ROBOT-CENTRIC SCAFFOLDING ADVANCES THE FRONTIER OF VLM-BASED MANIPULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

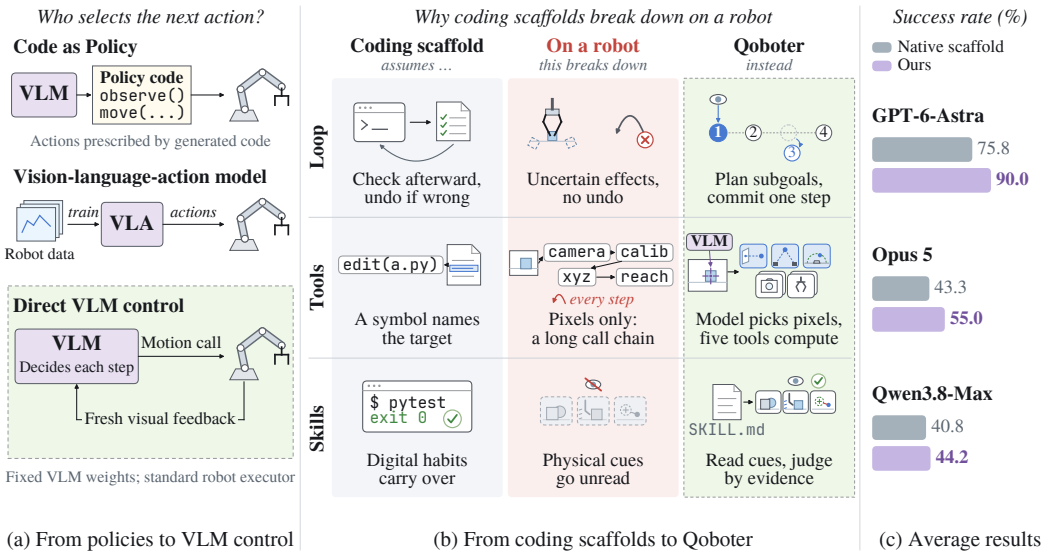
In 2024, as foundation models became able to handle real software tasks, coding scaffolds emerged and greatly accelerated AI coding. General-purpose vision-language models (VLMs) can now turn natural-language goals into useful manipulation without task-specific policies. Existing coding scaffolds complete some robot tasks but fail systematically, because they rely on three assumptions that hold only in digital environments: that actions can be checked afterward and undone, that targets are named exactly by symbols, and that habits formed on digital tasks carry over. We therefore propose Qoboter, a robot-centric scaffold that redesigns its loop, tools, and skills accordingly. Its loop plans task-level subgoals but commits only to the next local action, chosen from the latest observation. Its five core tools let the model select target pixels, then convert them into 3D points, check reachability before execution, and batch frequently co-occurring operations. Its skills guide the model to interpret occlusion, contact, and control error and to judge progress by observed evidence. On four simulators and a real robot, we weigh success against output-token cost. With model weights fixed, Qoboter raises success on every frontier model at a similar token cost, from 75.8% to 90.0% on GPT-6-Astra and from 43.3% to 55.0% on Opus 5. The same configuration transfers unchanged to three open-weight models, raising their success by 5.0 to 8.4 points. These results suggest that scaffolding complements model scaling as a lever for reliable VLM-based manipulation.

1 INTRODUCTION

In coding, what turned foundation models from writing isolated snippets into reliably completing real software tasks was not only stronger models but also the scaffolding built around them. Agents such as Claude Code and Codex wrap the model in an interaction loop, tools, and skills, and the same model performs markedly better once placed in a stronger scaffold (Yang et al., 2024a; Wang et al., 2024c). The same general-purpose foundation models have now reached a similar starting point in the physical world: without a task-specific policy, they can turn a natural-language goal into useful manipulation behavior. If the experience of coding is any guide, turning this nascent physical competence into reliable manipulation will likewise depend on scaffolding.

The most direct candidates are existing coding scaffolds. In our experiments, applying them directly to robots completes some tasks but fails systematically, because their assumptions at three levels largely hold in digital environments yet break down in physical interaction. **At the loop level**, coding scaffolds assume that an action's effects can be inspected afterward and undone if wrong, so they work in complete attempts: write a solution, run it, check the result, and retry if needed. In manipulation, effects are neither fully predictable nor reversible: objects shift, grasps slip, and the VLM's estimate of the scene is itself uncertain. A controller's return status, like an exit code, reports how a motion ended but not whether the object moved as intended, so an action sequence fixed in advance soon rests on unconfirmed states. **At the tool level**, coding tools operate on objects that a symbol specifies exactly, such as a file path or a variable name, so one call can act on the target. In manipulation, the target appears only in images, and reaching it takes a long chain of calls: reading cameras and calibration, computing 3D coordinates, and checking reachability. With only low-level tools, the model rebuilds this chain and re-derives the geometry at every step, an overhead that closed-loop execution incurs throughout the episode. **At the skill level**, coding scaffolds assume

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) From policies to VLM control (b) From coding scaffolds to Qoboter (c) Average results

Figure 1: **Qoboter keeps a fixed VLM in charge of every action and rebuilds the scaffold around it.** (a) Unlike code as policy and VLA models, the VLM selects each motion from fresh feedback. (b) Coding scaffolds' loop, tool, and skill assumptions break down on a robot; **Qoboter** redesigns each. (c) Average success over each model's native scaffold.

that the model's execution habits carry over across domains. Yet even with fresh observations and usable tools, habits formed on digital tasks readily misfire on a robot: the model fails to exploit cues such as occlusion, contact feedback, and control error, relies on outdated observations, or persists with strategies the evidence has refuted, wasting turns and sometimes causing failure.

To address these problems, we propose **Qoboter**, a robot-centric scaffold that reshapes the loop, the tools, and the skills around the model (Figure 1). **At the loop level**, Qoboter separates the planning horizon from the execution commitment: the model outlines a task-level plan of subgoals, which keeps a long-horizon task on course, then commits only to the next local action, chosen from the latest observation, and revises later steps when new evidence requires while keeping the subgoals that still hold. **At the tool level**, Qoboter leaves target selection to the model and delegates computation to tools. Through extensive experiments, we distill a core interface of five tools: general primitives for synchronized observation and motion, shared across all platforms, and geometric tools that turn model-selected pixels into 3D points with diagnostics and check reachability before the arm moves. Batching merges frequently co-occurring operations into single calls, reducing routine round trips. **At the skill level**, Qoboter encodes robot-centric behavioral priors as reusable skills that guide the model to interpret occlusion, contact, and control error and to judge progress by observed evidence; we identify a small set of skills that consistently reduce interaction turns.

