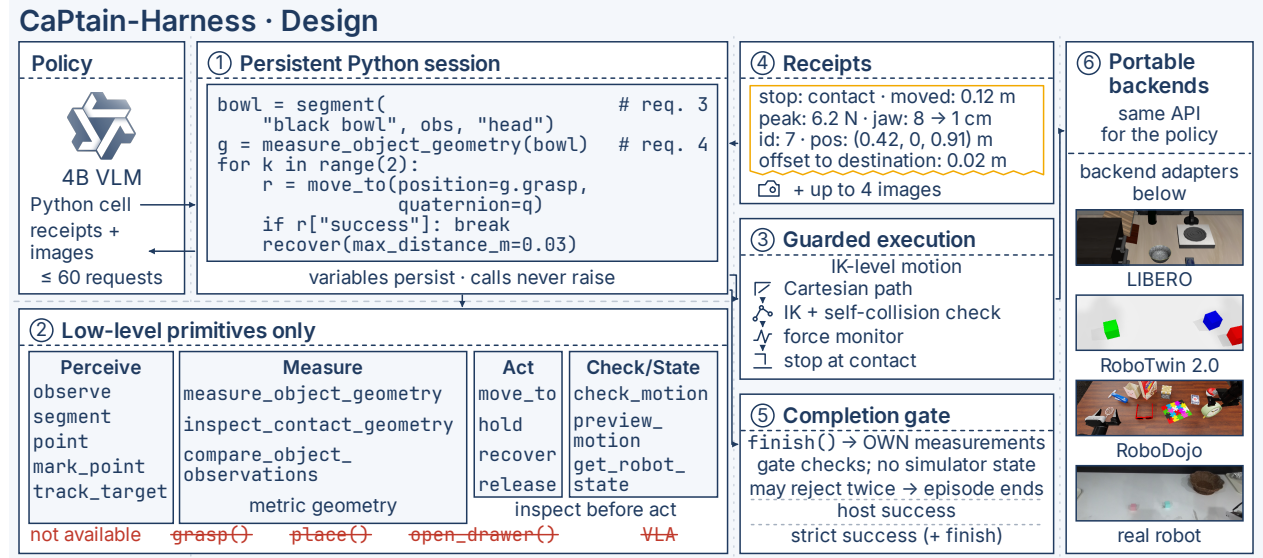


Figure 7 Design of CaPtain-Harness. The policy writes Python into a persistent session ① that exposes only low-level primitives ②; motions are executed with IK and guarded contact ③, every call returns a receipt ④, a completion gate checks the agent’s own measurements ⑤, and environment-specific backends sit below the API ⑥. Receipt values are illustrative.

Policy. The policy π_θ is Qwen3.5-4B [59]: 32 layers with hybrid Gated-DeltaNet and full attention, plus a 24-layer vision tower. It is served with vLLM [34] and sampled at temperature 1 with top- p 1, in training and in evaluation. Each *request* is one chat completion of at most 4,096 tokens with thinking enabled. The request uses the same chat template (system prompt, developer messages with `TASK.md`, tool schema) as the SFT trajectories, so training data and policy requests have identical format. The completion may contain calls to a single tool, `python`. The context window is 65,536 tokens, and older turns are compacted beyond 60k tokens.

The `python` tool and receipts. The `python` tool executes code inside a persistent, sandboxed Python 3.11 session (`numpy`, `scipy`, `Pillow`) with a pre-imported robot client module, allowing variables and helper functions to persist across requests. Robot calls never raise exceptions; instead, each returns a structured dictionary {`success`, `executed`, `failure_category`, `message`, `value`, `diagnostics`, `budget`} alongside a concise (≤ 300 -character) text *receipt* appended to the cell output. Motion receipts report the stop reason, executed distance, peak contact force, and jaw opening before and after execution, whereas perception receipts report the detection identifier, metric position, and object-to-destination offset. The policy may attach up to four camera images per cell, with the conversation history retaining the twelve most recent images.

Robot API. The API has 22 capabilities plus `finish()` (full list in Appendix B). They fall into five groups:

- **Perception:** `observe` (head and wrist RGB-D at 512×512), `segment` (SAM 3 [6]), `point` (Molmo2-8B [14]), `mark_point` (back-projects a pixel the model chose itself), `select_detection` and `track_target`.
- **Measurement:** `measure_object_geometry`, `measure_image_points`, `inspect_contact_geometry`, `inspect_release_geometry` and `compare_object_observations`.
- **Motion:** `move_to`, `hold`, `recover` and `release`. `move_to` takes either an absolute end-effector pose or a single jaw command.
- **Planning:** the read-only checks `check_motion` and `preview_motion`.
- **State:** `get_robot_state`, `gripper_state`, `robot_geometry`, `read_motion_trajectory` and `transform_pose`.

There are no grasp or place macros. The policy must choose poses, approach directions and jaw commands itself. Motions are straight Cartesian sweeps executed by an operational-space controller. Damped least-squares IK and self-collision checks run before execution, and per-substep force monitoring applies a guarded stop at first contact or tolerates load in `manipulation` mode. `TASK.md` tells the policy the instruction, the API, the coordinate frames and the standards by which the final scene is judged. A host-side completion gate checks only the agent’s *own* recorded measurements, never simulator state. It can reject a `finish()` twice; a third call is accepted.

Budgets and success. An episode may use at most 60 requests, 1,080 s of wall time and 150 physical actions (120 during RL). *Host success* is the benchmark’s goal predicate evaluated on the final state. *Strict success* additionally requires that the agent ended the episode by calling `finish()`. A *false finish* is a `finish()` on a failed state. Strict success is the deployable metric: a robot that never declares completion cannot be used.

A.2 RL

Privileged state. Before each request t , the environment server records a hidden snapshot that the policy cannot access. From this mark we compute 36 features $x(s_t)$:

- end-effector position and jaw state;
- goal bookkeeping: number of goals, number and fraction satisfied, active goal, and whether that goal involves a joint (articulated part) or contact;
- distances: target to goal (raw and normalized), end-effector to target, and target to destination;
- target lift and displacement, maximum displacement of any object, and the nearest-other-object distance;
- joint position and change for articulated goals, plus *held* and *ever-held* flags.

Counters of the policy’s own behavior, such as the number of perception calls or failed calls, are excluded so that elapsed time does not leak into the value estimate (§4.5).