We systematically study how scaffold design affects manipulation success and cost. On four simulators (RoboDojo, RoboTwin2.0, RoboLab, and LIBERO) and a real robot, we evaluate every method on a success-cost frontier that weighs single-attempt success against the cumulative output tokens of each episode. With model weights fixed, Qoboter raises success on every frontier model, from 43.3% to 55.0% on Opus 5 and from 75.8% to 90.0% on GPT-6-Astra, at a similar token cost. A control that removes the coding scaffold without adding any robot-centric design is cheaper but less successful, so the gain comes from the design itself. Unchanged, the same configuration raises success by 5.0 to 8.4 points on three open-weight models, a 2.7-fold gain on Qwen3.8-27B. Among the ablations, switching to granular tools causes the largest success drop, up to 18.4 points.

Our contributions are threefold: (1) we trace the failures of coding scaffolds in robotic manipulation to digital-task assumptions at the loop, tool, and skill levels that break down in physical interaction; (2) we propose Qoboter, a robot-centric scaffold that redesigns these three levels for physical interaction; and (3) through controlled experiments, we characterize how scaffold design, tool interfaces, and procedural skills affect manipulation success and cost.

2 Related Work

Foundation models. Instruction-tuned multimodal models (Alayrac et al., 2022; Liu et al., 2023b) are now released open-weight across a wide capacity range (Bai et al., 2023; Wang et al., 2024b; Bai et al., 2025; Chen et al., 2024b; Deitke et al., 2025) and as hosted frontier systems (OpenAI et al., 2023; Gemini Team et al., 2023), and how their competence grows with scale is itself contested (Kaplan et al., 2020; Wei et al., 2022). For physical tasks the gap is specific: models that answer questions well still fail at fine-grained perception (Fu et al., 2024) and need dedicated supervision for metric and relational spatial reasoning (Chen et al., 2024a; Cheng et al., 2024). Specialist segmentation, detection, and depth models (Kirillov et al., 2023; Liu et al., 2024b; Yang et al., 2024b) are the usual remedy, but what they return, a mask, a box, or a relative depth map, is not itself an action target, and whether it helps a model that must still decide where to move is an open question that we test directly. We hold the model fixed and ask what has to be built around it.

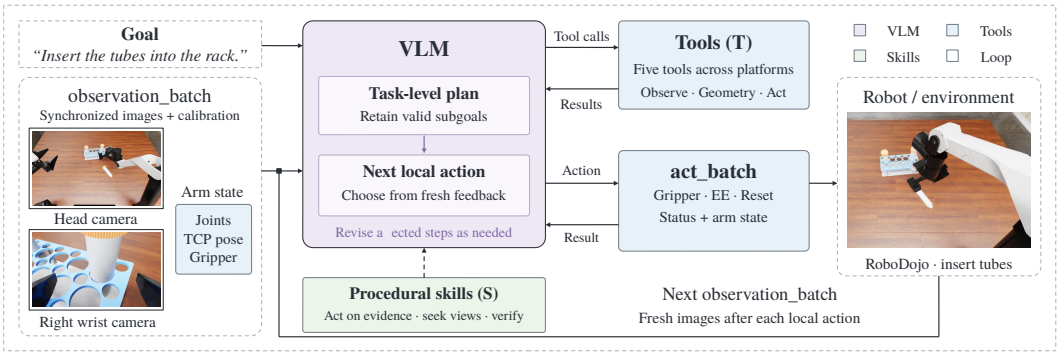
Embodied agents. Foundation models have served as planners over pretrained skills (Ahn et al., 2022; Huang et al., 2022; Driess et al., 2023), as writers of the policy through code, value maps, keypoints, or visual prompts (Liang et al., 2023; Huang et al., 2023; 2024; Nasiriany et al., 2024; Liu et al., 2024a; Yuan et al., 2024), and as end-to-end action models (Brohan et al., 2023; Open X-Embodiment Collaboration et al., 2024; Kim et al., 2024; Black et al., 2024; Gemini Robotics Team et al., 2025; Zawalski et al., 2024), a line that continues through 2026 in generalist robot foundation models (Yuan et al., 2026; Deng et al., 2026; Li et al., 2026a; Cai et al., 2026). Work nearer to ours adds one execution mechanism at a time: carried state (Yoneda et al., 2024), agent-side memory over long horizons (Hu et al., 2026), failure explanation and recovery (Liu et al., 2023c; Duan et al., 2025; Liu et al., 2026), and test-time compute allocation (Dao et al., 2026). Recent 2026 work makes the harness itself the object of study, in manipulation (Chen et al., 2026b; Xu et al., 2026; Ding et al., 2026), in navigation (Lei et al., 2026b; Li et al., 2026b), and in evaluations asking whether today's models hold up past short horizons (Fan et al., 2026); a parallel line fine-tunes VLAs into tools that report progress to the agent (Lei et al., 2026a). We share its premise but train no weights, and instead redesign the loop, the tools, and the skills.

Agent scaffolding. In digital settings the scaffold around a fixed model is an established design surface: reasoning interleaved with acting (Yao et al., 2023), verbal self-revision (Shinn et al., 2023), tool use (Schick et al., 2023), a growing skill library (Wang et al., 2024a), and explicit context management (Packer et al., 2023). Software engineering made the point sharply, where a purpose-built agent-computer interface rather than a new model drove much of the progress (Jimenez et al., 2024; Yang et al., 2024a), and similar harnesses now run web and desktop agents (Zhou et al., 2024; Xie et al., 2024). All of them assume predictable effects, textual observations, and reversible failures, none of which holds for a gripper in contact. Manipulation benchmarks, in turn, compare policies rather than scaffolds (Liu et al., 2023a; Mu et al., 2025; Li et al., 2024) and report success without the tokens and interaction turns an agentic loop spends reaching it, which is the cost we measure.

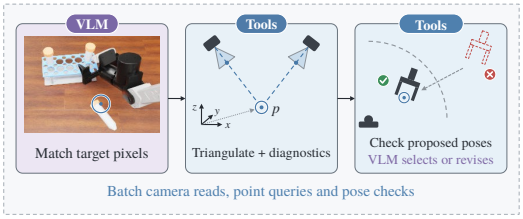
3 Qoboter

Problem formulation. An episode starts from a natural-language instruction g. A VLM π_θ with fixed weights θ acts through a scaffold $\sigma = (L, T, S)$, consisting of an interaction loop L, a tool set T, and procedural skills S. The physical state is only partially observed: an observation $o_t = (\{I_t^k\}_{k \in K}, q_t)$ contains images from the platform's fixed and wrist cameras K and the measured arm state q_t, with camera calibration supplied alongside the images. The initial context c_0 holds g, the initial observation, the tool schemas, and the skills. At turn t, the model emits a tool call $u_t \sim \pi_\theta(\cdot \mid c_t)$ to read an observation, compute geometry, or command motion. The loop dispatches the call and incorporates its result r_t into the next context, $c_{t+1} = L(c_t, u_t, r_t)$. An independent judge determines success, and execution is bounded by a shared episode budget B (Appendix B). At termination turn τ, we record the outcome $y \in \{0, 1\}$ and cost $C = (N_{\text{tok}}, \tau)$, where N_{tok} sums output tokens over all model requests, including recovery. Failed episodes also contribute to the reported cost. With θ fixed, the design objective is to raise $\Pr[y = 1]$ without increasing $\mathbb{E}[C]$, advancing the success-cost frontier.

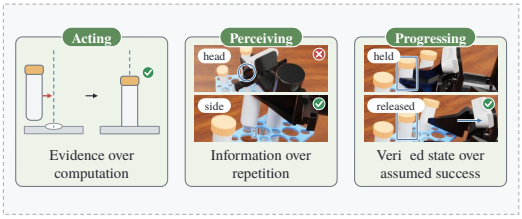
Qoboter addresses this objective at three levels. The loop couples a task-level plan with local actions chosen from updated observations. The tools expose a compact, composable interface to perception,



(a) Task-level planning with closed-loop execution (L)



(b) Compact, composable tools with batching (T)



(c) Procedural skills (S)

Figure 2: **Qoboter overview**, shown on the RoboDojo tube-insertion task. **(a)** The interaction loop L: a fixed VLM retains valid task-level subgoals, chooses the next local action from the latest observation, and revises affected steps as new evidence requires. `act_batch` returns action status and arm state; the next `observation_batch` supplies fresh images. **(b)** The tools T: five core tools shared across platforms provide observations, geometry, and motion primitives, with batching for recurring operations. The VLM selects pixels and proposes TCP poses; tools compute 3D points with diagnostics and check reachability without moving the arm. The VLM uses the results to select or revise its candidates. **(c)** The procedural skills S: evidence over computation, information over repetition, and verified state over assumed success. Images come from a recorded RoboDojo trajectory and illustrate the scene and observations; they do not replay a single step-by-step sequence.

geometry, and robot control, and batch routine operations to reduce repeated model calls. The skills supply behavioral priors for interpreting contact, occlusion, and control error and for using that evidence to guide execution, so that fewer turns are wasted. Figure 2 shows their interaction. For comparison, the Raw API loop uses $\sigma_0 = (L_0, T_0, S_0)$: ordinary tool dispatch, a basic robot API, and interface documentation. The tool and skill ablations vary these components while keeping Qoboter's loop fixed (Section 4.4).

3.1 Task-level planning with closed-loop execution

Coding scaffolds typically iterate over complete attempts: the model generates a solution, executes it, verifies the outcome afterward, and reruns a revised attempt if it fails. This works because a failed attempt costs little: its effects can be inspected afterward and undone. In robotic manipulation, effects are neither fully predictable nor reversible. Object poses may shift during motion, grasps may slip after contact, and occlusions may invalidate earlier observations, while VLM-based state estimation remains inherently uncertain. As a result, an action sequence planned upfront can diverge from the true scene state before it finishes, and a failed attempt leaves the scene changed rather than restoring its initial state. Controller feedback reports how a command ended but not the resulting object pose or grasp state, so it cannot reveal this divergence by itself. Effective manipulation therefore requires continual decision updates from newly acquired observations.

Qoboter addresses this problem through *task-level planning with iterative closed-loop execution*, which separates the planning horizon from the execution commitment. Given a task goal and the initial scene, the model first produces a task-level plan that specifies intended subgoals and their ordering. This plan provides long-horizon task structure, while concrete motion decisions are deferred to execution time. At each step, the model observes the current scene and robot state, invokes geometric queries when needed, and commits only to the next local action. After execution, the model

Table 1: **Qoboter's five core tools.** The interface groups routine reads, geometric computations, and motion commands while leaving target selection and task-level decisions with the VLM. Implementation is in Appendix E.

Tool	Operation exposed to the model
`observation_batch`	Synchronized camera images, calibration, and arm states
`unproject_pixels`	Pixels to calibrated rays or intersections with a plane
`triangulate_batch`	Matched pixels to 3D points with ray-gap diagnostics
`check_tcp_pose_reachable`	Candidate pose feasibility without moving the robot
`act_batch`	Gripper, absolute/relative end-effector motion, joint reset

acquires fresh observations, updates its estimate of the interaction state, and revises subsequent steps where the new evidence requires. Under the batched interface, visual feedback is obtained through `observation_batch`, while `act_batch` executes actions without directly returning images.

This design preserves task-level consistency while enabling adaptive execution under evolving scene states. Failures are handled through observation-driven retries rather than replaying fixed action sequences: the model revises only the affected local step while retaining valid portions of the overall plan. Appendix C.1 provides an example in which a failed release is corrected using updated wrist-camera observations without abandoning the original placement objective.

3.2 Compact, composable tools with batching

As discussed in the introduction, Qoboter leaves target selection to the model and delegates computation to tools, so the model does not rebuild the chain from images to motion at every local step. The tool set has three parts: compact primitives, geometric computations, and batching.

(i) Compact primitives across embodiments. We started from a saturated interface that exposed every available perception, geometry, and control call, and kept the operations that development runs needed again and again across tasks. The result is a core of five tools (Table 1). They are general enough for the model to compose approaches, grasps, placements, and contact motions. Each platform plugs its cameras, arm state, and motion backend into the same conventions (Appendix F).