Critic. The critic $V_\phi(s, t) = \sigma(\frac{1}{5} \sum_{k=1}^5 f_{\phi_k}([x(s); t; \text{onehot}(\text{task})]))$ is an ensemble of five MLPs (three hidden layers of 256 units) whose logits are averaged. It predicts the probability of host success from the current state. We include the request index t explicitly but hold it fixed inside each difference (below), so elapsed time does not leak into the credit. The critic is warm-started on 1,000 episodes of an earlier policy (45k request states, held-out AUC 0.907, Brier 0.075). It is then *co-trained* online as in SAO: after each round of 256 episodes it takes two AdamW steps of binary cross-entropy against host success over all request states of that round. Each round’s advantages use the critic *before* that round’s update.

Advantage. For episode i with requests $t = 1, \dots, T$, we define:

$$\delta_t = c_t + V_\phi(s_{t+1}, t) - V_\phi(s_t, t) \quad (t < T), \quad \delta_T = c_T + R_i - V_\phi(s_T, T), \quad (9)$$

and $A_t^{\text{GAE}} = \sum_k \lambda^k \delta_{t+k}$ with $\lambda = 0.9$ and no discount [62]. Each start state is rolled out $G=4$ times. We add a group-relative outcome term:

$$A_{i,t} = (G_i - \overline{G}_{\text{group}}) + (A_{i,t}^{\text{GAE}} - \overline{A_{i,\cdot}^{\text{GAE}}}), \quad G_i = R_i + \sum_t c_t, \quad (10)$$

and divide by a running estimate of the advantage standard deviation, without re-centering. All tokens of request t share the advantage $A_{i,t}$. Holding t fixed inside each difference prevents elapsed time from leaking into δ_t , while online co-training keeps the value estimate calibrated to the improving policy. Physical state changes only through physical actions, so the one-step term $V_\phi(s_{t+1}, t) - V_\phi(s_t, t)$ is near zero for requests that only perceive, regardless of how many tokens they contain. Such requests still receive credit through GAE from the physical actions they enable, and through the group outcome term.

Returns and costs. The terminal return R_i is:

- 1 for host success with `finish()`;
- 0.25 for success without `finish()` (a missed finish);
- -0.3 for a false finish;

- -0.1 when a budget is exhausted;
- 0 for any other failure.

The immediate costs $c_t \leq 0$ penalize:

- physical actions beyond $1.5\times$ the median action count of the task’s SFT episodes (-0.008 each);
- failed or rejected physical actions (-0.02 each);
- actions and requests after the host first judges the scene solved (capped at -0.45 in total).

The post-success cost pushes the policy to declare completion; without it, the missed-finish rate rose to about 20% (Appendix E). For the eight hardest tasks (four from libero_10 and four from libero_goal) we add a staged one-shot subgoal reward. A failed episode that reaches rung d of an n -rung ladder (grasp, lift, place, joint opened, ...) receives $0.15 d/(n+1) \leq 0.1125$. Rungs are judged from hidden snapshots; place rungs and rungs for moving an object toward its goal must still hold at the end. The resulting ordering is: success > missed finish > partial progress > 0 > budget exhaustion > false finish.

Policy update. Following SAO [25], we use single-pass, streaming RL with double-sided token masking (unlike SAO, with groups of $G=4$ rollouts per start). A token is kept only if $0.2 < \pi_\theta/\pi_{\text{rollout}} < 4.0$; it is then weighted by that ratio. We also apply a k3 KL penalty to the SFT model ($\beta=0.002$) on kept tokens. At most 8 requests per trajectory are trained, sampled uniformly and inclusion-weighted; the critic still uses all requests. Gradients are accumulated over $K=4$ rounds of 256 episodes (64 start groups $\times 4$), i.e. 1,024 episodes per optimizer step, matching the empirical gradient-noise scale (Appendix H). We use AdamW with a constant learning rate of 4.24×10^{-6} after warm-up, full fine-tuning with FSDP2, and at most two policy versions of staleness.

Training distribution. RL samples from a pool of 7,360 fresh start states over all 40 LIBERO tasks (spatial, object, goal, long), of which 649 libero_object starts with a degenerate goal region are never drawn:

- 5,600 wide-layout starts with light, camera and material perturbations;
- 1,760 official initial states.

The 240 official states used for evaluation (indices 11–16 of each task) are never drawn. To save rollout time, the critic also stops rollouts early: an episode is ended when $V_\phi < 0.02$ for five consecutive requests, except in a 5% control slice and on the hard tasks. A cut episode settles as budget-exhausted ($R=-0.1$), so the critic also shapes returns. A held-out monitor stream (every tenth episode) receives no gradient.

Infrastructure and compute. The trainer occupies 8 H100s. A second 8-GPU node serves 6 vLLM policy servers, SAM 3 and Molmo, and renders 64–80 concurrent MuJoCo episodes. Weights are synchronized to vLLM by NCCL in under 1 s. A round takes 812–1,053 s on 16 GPUs. Per request, generation takes about 2.6 s of the 6.7 s wall time, and the environment, tools and perception take the rest. The main checkpoint, CaPtain-4B, has seen 300 rounds, i.e. 75 optimizer updates on 76,800 episodes, at about 1,290 H100-hours (derived from wall time). The last 8 rounds ran on a single 8-GPU node (trainer world size 4, resharded; 1,551–1,626 s per round) with the RL hyperparameters unchanged.

B Robot API

Table 4 lists the full CaPtain-Harness API used in all experiments: the 22 capabilities and `finish()`, each with a one-line description. Physical actions count against the action budget; all other calls are free.

C SFT details

Data generation. The synthetic episodes were generated by frontier coding agents (GPT-6 Astra and GPT-6.x Sol, see the table caption) acting in the same harness, with the same API, `TASK.md` and chat template as the 4B policy. The raw episodes contained 226 distinct variants of the same “preview, then move” helper. We rewrote all trajectories onto a single helper library of four functions (`val`, `need`, `downq`, `go`) that the

Table 4 The 22 capabilities plus `finish()`. Physical actions (counted against the action budget) are marked ●.