(ii) Geometry as a reusable computation. The model picks the target pixels, and the tools turn them into metric quantities. `unproject_pixels` maps pixels to calibrated rays or to points on a given plane, and `triangulate_batch` combines matched pixels from several views into 3D points. Both return diagnostics such as the ray gap, which lets the model reject a bad match (Appendix C.2). `check_tcp_pose_reachable` asks the platform whether a candidate pose is feasible without moving the arm. A reachable pose is still only a candidate: it does not guarantee a successful contact or a collision-free motion (Appendix E).

(iii) Batching recurring operations. Development runs also showed calls that recur in groups: reading several cameras and arms, locating several points, or checking several candidate poses. Qoboter turns each group into one request with one result, which saves a model round trip per item. `act_batch` can move through several poses under one gripper command and reports where execution stopped. When a later step depends on the outcome, such as transport after a grasp, the model still observes first. Batching therefore trims routine calls inside a local step without weakening the closed loop.

3.3 Reusable skills for interpreting physical feedback

As discussed in the introduction, execution habits formed on digital tasks misfire on a robot, and the usual cost is wasted turns rather than outright failure. Qoboter supplies robot-centric behavioral priors as skills: short structured documents, in the front-matter format of coding-agent skills, placed in the model's context. They guide how the model acts, how it perceives, and how it judges progress.

(i) Acting: evidence over computation. A computed target is an estimate that motion and observation must confirm. The skills summarize this as *compute less, do more, look more*: use geometry for a coarse target, move in short steps, and compare each commanded motion with the measured result and fresh images. Steps shrink as contact nears, and the final approach is refined in wrist-camera image space, a form of visual servoing (Hutchinson et al., 1996) carried out by the model itself. A

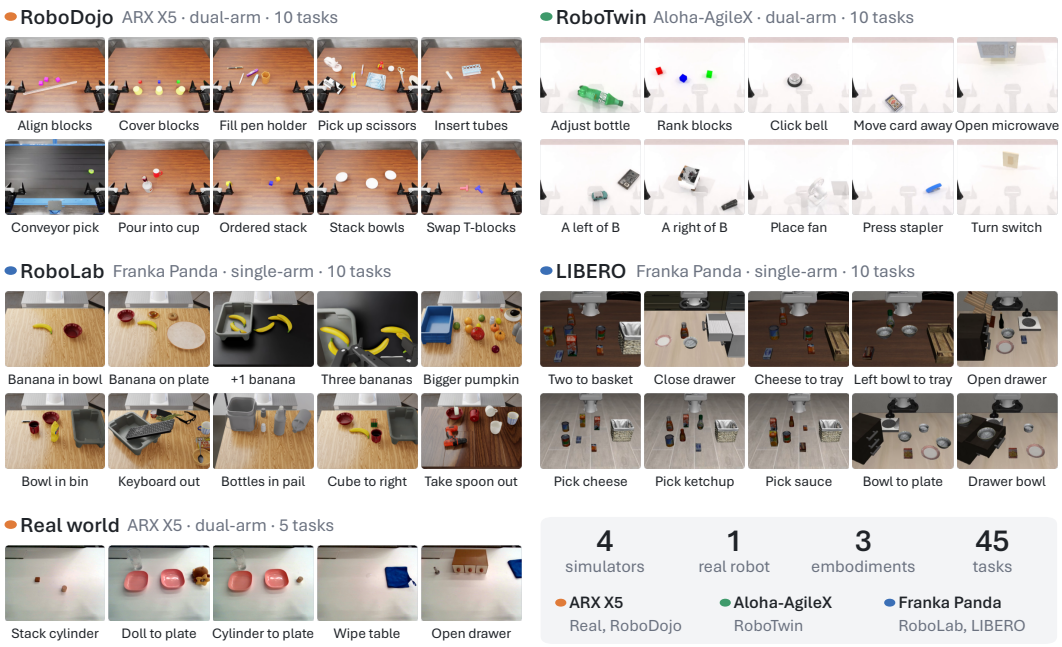


Figure 3: **Evaluation tasks.** Initial scene of every task, taken from the first frame of the primary camera. The benchmark spans four simulators with ten tasks each (RoboDojo, RoboTwin, RoboLab, and LIBERO) and five tasks on a real robot, covering three embodiments: the dual-arm ARX X5 (real robot and RoboDojo), the dual-arm Aloha-AgileX (RoboTwin), and the single-arm Franka Panda (RoboLab and LIBERO).

controller status is read only for what it actually establishes (Appendix D). Unexpected contact, or a persistent gap between commanded and measured positions, is treated as evidence about the scene and shapes the next motion. In Figure 6, the controller reports `converged`, and the model still uses the residual to revise where it presses.

(ii) Perceiving: information over repetition. Zooming into the same occluded or poorly angled frame does not reveal anything new. Before observing, the model names the quantity it is missing and picks a view that can show it: a fixed camera for identity and layout, the wrist camera for lateral alignment, or a side view for height and depth. If no available view resolves the quantity, the model changes the viewpoint, for example by moving the idle arm of a bimanual platform to serve as a side camera, a use of embodiment for active perception (Bajcsy, 1988) (Figure 7). Old images stay valid only while the scene is unchanged; once a motion may have moved the relevant object, the model observes again. Further rules and single-arm alternatives are in Appendix D.

(iii) Progressing: verified state over assumed success. Each subgoal is tied to evidence that it is done. After a close command the model checks the gripper reading and an image, a test lift shows whether the object moves with the hand, and transport starts only from a confirmed grasp. Recovery follows the failure signal: an unreachable pose calls for a new target, unexpected contact calls for rethinking the local geometry, and repeated identical outcomes call for a new diagnosis. The task-level plan then either advances or revises the current step based on what was observed.

4 Experiments

Our experiments answer three questions. **(Q1)** With the model weights fixed, does Qoboter raise success over general-purpose coding scaffolds without raising cost, and does the gain come from its robot-centric design rather than merely from dropping the coding scaffold? We answer it in simulation (Table 2) and on a real robot, whose setup is given in Appendix G and whose results are reported on our anonymous website.[1] **(Q2)** Does the gain transfer, without per-model tuning, to

[1] `https://anonymous.4open.science/w/RoCS-F1CF/`

Table 2: **Full-system scaffold comparison in simulation.** Success rate per simulator, its unweighted average, and output tokens per episode. Small numbers in the **Average** column give the change in points from the coding scaffold of the same block. CaP-X supports single-arm robosuite environments and is evaluated only on LIBERO; its four-simulator averages for success and cost are not applicable. Settings are in Appendix B.