Capability	Semantics
<code>observe</code>	head + wrist RGB-D (512 ²), calibration, robot state
<code>segment</code>	SAM 3 text segmentation (Molmo point fallback) → depth-backed detections
<code>select_detection</code>	bind a chosen detection to identity evidence (read-only)
<code>point</code>	Molmo pointing → deprojected, point-segmented evidence
<code>mark_point</code>	validate and back-project a model-chosen pixel → detection id
<code>track_target</code>	optical-flow tracking of a detection into a newer observation
<code>read_motion_trajectory</code>	page through the measured pose/joint/force trace of a motion
<code>get_robot_state, gripper_state</code>	measured EEf pose, joints, aperture, wrist force
<code>move_to●</code>	absolute EEf pose <i>or</i> one jaw command; guarded / manipulation contact mode
<code>hold●, recover●, release●</code>	keep pose; bounded reversal along the last trace; bounded opening under load
<code>preview_motion, check_motion</code>	read-only timing; IK, self-collision and surface check
<code>measure_image_points, transform_pose</code>	metric points at pixels; pose arithmetic
<code>robot_geometry</code>	EEf/pad reference, pad envelope, gripper limits
<code>inspect_contact_geometry</code>	project the finger-pad closing volume onto the target’s RGB-D
<code>inspect_release_geometry</code>	signed footprint clearance at a proposed release
<code>compare_object_observations</code>	target motion in robot frame across time
<code>measure_object_geometry</code>	symmetry-completed center, extents, yaw, object-to-destination offset; part motion
<code>finish(note)</code>	end the episode; the host scores the final state (no verdict returned)

Table 5 CaPtain-4B-SFT data mix (25,961 rows, 861M tokens). Generating agents: GPT-6 Astra (654M tokens), GPT-6.1 Sol (63M), GPT-6 Sol (14M); RL policy (52M).

Source	Rows	Tokens	Share
Broad-coverage trajectories (code-normalized, filtered)	7,903	348.9M	40.5%
Hard-task targeted trajectories (Round 2)	5,090	238.4M	27.7%
Hard-task targeted trajectories (Round 1)	2,138	99.3M	11.5%
Hard-task trajectories (60-request cap)	925	43.4M	5.0%
Self-generated RL rollouts (successful episodes)	1,386 (267 episodes)	52.3M	6.1%
Visual QA (loss weight 0.5)	8,519	78.8M	9.2%

model learns to write itself. Equivalence of each rewrite was checked call site by call site against a mock robot, and replaying 6 rewritten episodes changed robot commands by less than 1 cm. Turns with repeated execution failures or recovery loops (about 3.6% of requests) are masked from the loss, as are requests after a gate-rejected finish on one long-horizon task.

Training. Only assistant turns are supervised. A text-hint mask replaces earlier `mark_point` pixel coordinates in the history with `<masked>` with probability 0.2. This forces the model to look at the image instead of copying coordinates. Training uses full fine-tuning of the language model with the vision tower frozen. The batch is 16 packed rows of at most 66k tokens. We use DeepSpeed ZeRO-3 in bf16, gradient clipping at 1.0, and 2,435 steps (1.5 epochs). The final model scores 42/80 on our 80-episode validation set.

D Diversity of tasks and layouts during post-training

Diversity of tasks and scene layouts in the post-training data is essential for keeping the visual grounding of the base model. In an earlier version of our data, most episodes started from the official LIBERO initial states, in which each object occupies nearly the same position across episodes (per-object position standard deviation about 1.4 cm). Models fine-tuned on these data learned to reproduce object positions from the task identity

rather than reading them from the image: replacing the input image left the likelihood of the predicted pixel coordinates unchanged. The policy had effectively memorized the layouts and lost the vision-language model’s ability to ground objects in new scenes. We therefore collect SFT data on wide-layout starts, whose per-object position standard deviation is about 15 cm, mix all task suites, and randomize lighting, camera pose and materials during RL. With this diversity, the model grounds every `mark_point` and `segment` call in the current image, which is also what allows it to transfer to the unseen objects and layouts of the real-robot tasks (§4.4).

E Reward, costs and early stopping

The post-success cost (every physical action at -0.008 plus -0.005 per request after the host first judges success, capped at -0.45) was introduced after an earlier run in which missed finishes rose to about 20% per round. The action costs were introduced after RL inflated the number of `move_to` calls per request: from 0.69 to 1.05 in successful episodes and from 0.78 to 1.39 in failed ones. The request cap did not constrain the number of actions packed into one request. The staged subgoal reward is a one-shot, monotone ladder judged from hidden snapshots.

- It applies to 8 hard tasks (libero_10 3/6/8/9; goal 0/3/5/7), from round 9 onward.
- A grasp rung requires the object lifted by at least 5 cm with the end-effector within 13 cm.
- A joint rung requires the articulated part to cover at least 20% of its initial distance to the goal position.

The critic also stops rollouts early, for speed.

- An episode is cut when $V < 0.02$ for 5 consecutive requests after request 5. A deterministic 5% control slice is never cut.
- An episode is stopped and settled as a missed finish once all goals have held for 15 consecutive requests without `finish()`.

F RL infrastructure

Layout. Rollout and training run on separate 8-GPU nodes. The rollout node serves six vLLM policy replicas, SAM 3 and Molmo, and renders 64–80 concurrent MuJoCo episodes; the trainer uses FSDP2 on the other node. Rollouts are streamed: the environment server keeps generating while the trainer updates, and a request may come from a policy that is at most two versions old. New weights reach the rollout servers by NCCL in under one second, and checkpoints are written asynchronously, which reduces the blocking time of a checkpoint from about 500 s to 5–9 s.

Speed-ups. Table 6 lists the optimizations we kept. Together they reduced the time of a training round from 1,812 s to 1,246 s and raised throughput from 520 to 824 episodes per hour, after which training became trainer-bound.

Table 6 RL infrastructure optimizations that passed the numerical gates below.

Optimization	Effect
FlashInfer attention and a chunked-prefill budget of 8,192 tokens in vLLM	+17.6% policy throughput (30.4→35.7 requests per minute per GPU); log-probabilities within the kernel noise floor
Sensor artifacts in shared memory	<code>observe</code> latency 2.2 s→0.16 s
Harness fast loop and lazy rendering	−27% episode wall time, bit-identical physics and responses
Critic-based early stopping (5% control slice never stopped)	24–26% of episodes cut early; no successful episode in the control slice would have been cut
KL reference evaluated every 8 steps	unbiased KL estimate at 1/8 of the reference cost
Pack-level rank assignment	balanced trainer ranks; identical log-probabilities, gradient cosine 0.9999

Numerical gates. Every speed-up had to keep the trained objective unchanged: per-token log-probabilities and batch gradients of the new path had to stay within the difference between two runs of the reference path. FlashAttention-3 was 21% faster per pass but moved log-probabilities beyond this floor, and `torch.compile` and FP8 training failed similar checks; we kept FlashAttention-2 / SDPA in the trainer.