	Success rate (%)					Tokens/ep
Scaffold / config	Dojo (↑)	Twin (↑)	Lab (↑)	LIBERO (↑)	Average (↑)	Output (k) (↓)
GPT-6-Astra						
Codex	43.3	90.0	70.0	100.0	75.8	16.7
Raw API loop	23.3	90.0	53.3	83.3	62.5 (−13.3)	12.4
CaP-X	–	–	–	56.7	–	N/A
Qoboter	80.0	100.0	80.0	100.0	90.0 (+14.2)	14.9
Opus 5						
Claude Code	23.3	63.3	26.7	60	43.3	88.9
Raw API loop	16.7	56.7	33.3	50	39.2 (−4.1)	76.9
CaP-X	–	–	–	33.3	–	N/A
Qoboter	40.0	80.0	36.7	63.3	55.0 (+11.7)	81.9
Qwen3.8-Max						
Claude Code	26.7	73.3	30.0	33.3	40.8	142.9
Raw API loop	23.3	63.3	30.0	36.7	38.3 (−2.5)	138.5
CaP-X	–	–	–	23.3	–	N/A
Qoboter	26.7	80.0	36.7	33.3	44.2 (+3.4)	151.8

open-weight models that took no part in development (Table 3)? **(Q3)** How much of the gain do the tools and the skills each contribute (Table 4)? We first describe the setup shared by all three.

4.1 Experimental setup

All three questions are studied on the same 45 tasks (Figure 3): ten in each of four simulators, RoboDojo (Chen et al., 2026a), RoboTwin (Mu et al., 2025), RoboLab (Yang et al., 2026), and LIBERO (Liu et al., 2023a), and five on a real dual-arm ARX X5. Together they cover dual-arm and single-arm platforms, short and long horizons, and instructions that require grounding language beyond naming an object (Appendix A). Every simulator task is run from three initial scenes, giving 30 episodes per simulator and 120 per configuration, and each real task is run five times. Within a model block only the scaffold differs: weights, tasks, initial states, sensor access, low-level executor, success judge, and budget are shared, and comparisons are made only within a block. Success is judged independently of the agent, never by its own claim, and each episode is a single attempt (pass@1). We report the success rate per simulator and their unweighted average, and cost as output tokens per episode; the ablation also reports interaction turns. Appendix B gives the full protocol.

For the scaffold comparison (Q1), we pair three frontier models with the general-purpose coding scaffold that is their natural default: Codex (OpenAI, 2025) for GPT-6-Astra and Claude Code (Anthropic, 2025) for Opus 5 and Qwen3.8-Max. Each model block also includes a **Raw API loop**, which runs the off levels L_0, T_0, and S_0 of Section 3 in a minimal API loop: it drops the coding scaffold without adding any robot-centric mechanism, and so tells the two sources of a gain apart. As a robot-agent baseline we add CaP-X (Fu et al., 2026), which acts by writing policy code and supports single-arm robosuite environments, limiting our comparison to LIBERO. On the real robot we compare Qoboter with the coding scaffold of each model block (Appendix G). To test transfer (Q2), we freeze the configuration developed with Opus 5 and run it on three open-weight models that took no part in development, DeepSeek-V4.1-Flash, Qwen3.8-Flash-Next, and Qwen3.8-27B, each compared with Claude Code; only mechanical adaptation (tool JSON, image encoding) is allowed.

For the ablation (Q3), we use Qwen3.8-Max and Qwen3.8-Flash-Next, with the loop on throughout. Each variant changes one factor of the complete configuration: *with SAM* adds an open-vocabulary segmentation tool; *without skills* removes the skills; *minimal tools* keeps only `observation_batch` and `act_batch`; and *granular tools* replaces the batched tools with the discrete `get_*` and `act_*` calls of the saturated interface (Section 3.2). Frontier models such as GPT-6-Astra are expensive to run at this scale: even with stable access through the official API

Table 3: **Cross-model generalization of the scaffold.** The same configuration runs without per-model tuning on three open-weight models. Columns follow Table 2; small numbers in the **Average** column give the change in points from Claude Code on the same model. Settings are in Appendix B.

	Success rate (%)					Tokens/ep
Model / config	Dojo (↑)	Twin (↑)	Lab (↑)	LIBERO (↑)	Average (↑)	Output (k) (↓)
DeepSeek-V4.1-Flash (MoE, 552B total, 16B active)						
Claude Code	13.3	66.7	26.7	40.0	36.7	273.3
Qoboter	16.7	83.3	33.3	33.3	41.7 (+5.0)	289.2
Qwen3.8-Flash-Next (MoE, 125B total, 6B active)						
Claude Code	6.7	53.3	10.0	31.7	25.4	319.0
Qoboter	13.3	60.0	33.3	28.3	33.8 (+8.4)	277.1
Qwen3.8-27B (dense, 27B)						
Claude Code	0.0	16.7	3.3	0.0	5.0	371.5
Qoboter	3.3	36.7	0.0	13.3	13.3 (+8.3)	220.7

Table 4: **Tool and skill ablation** on Qwen3.8-Max and Qwen3.8-Flash-Next, averaged over the four simulators. Each variant changes one factor of the complete configuration (**Qoboter**), with the loop on throughout: *with SAM* adds the open-vocabulary `sam` segmentation tool to **Qoboter**, and the other three variants remove or replace a factor. Small numbers in the **Average** column give the change in points from **Qoboter** on the same model. Settings are in Appendix B.

		Per episode		
Setting	**Model**	Average SR (%) (↑)	Turns (↓)	Out tokens (K) (↓)
Qoboter (complete)	Qwen3.8-Max	44.2	69	151
	Qwen3.8-Flash-Next	33.8	99	277
with SAM	Qwen3.8-Max	46.7 (+2.5)	70	146
	Qwen3.8-Flash-Next	44.2 (+10.4)	90	232
without skills	Qwen3.8-Max	41.7 (−2.5)	74	144
	Qwen3.8-Flash-Next	34.2 (+0.4)	107	293
minimal tools	Qwen3.8-Max	37.5 (−6.7)	80	162
	Qwen3.8-Flash-Next	26.7 (−7.1)	101	294
granular tools	Qwen3.8-Max	25.8 (−18.4)	55	146
	Qwen3.8-Flash-Next	25.0 (−8.8)	106	310

and every request hitting the prompt cache, a single experimental round costs several thousand US dollars, which limits how many configurations we can evaluate with these models.

4.2 Qoboter versus general-purpose scaffolds (Q1)

Higher success at similar cost. With the model fixed, Qoboter raises the average success rate over the coding scaffold in every model block (Table 2): from 75.8% to 90.0% with Codex on GPT-6-Astra (+14.2 points), from 43.3% to 55.0% with Claude Code on Opus 5 (+11.7 points), and from 40.8% to 44.2% with Claude Code on Qwen3.8-Max (+3.4 points). This gain comes at a similar token cost: output tokens per episode fall by 10.8% on GPT-6-Astra and by 7.9% on Opus 5, and rise by 6.2% on Qwen3.8-Max. On GPT-6-Astra and Opus 5, no simulator gains more than RoboDojo, whose long, dual-arm tasks give the most room for an unverified step to derail an episode (from 43.3% to 80.0% and from 23.3% to 40.0%); on Qwen3.8-Max, the gain comes from RoboTwin and RoboLab. Qoboter never falls below the coding scaffold on any simulator.

The gain comes from the robot-centric design. Dropping the coding scaffold alone does not help. The Raw API loop, which removes the scaffold without adding any robot-centric mechanism, is cheaper than the coding scaffold (25.7% fewer tokens on GPT-6-Astra, 13.5% fewer on Opus 5) but also less successful (-13.3 and -4.1 points). Qoboter is more successful than both at a token cost similar to that of the coding scaffold.

Comparison with a robot-agent scaffold. CaP-X, which acts by writing policy code, trails both the coding scaffold and Qoboter on LIBERO, the one simulator it supports: 56.7% against

100.0% for Qoboter on GPT-6-Astra, 33.3% against 63.3% on Opus 5, and 23.3% against 33.3% on Qwen3.8-Max. These results are consistent with the value of revising actions during execution. In the LIBERO case in Figure 4, a stalled release prompts Qoboter to choose a new release location using fresh wrist-camera feedback while retaining the task-level plan.

4.3 Transfer to open-weight models (Q2)

The gain transfers without retuning. Run unchanged on three open-weight models, Qoboter again raises the average success rate over Claude Code (Table 3): from 25.4% to 33.8% on Qwen3.8-Flash-Next (+8.4 points), from 5.0% to 13.3% on the dense Qwen3.8-27B (+8.3 points), and from 36.7% to 41.7% on DeepSeek-V4.1-Flash (+5.0 points). On Qwen3.8-27B, the success rate reaches 2.7 times that of the baseline. Output tokens per episode fall by 13.1% on Qwen3.8-Flash-Next and by 40.6% on Qwen3.8-27B, and rise by 5.8% on DeepSeek-V4.1-Flash. Most of the gain comes from RoboTwin and RoboLab; the three cells where Qoboter trails differ by at most two episodes.

4.4 Contribution of tools and skills (Q3)

Switching to granular tools or removing geometry lowers success on both models (Table 4). The granular variant has the largest drop: the average success rate falls from 44.2% to 25.8% on Qwen3.8-Max (-18.4 points) and from 33.8% to 25.0% on Qwen3.8-Flash-Next (-8.8 points). Removing the geometric tools costs 6.7 and 7.1 points respectively, and raises the number of turns per episode from 69 to 80 on Qwen3.8-Max. We also include this experiment to evaluate whether specialized perception modules can provide complementary benefits beyond the shared robot scaffold, particularly for visual grounding during manipulation. Adding SAM raises success from 44.2% to 46.7% on Qwen3.8-Max (+2.5 points) and from 33.8% to 44.2% on Qwen3.8-Flash-Next (+10.4 points), while output tokens fall on both models and turns per episode fall from 99 to 90 on Qwen3.8-Flash-Next. These results suggest that lightweight task-specific models such as SAM can further improve embodied agents, especially for weaker VLMs. The gain concentrates on the Flash model, whose visual grounding is weaker, which suggests that such models rely more on external segmentation to locate targets. In preliminary experiments with GPT-6-Astra and Opus 5, the tool brought no gain, so we leave it out of the core interface. The skills mainly affect interaction length: removing them raises the number of turns per episode on both models, from 69 to 74 on Qwen3.8-Max and from 99 to 107 on Qwen3.8-Flash-Next. Their effect on success is small and differs in sign: 2.5 points lower without skills on Qwen3.8-Max and 0.4 points higher on Qwen3.8-Flash-Next.

Appendix C visualizes five successful episodes, four in simulation and one on the real robot, that show each component at work, such as a ray gap exposing a wrong pixel match (Figure 5).

5 Conclusion

We presented Qoboter, a robot-centric scaffold that lets a VLM with fixed weights perform robotic manipulation directly. It redesigns three levels: the interaction loop couples task-level planning with closed-loop execution; the tool interface leaves target selection to the model and delegates geometric computation to tools; and the skills guide the model to turn physical feedback such as contact, occlusion, and control error into evidence. Across four simulators, Qoboter outperforms Codex and Claude Code on every frontier model at a similar token cost, and the same configuration transfers unchanged to affordable open-weight models. Ablations show that replacing batched tools with granular calls lowers success the most, while the skills mainly reduce wasted turns.

Limitations. First, evaluating frontier models is expensive, which limits the scale of some experiments; for example, the ablations cover only two Qwen models. Second, the scaffold amplifies a model's visual grounding but cannot supply it: absolute success remains low on smaller open-weight models (13.3% on Qwen3.8-27B). Third, although Qoboter works zero-shot without any training, every step waits for a request to a large model, so execution is slow; practical deployment will require this capability to emerge in smaller, faster models. Finally, contact-rich and fine-grained stages may call for combining the VLM with classical robot algorithms or learned policies.

Future work. Since this zero-shot ability works on frontier models, it is worth exploring what data can strengthen a model's manipulation ability and which environments suit reinforcement learning.

AI USE STATEMENT

We used AI tools to polish and translate the manuscript and to assist with programming. The authors checked the resulting text, code, and outputs.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, et al. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In *Conference on Robot Learning (CoRL)*, 2022.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, et al. Flamingo: a Visual Language Model for Few-Shot Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Anthropic. Claude Code. `https://github.com/anthropics/claude-code`, 2025.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-VL: A Versatile Vision-Language Model for Understanding, Localization, Text Reading, and Beyond. *arXiv preprint arXiv:2308.12966*, 2023.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, et al. Qwen2.5-VL Technical Report. *arXiv preprint arXiv:2502.13923*, 2025.