G Lessons from RL on a multi-turn agent

- **Measure the gradient noise before choosing the batch.** With two start states per optimizer step, the weight displacement from the SFT model grew as the square root of the number of steps and consecutive updates were nearly orthogonal: the policy performed a random walk. The gradient-noise scale of outcome-only advantages is in the thousands of episodes (Appendix H), so we accumulate 1,024 episodes per optimizer step.
- **Start states must cover the whole task.** A curriculum that started episodes at intermediate states of successful demonstrations placed every start after the grasp, because a grasp does not change the distance of the object to its goal; the policy then learned only to place held objects. We train from fresh initial states only.
- **Layout diversity keeps the model looking.** Without diverse layouts, the model memorizes object positions and stops grounding its calls in the image (Appendix D).
- **Keep the policy’s behavior out of the critic.** Counters of perception or failed calls act as proxies for elapsed time and move credit from grasps to perception (§4.5).
- **Termination needs its own signal.** A critic trained on task success values every post-success state near one, so the decision to stop receives no credit from the critic; post-success costs and a missed-finish return are needed to make the agent declare completion (Appendix I).

H Gradient-noise analysis

Table 7 Gradient-noise scale B_{noise} (episodes) of advantage estimators on 109 groups $\times$ 16 SFT rollouts (offline; last four layers sketched). Lower is better.

Estimator	B_{noise}
GRPO (0/1 outcome)	9.0k
simulator critic, $\lambda=0$	12.6k
simulator critic, $\lambda=0.9$	6.7k
simulator critic, $\lambda=1$	6.5k

For the 4B model, the simple noise scale at an early checkpoint was about 9,500 episodes (95% lower bound 3,300). This motivated accumulating $K=4$ rounds (1,024 episodes) per update. Without accumulation, consecutive updates had near-zero cosine similarity.

I Host and strict success late in RL

Host and strict success diverge late in training (Figure 8). After about round 340, host success holds at about 65% while strict success falls from about 60% to 44%. The lost episodes run out of requests while re-verifying solved scenes, while false finishes stay rare. Because strict success drops between rounds 300 and 400, we use the round-300 checkpoint as CaPtain-4B.

The mechanism is over-verification. After the task is solved, the policy keeps measuring, re-observing and comparing instead of calling `finish()`. The return structure makes this a slow drift rather than a cliff:

- a missed finish still earns 0.25;
- the post-success cost is capped;
- the critic, trained on host success, assigns $V \approx 1$ to every post-success state.

The critic therefore gives the decision to stop no credit; only the terminal return and the small post-success request cost distinguish it. We observed the same failure in an earlier run, where it doubled missed finishes

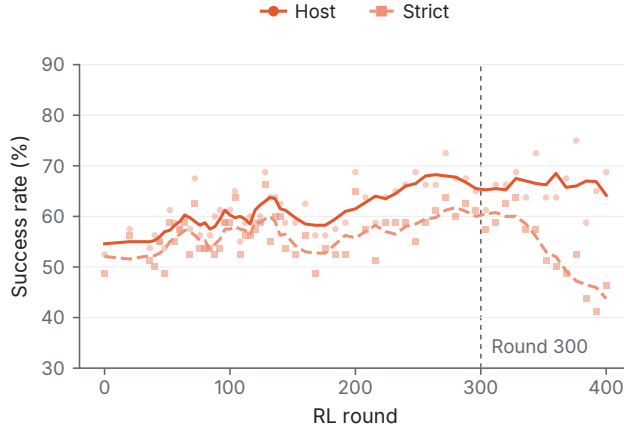


Figure 8 Finish collapse. Evaluation of saved checkpoints during RL: 40 official seed-11 episodes plus 40 wide-layout episodes per checkpoint; dots are raw values and lines a 5-checkpoint moving average. After about round 340, host success stays flat while strict success falls.

Table 8 First blocking failure of failed transfer episodes (counts). Checkpoints: a checkpoint from an earlier RL run and CaPtain-4B-SFT, pooled; RoboTwin (dual Franka) and RoboDojo with earlier adapter versions.

Failure Category	RoboTwin ($n = 437$)	RoboDojo ($n = 46$)
Bimanual coordination required	167	1
Contact / force guard triggered	149	4
Reach / inverse kinematics (IK)	22	21
Perception / viewpoint occlusion	34	8
Request cap / looping	19	—
Grasp failure (empty / slip)	7	6
Post-contact / placement slip	—	6
Other (tracking, false finish, tool misuse)	39	—

and returned strict LIBERO-Plus success to the SFT level. It was partially fixed by the post-success cost. This suggests giving the decision to finish its own target, e.g. success-and-finished.

J Failure taxonomy for zero-shot transfer

We classified the first blocking failure of every failed episode (Table 8). Three classes dominate:

- **Bimanual coordination.** On RoboTwin, tasks that require simultaneous two-arm motion succeed in 0 of 35 instances in every configuration.
- **Embodiment reach.** On RoboDojo, 21 of 46 first blockers are inverse-kinematics or reach failures of the 6-DoF X5 arms.
- **Running out of requests.** On RoboTwin, most episodes end at the request cap, often through re-verification of a solved scene: 7 of 11 host successes in an early evaluation run never called `finish`, and the round-400 checkpoint called it in only 13–15 of 250 episodes. This matches the late-RL drift toward over-verification (Appendix I).

In one test, a training-aligned variant of RoboTwin was not better than the official layout ($p=0.86$). The variant used a LIBERO-like camera, layouts moved into reach and LIBERO brightness. This suggests that viewpoint and lighting are not the main constraint there.

Table 9 LIBERO Held-Out (240 Episodes) Breakdown. Episode counts out of 240 overall, and per-suite Host/Strict counts out of 60.

Model	Host	Strict	Req. Cap	False Finish	Missed Finish	Spatial	Object	Goal	Long
CaPtain-4B-SFT	133	125	96	17	8	47/46	46/45	25/21	15/13
RL, round 128	174	159	68	10	15	54/54	59/57	35/31	26/17
RL, round 292	177	156	80	4	21	56/56	57/51	38/35	26/14
CaPtain-4B (RL, round 300)	177	162	73	5	15	58/58	55/52	41/36	23/16

K Additional LIBERO results

Per-suite paired gains of CaPtain-4B over SFT (host): spatial +18.3 ($p=10^{-3}$), object +15.0 ($p=0.035$), goal +26.7 ($p=1.4\times 10^{-4}$), long +13.3 ($p=0.13$). The libero_10 task 5 predicate has a 3.5mm offset that rejects correct flat placements; every model scores 0/6 on it. Raising the request cap from 60 to 120 on libero_10 did not change success (17 vs. 18/60 for an earlier checkpoint). On LIBERO-PRO Pos+Task, a clean subset of 78 episodes per model uses initial state 11, which was never used in training. It gives the same picture: CaPtain-4B reaches 74.6% (3 suites) and 66.4% (all 4 suites) host success vs. SFT at 57.8% and 47.4%.