Ruzena Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):966–1005, 1988.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, et al. π_0: A Vision-Language-Action Flow Model for General Robot Control. *arXiv preprint arXiv:2410.24164*, 2024.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, et al. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. In *Conference on Robot Learning (CoRL)*, 2023.

Xiaowei Cai, Yunuo Cai, Bingao Chen, Jingxiao Chen, Zhi Chen, Siyuan Feng, Tengyu Hou, Jingshun Huang, Han Jiang, Runkun Ju, Dong Li, Mingxiang Li, Shaowei Li, Xinchen Li, et al. τ_0-VLA: a Hierarchical Robot Foundation Model with World-Model-Guided Test-Time Computation. *arXiv preprint arXiv:2608.16885*, 2026.

Boyuan Chen, Zhuo Xu, Sean Kirmani, Brian Ichter, Danny Driess, Pete Florence, Dorsa Sadigh, Leonidas Guibas, and Fei Xia. SpatialVLM: Endowing Vision-Language Models with Spatial Reasoning Capabilities. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024a.

Tianxing Chen, Yue Chen, Zixuan Li, Junyuan Tang, Kailun Su, Haoran Lu, Weijie Wan, Baijun Chen, Songling Liu, Haowen Yan, Honghao Su, Zhiyang Dou, Kaixuan Wang, Dandan Zhang, Yunze Liu, Yan Qin, Qiwei Liang, Qiwei Wu, Zijian Lin, Wenwei Lin, Yuran Wang, Minghua He, Tianshu Wu, Ruihai Wu, Jingquan Zhou, Kai-Chong Lei, Haibao Yu, Yuanfeng Ji, Weiyang Jin, Guanyu Lin, Xiaofan Li, Qi Xiong, Renjing Xu, Zhongyu Li, Wenhao Chai, Enze Xie, Ziwei Wang, Yao Mu, Hao Dong, Wojciech Matusik, Mingyu Ding, Wenbo Ding, Ping Luo, and Masayoshi Tomizuka. RoboDojo: A Unified Sim-and-Real Benchmark for Comprehensive Evaluation of Generalist Robot Manipulation Policies. *arXiv preprint arXiv:2607.04434*, 2026a.

Yanzhe Chen, Zechen Bai, Zhijun Cao, Wenzheng Zeng, Kevin Qinghong Lin, Yiqi Lin, Guoqiang Liang, Kevin Yuchen Ma, Qiming Huang, and Mike Zheng Shou. Show-Harness: Just a VLM Agent Can Play Robots. *arXiv preprint arXiv:2609.10522*, 2026b.

Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, Bin Li, Ping Luo, Tong Lu, Yu Qiao, et al. InternVL: Scaling up Vision Foundation Models and Aligning for Generic Visual-Linguistic Tasks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024b.

An-Chieh Cheng, Hongxu Yin, Yang Fu, Qiushan Guo, Ruihan Yang, Jan Kautz, Xiaolong Wang, and Sifei Liu. SpatialRGPT: Grounded Spatial Reasoning in Vision-Language Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Jadelynn Dao, Milan Ganai, Yasmina Abukhadra, Ajay Sridhar, Mozhgan Nasr Azadani, Katie Luo, Clark Barrett, Jiajun Wu, Chelsea Finn, and Marco Pavone. DIRECT: When and Where Should You Allocate Test-Time Compute in Embodied Planners? In *Conference on Robot Learning (CoRL)*, 2026.

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, Jiasen Lu, Taira Anderson, Erin Bransom, Kiana Ehsani, et al. Molmo and PixMo: Open Weights and Open Data for State-of-the-Art Vision-Language Models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Haoyuan Deng, Haichao Liu, Wenkai Guo, Yuan Ling, Zaijia Yang, Yuanjiang Xue, Haosheng Sun, Liangzi Wang, and Ziwei Wang. Facet-0: A Robotic Foundation Model for Contact-Rich Precise Manipulation. *arXiv preprint arXiv:2609.01596*, 2026.

Xin Ding, Liang Mi, Mingzhe Huang, Zixuan Wang, Chao Zhang, Zixu Hao, Fu Chen, Xiangyu Li, Yikai Zheng, Yaoyu Guo, Weijun Wang, Kun Li, Hao Wu, Yunxin Liu, et al. Zetta ζ: An Efficient Closed-Loop Embodied Harness for Self-Evolving Physical Intelligence. *arXiv preprint arXiv:2608.16590*, 2026.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, et al. PaLM-E: An Embodied Multimodal Language Model. In *International Conference on Machine Learning (ICML)*, 2023.

Jiafei Duan, Wilbert Pumacay, Nishanth Kumar, Yi Ru Wang, Shulin Tian, Wentao Yuan, Ranjay Krishna, Dieter Fox, Ajay Mandlekar, and Yijie Guo. AHA: A Vision-Language-Model for Detecting and Reasoning Over Failures in Robotic Manipulation. In *International Conference on Learning Representations (ICLR)*, 2025.

Zhenxuan Fan, Bo Zhang, Yutong Lin, Yuqian Yuan, Juekai Lin, Liang Liang, Zhuoyi Huang, Wenqiao Zhang, Juncheng Li, Siliang Tang, Jun Xiao, and Yueting Zhuang. RoboSPA: Can VLA Models Go Beyond Simple Scenes and Short-Horizon Tasks? In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2026.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, Guanya Shi, Jiajun Wu, Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi Fan. CaP-X: A Framework for Benchmarking and Improving Coding Agents for Robot Manipulation. In *International Conference on Machine Learning (ICML)*, 2026.

Xingyu Fu, Yushi Hu, Bangzheng Li, Yu Feng, Haoyu Wang, Xudong Lin, Dan Roth, Noah A. Smith, Wei-Chiu Ma, and Ranjay Krishna. BLINK: Multimodal Large Language Models Can See but Not Perceive. In *European Conference on Computer Vision (ECCV)*, 2024.

Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, Steven Bohez, Konstantinos Bousmalis, Anthony Brohan, Thomas Buschmann, et al. Gemini Robotics: Bringing AI into the Physical World. *arXiv preprint arXiv:2503.20020*, 2025.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, et al. Gemini: A Family of Highly Capable Multimodal Models. *arXiv preprint arXiv:2312.11805*, 2023.