Table 10 Exact two-sided McNemar p -values for paired comparisons between CaPtain-4B and CaPtain-4B-SFT across all LIBERO benchmark suites (corresponding to Table 1).

Benchmark	n	Host p -value	Strict p -value
LIBERO	240	4×10^{-8}	7×10^{-6}
LIBERO-PRO (3 suites)	600	1×10^{-20}	2×10^{-14}
LIBERO-PRO (All 4 suites)	800	8×10^{-28}	1×10^{-16}
LIBERO-Plus	700	8×10^{-21}	5×10^{-15}
LIBERO-PRO (4 categories)	160	4×10^{-7}	3×10^{-3}
LIBERO-MAX Dynamic	80	2×10^{-3}	0.15
LIBERO-MAX Base	80	2×10^{-3}	4×10^{-3}

Baseline protocol notes (Tables 2 and 3). In Table 2 (LIBERO-PRO Pos+Task): OpenRUA [13] uses latched success and a wider turn budget (100–500 turns); Harness VLA [94], K1 [40], and RATs [88] use test-time oracle signals, play experience, or memory collected on other seeds of the evaluated tasks; WAA [93] and SPARK [23] use privileged simulator geometry or ground-truth depth; ASPIRE [50], SPARK, CaP-Agent0 [20], and $\pi_{0.5}$ [97] evaluate 50 trials per task; SPARK and RACaP [39] run at temperature $T=0$; K1 and RACaP evaluate 30 episodes per cell; and AgenticRoboRL [48] uses a 100–500 turn budget (its 6-cell average is recomputed from the paper’s per-cell numbers). In Table 3 (zero-shot transfer): on RoboTwin 2.0, PyRUA-Lean Coder-4B [67] evaluates 250 randomized scenes with latched success on native Aloha-AgileX hardware using a RoboTwin-trained VLA tool, whereas CaPtain-4B and the VLA/WAM baselines follow the official Hard protocol (100 trials per task across 50 tasks; single-arm VLA numbers are averaged over the 33 single-arm tasks from the official leaderboard [11, 61, 76]); on RoboDojo [60], baselines run on official layouts while CaPtain-4B is evaluated on 15 layouts per task across all 42 tasks (single-arm GPT-6 Astra is averaged over the 32 single-arm tasks, reaching 31.9 Score / 26.8% SR); on SimplerEnv, PhysBrain 1.0 and $\pi_{0.5}$ run under the official VLA step limit [41]; and on CaP-Bench, CaP-X (multi-turn) corresponds to the M2 visual-feedback setting while CaP-RL and CaP-X (single-turn) run in the open-loop S2 setting [20].

L Harness adaptation per transfer benchmark

Table 11 lists the backend changes per benchmark and marks each as generic, i.e. applicable to any benchmark, or specific to one benchmark or embodiment.

Table 11 Backend changes per transfer benchmark (*Scope*: Generic applies to any benchmark; Benchmark-specific applies to a single benchmark or embodiment).

Benchmark	Change	Scope
RoboTwin 2.0	Optional arm argument; a call without it goes to the arm nearer the target	Generic
	Contact detection recalibrated for the Franka gripper	Generic
	Tilted re-approach when IK fails; gripper closes on contact; grasp check reported in the receipt	Generic
RoboDojo	Automatic hand-over for tasks that pass an object between arms	Specific
	Success check that does not end the episode (the official check is final)	Generic
	Simulated depth and camera calibration as perception inputs	Specific
	Wrist force estimated from the last arm joint (the X5 has no wrist sensor)	Specific
	Speed limits and IK initialization for the X5 arm	Specific
	Return to the home pose before the official check; automatic grasp retry; a finish instruction in TASK.md	Specific
SimplerEnv	Arm joint targets through the robot’s own drives instead of the delta end-effector controller; rendered depth and camera parameters, and an added wrist camera, as perception inputs	Specific
CaP-Bench	CaP-Gym tasks rebuilt in robosuite 1.4; agent camera at CaP-Gym’s pose; wrist flip disabled	Specific

M Compute accounting

- **RL rounds.** A round takes 812s before the staged reward and 991–1,053s after it on 16 H100s; on the 8-GPU layout it takes 1,551–1,626s.
- **Rollout supply.** 873–1,135 episodes per hour.
- **Per request.** About 6.7s, of which 2.6s is in vLLM (0.3s prefill, 2.1s decode).
- **Trainer.** About 0.53s per packed sequence and 1,269–1,507 sequences per round. The time splits into 24% forward, 72% backward and 3% reference forward. FlashAttention-3 is $0.79\times$ the time of FA2 on 45–51k-token samples.
- **Weight sync and checkpoints.** Weights are synchronized to all vLLM servers over NCCL in 0.8s, and exporting a checkpoint takes 25s.
- **Totals.** About 1,290 H100-hours for RL to CaPtain-4B (300 rounds) and 120 H100-hours for SFT, both derived from wall time $\times$ GPUs and including restarts. The critic warm start takes about 16 GPU-hours of rollouts.

N Example episode

Excerpt from a strict-success CaPtain-4B episode: *“pick up the black bowl between the plate and the ramekin and place it on the plate”* (libero_spatial task 0, official state 14; 14 requests in total). Code is verbatim, with long lines and printing trimmed ([...]). go is the policy’s own helper (§3.2). Receipts are shortened.

```
# request 6 (after grasp): lift, re-observe, re-identify bowl and plate by clicking pixels, measure
lift=go(np.array(obs2["value"]["robot_state"]["end_effector_position_m"])+[0,0,.12], q,
    speed=0.035, mode="manipulation")
obs3=robot.observe(); image(obs3["value"]["images"])
b2=robot.mark_point(obs3["value"]["observation_handle"],"agentview",.75,.421,label="held black bowl")
p2=robot.mark_point(obs3["value"]["observation_handle"],"agentview",.78,.725,label="plate")
g2=robot.measure_object_geometry(b2["value"]["detection_id"],p2["value"]["detection_id"])
>>> go: True none bounded Cartesian sweep reached its endpoint [...]
# request 7: carry the bowl by the measured object-to-destination offset
transfer=np.array(obs3["value"]["robot_state"]["end_effector_position_m"])
transfer[:2]-=np.array(g2["value"]["object_to_destination_xy_offset_m"])
trans=go(transfer, q, mode="manipulation")
>>> move_to: ok; stopped_reason=target_reached executed_distance_m=0.1681 max_observed_force_n=5.198
# requests 8-10: head view is occluded by the held bowl -> re-measure from the wrist camera
p3w=robot.mark_point(obs4["value"]["observation_handle"],"robot0_eye_in_hand",.53,.58,label="plate")
b3w=robot.mark_point(obs4["value"]["observation_handle"],"robot0_eye_in_hand",.80,.62,label="black bowl")
g3w=robot.measure_object_geometry(b3w["value"]["detection_id"],p3w["value"]["detection_id"])
# request 11: open, retreat, settle, leave a note for itself
opened=robot.move_to(gripper="open")
retreat=go(release_pose+np.array([0,0,.14]), q, speed=0.035); settled=robot.hold(duration_s=1.0)
>>> move_to: ok; gripper_open_fraction=0.04354->0.9904 ... hold: ok
```

```
# request 13: fresh final measurement
pf2=robot.mark_point(obs6["value"]["observation_handle"],"agentview",.70,.74,label="plate",
granularity="object")
final2=robot.measure_object_geometry(bf["value"]["detection_id"],pf2["value"]["detection_id"])
>>> {'object_to_destination_xy_offset_m': [0.0012, 0.0170], 'xy_distance_m': 0.0171,
'resting_height_consistent': True}
"The bowl is resting on the plate. Fresh measurements show a center offset of about 1.7 cm,
within the required 2 cm tolerance." -> robot.finish(...) [host: success, agent_finished]
```

The episode shows three behaviors of the RL policy. It grounds objects by clicking pixels and back-projecting them rather than calling text segmentation. When the held object occludes the head camera, it switches to the wrist camera. It verifies the final relation with a fresh measurement against the `TASK.md` tolerance before calling `finish()`.

O Real-robot episode traces

Tables 12 and 13 list, request by request, the two real-robot episodes of Figure 5, reconstructed from the CaPtain-Harness call logs of CaPtain-4B.

Table 12 Real-robot trace (a). Instruction in `TASK.md`: “put the yellow cube in the wooden tray”. Times are seconds from episode start in the log and include model inference; the video in Figure 5 is shortened, so its time stamps differ.

Request	Time (s)	Action	Outcome
1	7.7	<code>observe()</code> , <code>robot_geometry()</code>	Head and wrist images; cube lies just outside the tray’s right edge
2	11.0–15.5	<code>mark_point</code> on cube and tray in head image; <code>measure_object_geometry</code>	Cube mask leaks into background (height 0.7 mm); not used
3–5	19.0–27.2	<code>segment("yellow cube")</code> , <code>segment("wooden tray")</code> ; <code>select_detection</code> bound to clicked points; re-measure	First binding rejected (point outside mask); after a new click, cube 3.0 cm tall and tray interior clear
6	31.7	<code>inspect_contact_geometry</code> , <code>preview_motion</code>	Top-down grasp with cube between the jaws
7	34.9–43.2	<code>move_to</code> 11 cm above the cube; <code>observe</code>	<code>target_reached</code> ; wrist view shows cube below gripper
8	45.9–52.2	<code>move_to</code> down at 2.5 cm/s; <code>observe</code>	<code>target_reached</code> , no contact stop
9	54.2	<code>move_to(gripper="close")</code>	Open fraction 1.00→0.33; cube between fingers
10	60.2–65.0	Lift 11.5 cm; <code>observe</code>	Cube rises with gripper; table spot empty
11	68.5–73.4	<code>move_to</code> above tray center	12 cm transfer, <code>target_reached</code>
12–16	76.3–98.7	<code>inspect_release_geometry</code> ; re-select cube (wrist) and tray (head); measure	First query rejected as stale; release projection and occluded masks judged unreliable
17	101.3–105.5	Raise 6 cm for clearance; <code>observe</code>	Cube still held
18	111.9–114.5	Measure held cube in wrist view	Visible face about 3.3 cm above table, consistent with a held cube
19	117.4–127.7	Lower to about 1 cm above tray floor at 2 cm/s	17 cm descent, <code>target_reached</code>
20	131.4	<code>move_to(gripper="open")</code>	Open fraction 0.33→1.00; cube in tray
21	136.3–144.4	Retreat 17 cm; <code>hold(1 s)</code> ; <code>observe</code>	Gripper clear of tray
22	148.7–152.5	Fresh <code>mark_point</code> on cube and tray; <code>measure_object_geometry</code>	Center inside tray footprint, resting height consistent, offset (2.0, −2.1) cm
23	157.2	<code>finish()</code>	Accepted; episode ends

P Zero-shot transfer episodes

We show one successful CaPtain-4B episode from each of RoboTwin 2.0 and RoboDojo (Section 4.3). In both, the benchmark scored the episode as a success and CaPtain-4B ended it by calling `finish()`. The frames are the head-camera images that CaPtain-4B requested during the episode. Each label gives the model request

Table 13 Real-robot trace (b). Instruction in TASK.md: “Put the cube that has the color of the sky into the container that is not round.” Times as in Table 12.

Request	Time (s)	Action	Outcome
1	7.8	<code>observe()</code> , <code>robot_geometry()</code>	Head and wrist images
2	10.4–14.0	Reasoning: the pale-blue cube matches the sky; the brown tray is rectangular and the white bowl is round, so it is excluded. <code>mark_point</code> on both; <code>measure_object_geometry</code>	Cube 2.7 cm tall; tray center 11 cm away
3	18.1–18.2	Re-measure with <code>shape="box"</code> ; <code>inspect_contact_geometry</code>	Cube footprint 3.4×3.5 cm; centered top-down grasp
4	22.4–35.7	<code>move_to</code> 12 cm above cube; guarded descent at 2.5 cm/s; <code>observe</code>	Approach <code>target_reached</code> ; descent stops (<code>motion_stalled</code>) about 4 mm above target with no force; cube between open jaws
5–7	39.8–56.6	Re-select cube (stale detection rejected; wrist camera has no depth, so head camera); <code>inspect_contact_geometry</code> ; close; lift 7 cm	Open fraction 1.00→0.33; cube follows gripper
8–13	60.6–86.3	Measure held cube and tray; raise 10 cm more	Occluded masks of the held cube rejected; cube still held
14	88.6–93.2	<code>move_to</code> above tray center	12 cm transfer, <code>target_reached</code>
15	95.4–104.7	Lower toward tray floor at 2 cm/s	Stops (<code>motion_stalled</code>) about 1 cm above endpoint, no force; cube over tray floor
16	110.5–118.8	<code>move_to(gripper="open")</code> ; retreat 15 cm; <code>observe</code>	Open fraction 0.33→1.00; cube resting in tray
17–27	125.1–172.2	Fresh final measurement: <code>mark_point</code> , <code>segment("floor of the brown rectangular tray")</code> , <code>select_detection</code>	Several tray masks rejected (table pixels, point–mask mismatch); final: center inside footprint and volume, resting height consistent, offset (−1.4, 3.6) cm
28	175.4	Re-measure as box	Same result
29	176.9	<code>finish()</code>	Accepted; episode ends

whose code cell took the image and the time since the episode started. Code is verbatim except for cuts (`#` ...), comments added by us (`#`) and shortened receipts (`>>>`). `go` is the policy’s own helper, which previews a `move_to` and then executes it (§3.2).

RoboTwin 2.0 (dual Franka, randomized scene). Instruction: “Use an arm to place the plastic cup for drinks on the light gray round grainy coaster” (`place_empty_cup`; 7 model requests, 8 physical actions, 77 s).

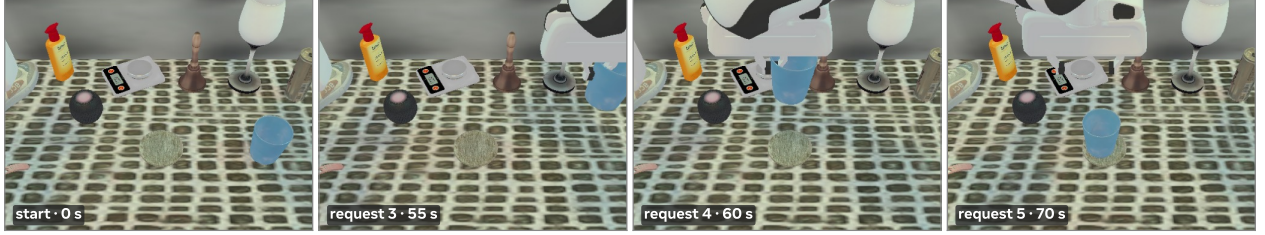


Figure 9 RoboTwin 2.0 episode. The scene starts with distractors around the cup and coaster. CaPtain-4B grasps the cup by its rim, carries it by the measured cup-to-coaster offset, and releases it on the coaster.

```
# request 2: click the cup and the coaster in the head image, measure both
cup = robot.mark_point(obs["value"]["observation_handle"],"agentview",0.86,0.47,
    label="blue plastic cup")
coaster = robot.mark_point(obs["value"]["observation_handle"],"agentview",0.51,0.50,
    label="light gray round grainy coaster")
cid=cup["value"]["detection_id"]; cdid=coaster["value"]["detection_id"]
geom=robot.measure_object_geometry(cid,cdid) # cup: cylinder, radius 4.1 cm
# request 3: grasp the +y rim at rim height, close, lift to 22 cm, observe
q=[0.0,2**-0.5,2**-0.5,0.0]
objg=geom["value"]["object"]; destg=geom["value"]["destination"]
grasp=np.array(objg["center_m"]); grasp[1]+=objg["footprint"]["radius_m"]-0.008
grasp[2]=objg["top_z_m"]-0.023
ic=robot.inspect_contact_geometry(cid,grasp.tolist(),q)
r=go(above, q, speed=0.05)
if r["success"]: r=go(grasp, q, speed=0.025, duration=25)
# ... close, lift to 22 cm, observe
# request 4: carry the cup by the measured cup-to-coaster offset
transfer=lift.copy()
transfer[:2]+=np.array(destg["center_xy_m"])-np.array(objg["center_xy_m"])
r=go(transfer, q, speed=0.06, mode="manipulation")
# requests 5-6: lower to 12 cm, open, retreat; fresh mark_point, measure_object_geometry
>>> object_to_destination_xy_offset_m=[-0.00139, 0.006599]
finish=robot.finish() # request 7, after writing the evidence to NOTES.md
```

CaPtain-4B names both objects from the first image and measures them by clicking pixels and back-projecting them with depth. From the fitted cup radius, it places the grasp on the cup’s rim rather than at its center. After the release it measures again from a fresh image. The offset of 6.7 mm is below the 2 cm tolerance in `TASK.md`, and CaPtain-4B calls `finish()`.

RoboDojo (dual ARX X5, reachable-layout variant). Instruction: “Stack the three blocks with different textures” (`stack_blocks`, layout 1; 20 model requests, 21 physical actions, 413 s).

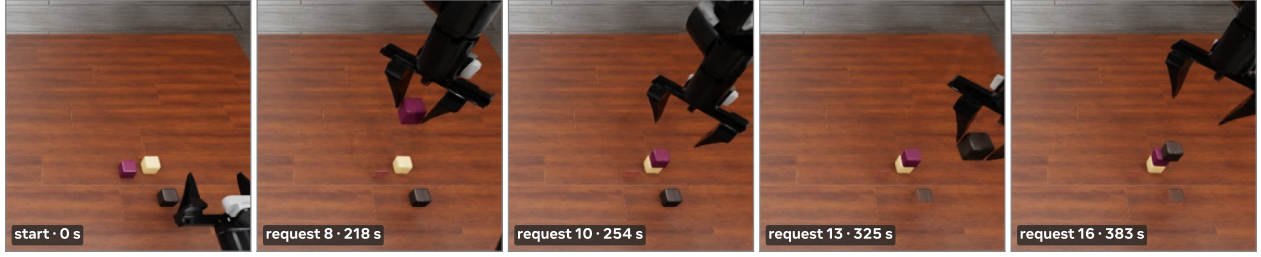


Figure 10 RoboDojo episode (head camera, cropped to the right half of the table). CaPtain-4B lifts the purple block, puts it on the yellow block, then picks up the dark gray block and puts it on top.

```
# request 4: top-down descent to the purple block
p=np.array(objp["value"]["object"]["center_m"]); p[2]=objp["value"]["object"]["top_z_m"]-.008
r=go(p, q, speed=0.04, duration=30, angular=0.4)
>>> move_to: FAILED controller_rejected; emergency_limit; max_observed_force_n=112.1
# request 6: back off 18 mm along the executed path
args=dict(max_distance_m=.018,max_duration_s=8.)
pr=robot.preview_motion("recover",args)
rr=robot.recover(**args) if pr["success"] else pr
# request 7: re-measure, inspect pad contact, close -> jaws close empty
# request 8: reopen, rotate the gripper 45 deg about the vertical, descend slower
r=robot.move_to(gripper="open")
q2=[0,math.cos(np.pi/8),math.sin(np.pi/8),0]
grasp=np.array(pg3["value"]["object"]["center_m"]); grasp[2]=pg3["value"]["object"]["top_z_m"]-.009
r=go(above, q2, speed=0.035, duration=30, angular=0.3)
if r["success"]: r=go(grasp, q2, speed=0.015, duration=30, angular=0.3)
# ... close, lift to 23 cm, observe: block held
# requests 9-10: measure the yellow block, carry over it, lower, open, retreat
transfer=lift.copy();transfer[:2]=dest[:2];transfer[2]=.16
place=transfer.copy();place[2]=.10
# requests 11-14: verify the pair, pick the dark gray block with q2, place it on purple
# requests 15-19: fresh mark_point on all three blocks, pairwise measurements
gd9=robot.measure_object_geometry(d9["value"]["detection_id"],p9["value"]["detection_id"],
                                shape="box",cameras="primary")
fin=robot.finish()
```

The first guarded descent stops 23 mm short of the grasp depth on the arm’s force limit. CaPtain-4B backs off with `recover`, and a closure test at the new height closes on nothing. It then reopens the gripper, rotates it by 45°, descends more slowly, and lifts the block. It repeats this grasp for the dark gray block. Before calling `finish()`, it checks the final stack in fresh images and measures the block tops at 7.0 cm (purple) and 10.5 cm (dark gray), one and two block heights above the 3.5 cm yellow block.

Q Full LIBERO-family comparison

Table 14 collects published results on standard LIBERO, LIBERO-PRO, LIBERO-Plus and LIBERO-MAX for vision-language-action models (VLAs), world-action models (WAMs) and code-as-policy (CaP) agents, next to our models. Most VLAs and WAMs are fine-tuned on LIBERO demonstrations and reach 94–99% on standard LIBERO, but they retain only 20–87% of that success under the perturbations of LIBERO-Plus. CaPtain-4B, trained without any LIBERO demonstration, retains 90%. Baseline numbers are copied from the cited papers; “–” marks benchmarks a method has not reported.

Table 14 LIBERO-family comparison (success %). *Frontier*: a frontier model is queried while the robot acts. *Demos*: trained on LIBERO demonstrations, directly or through a VLA skill. **Bold**: best among methods that use neither a frontier model at test time nor LIBERO demonstrations. *Plus / LIBERO*: LIBERO-Plus success as a percentage of standard LIBERO success, i.e. how much performance is retained under perturbation. Protocol differences are listed below the table.

Method	Family	Frontier	LIBERO	LIBERO	LIBERO-PRO		LIBERO-	LIBERO-MAX		Plus /
		at test	demos		3 Suites	4 Suites	Plus	Base	Dyn.	LIBERO
CaPtain-4B (ours)	CaP	X	X	73.8	78.5	68.6	66.1	70.0	61.2	90
CaPtain-4B-SFT (ours)	CaP	X	X	55.4	59.5	49.2	48.6	51.2	40.0	88
Qwen3.5-4B (base)	CaP	X	X	0.0	0.3	0.2	0.0	0.0	0.0	–
<i>Vision-language-action models</i>										
OpenVLA [30]	VLA	X	✓	76.5	0.0	0.0	15.6	–	–	20
OpenVLA-OFT [31]	VLA	X	✓	97.1	–	–	69.6	64.3	43.2	72
RIPT-VLA [68]	VLA	X	✓	97.5	–	–	68.4	–	–	70
π_0 [2]	VLA	X	✓	94.2	0.0	0.3 [†]	53.6	–	–	57
π_0 -FAST [54]	VLA	X	✓	85.5	–	–	61.6	–	–	72
$\pi_{0.5}$ [55]	VLA	X	✓	96.9	12.8 ^d	10.8	84.4 [†]	79.7	65.7	87
UniVLA [4]	VLA	X	✓	95.2	–	2.5	42.9	–	–	45
GR00T N1.6 / N1.7 ^j [52]	VLA	X	✓	97.0 [†]	–	–	–	69.3	50.0	–
MolmoAct2 [17]	VLA	X	✓	–	18.6 [†]	–	–	80.3	66.9	–
<i>World-action models</i>										
Cosmos-Policy [32]	WAM	X	✓	98.5	–	–	82.2 [†]	77.4	59.3	83
Fast-WAM [84]	WAM	X	✓	97.6 [†]	–	–	51.5 [†]	42.0	24.0	53
GE-Act + Robust-WAM [79]	WAM	X	✓	97.3	–	–	80.9	–	–	83
OpenWAM- α [76]	WAM	X	✓	99.3	–	–	69.2	–	–	70
<i>Code-as-policy agents</i>										
OpenRUA [13]	CaP	✓	X	–	91.8 ^{a,g}	87.0	–	–	–	–
Harness VLA (Opus 4.7) [94]	CaP	✓	✓	96.0	87.7 ^b	82.4	–	–	–	–
AgenticRoboRL [48]	CaP	X	✓	–	81.3 ^{g,h,i}	71.0	72.2	–	–	–
K1 [40]	CaP	✓	X	–	77.2 ^{b,f}	–	–	–	–	–
WAA [93]	CaP	✓	X	–	75.6 ^c	–	–	–	–	–
ASPIRE [50]	CaP	✓	X	–	71.7 ^{b,d}	–	–	–	–	–
SPARK [23]	CaP	X	X	–	66.8 ^{c,d,e}	–	–	–	–	–
RACaP [39]	CaP	✓	X	–	45.0 ^{e,f}	–	–	–	–	–
Guava [45]	CaP	X	X	–	≈ 41	–	–	–	–	–
CaP-Agent0 [20]	CaP	✓	X	–	18.2 ^d	–	–	–	–	–

Protocols. Ours: host success; LIBERO on the held-out official initial states 11–16 (240 episodes); LIBERO-PRO Pos+Task with initial states 1–10, 100 episodes per cell; LIBERO-Plus on a stratified 700-case sample; LIBERO-MAX on the 80 Lite cases with independent Base and Dynamic rollouts. Baselines: LIBERO with 500 episodes per suite (Harness VLA: 100); LIBERO-Plus on the full 10,030 instances; LIBERO-MAX on 8,000 pairs, with Dynamic replaying the Base prefix. ^alatched success; ^btest-time success oracle, or memory or programs built on other seeds of the evaluated tasks; ^cprivileged simulator geometry or ground-truth depth; ^d50 trials per task; ^e $T=0$; ^f30 episodes per cell; ^gbudget of 100–500 turns; ^hsix-cell average recomputed from the per-cell numbers; ⁱplanner trained on the LIBERO-PRO tasks with other initial states; ^jLIBERO for N1.6, LIBERO-MAX for N1.7; [†]evaluated by a later paper: Zhang et al. [94] (π_0 PRO 8c), Wang et al. [76] (LIBERO-Plus of $\pi_{0.5}$, Cosmos-Policy and Fast-WAM; LIBERO of GR00T N1.6 and Fast-WAM), Grant et al. [23] (MolmoAct2).