Yutong Hu, Fengjiao Chen, Xuezhi Cao, and Renaud Detry. 2AM: Grounding Agent-Side Memory as Guidance for Steerable Action Models in Long-Horizon Manipulation. *arXiv preprint arXiv:2609.11308*, 2026.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, et al. Inner Monologue: Embodied Reasoning through Planning with Language Models. In *Conference on Robot Learning (CoRL)*, 2022.

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.

Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. ReKep: Spatio-Temporal Reasoning of Relational Keypoint Constraints for Robotic Manipulation. In *Conference on Robot Learning (CoRL)*, 2024.

Seth Hutchinson, Gregory D. Hager, and Peter I. Corke. A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12(5):651–670, 1996.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? In *International Conference on Learning Representations (ICLR)*, 2024.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models. *arXiv preprint arXiv:2001.08361*, 2020.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, Peter David Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Yecheng Jason Ma, Patrick Tree Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, Huy Ha, Arhan Jain, Abraham Lee, Youngwoon Lee, Marius Memmel, Sungjae Park, Ilija Radosavovic, Kaiyuan Wang, Albert Zhan, Kevin Black, Cheng Chi, Kyle Beltran Hatch, Shan Lin, Jingpei Lu, Jean Mercat, Abdul Rehman, Pannag R Sanketi, Archit Sharma, Cody Simpson, Quan Vuong, Homer Rich Walke, Blake Wulfe, Ted Xiao, Jonathan Heewon Yang, Arefeh Yavary, Tony Z. Zhao, Christopher Agia, Rohan Baijal, Mateo Guaman Castro, Daphne Chen, Qiuyu Chen, Trinity Chung, Jaimyn Drake, Ethan Paul Foster, Jensen Gao, Vitor Guizilini, David Antonio Herrera, Minho Heo, Kyle Hsu, Jiaheng Hu, Muhammad Zubair Irshad, Donovon Jackson, Charlotte Le, Yunshuang Li, Kevin Lin, Roy Lin, Zehan Ma, Abhiram Maddukuri, Suvir Mirchandani, Daniel Morton, Tony Nguyen, Abigail O'Neill, Rosario Scalise, Derick Seale, Victor Son, Stephen Tian, Emi Tran, Andrew E. Wang, Yilin Wu, Annie Xie, Jingyun Yang, Patrick Yin, Yunchu Zhang, Osbert Bastani, Glen Berseth, Jeannette Bohg, Ken Goldberg, Abhinav Gupta, Abhishek Gupta, Dinesh Jayaraman, Joseph J Lim, Jitendra Malik, Roberto Martín-Martín, Subramanian Ramamoorthy, Dorsa Sadigh, Shuran Song, Jiajun Wu, Michael C. Yip, Yuke Zhu, Thomas Kollar, Sergey Levine, and Chelsea Finn. DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset. In *Robotics: Science and Systems (RSS)*, 2024.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, et al. OpenVLA: An Open-Source Vision-Language-Action Model. In *Conference on Robot Learning (CoRL)*, 2024.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment Anything. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Zixing Lei, Changxing Liu, Yichen Xiong, Minhao Xiong, Yuanzhuo Ding, Zhipeng Zhang, Weixin Li, and Siheng Chen. Towards Long-horizon Embodied Agents with Tool-Aligned Vision-Language-Action Models. *arXiv preprint arXiv:2605.13119*, 2026a.

Zixing Lei, Gengze Zhou, Xiong-Hui Chen, Jiazhao Zhang, Yiyang Huang, Hang Yin, Haoqi Yuan, Qi Wu, Weixin Li, and Siheng Chen. Scaffolding Foundation Models into Physical-World Agents Pushes the Frontier of Long-Horizon Navigation. *arXiv preprint arXiv:2608.30396*, 2026b.

Hongyu Li, Bowen Wen, Xinghao Zhu, Yixuan Wang, Yilun Du, Yunzhu Li, George Konidaris, Stan Birchfield, Soha Pouya, Chenran Li, and Yan Chang. Hydra-0: Action Flow for Generalist World Modeling and Control. *arXiv preprint arXiv:2608.18077*, 2026a.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Oier Mees, Karl Pertsch, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, et al. Evaluating Real-World Robot Manipulation Policies in Simulation. In *Conference on Robot Learning (CoRL)*, 2024.

Yijian Li, Changze Li, Hantian Shi, Jiaying Luo, Jiyuan Cai, Ming Yang, and Tong Qin. AgenticNav: Zero-Shot Vision-and-Language Navigation as a Tool-Calling Harness. *arXiv preprint arXiv:2606.10577*, 2026b.

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.

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*, 2023a.

Fangchen Liu, Kuan Fang, Pieter Abbeel, and Sergey Levine. MOKA: Open-World Robotic Manipulation through Mark-Based Visual Prompting. In *Robotics: Science and Systems (RSS)*, 2024a.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual Instruction Tuning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023b.

Lin Liu, Zhicheng Bao, Lu Zhang, Ziying Song, Wu Yang, Yuzheng Zhuang, Shuai Tao, Wulong Liu, Caiyan Jia, and Huchuan Lu. LIBERO-RECOVER: Beyond Task Success Towards Failure Recovery in Robotic Manipulation Models. *arXiv preprint arXiv:2609.05178*, 2026.

Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, and Lei Zhang. Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection. In *European Conference on Computer Vision (ECCV)*, 2024b.

Zeyi Liu, Arpit Bahety, and Shuran Song. REFLECT: Summarizing Robot Experiences for Failure Explanation and Correction. In *Conference on Robot Learning (CoRL)*, 2023c.

Yao Mu, Tianxing Chen, Zanxin Chen, Shijia Peng, Zhiqian Lan, Zeyu Gao, Zhixuan Liang, Qiaojun Yu, Yude Zou, Mingkun Xu, Lunkai Lin, Zhiqiang Xie, Mingyu Ding, and Ping Luo. RoboTwin: Dual-Arm Robot Benchmark with Generative Digital Twins. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Soroush Nasiriany, Fei Xia, Wenhao Yu, Ted Xiao, Jacky Liang, Ishita Dasgupta, Annie Xie, Danny Driess, Ayzaan Wahid, Zhuo Xu, Quan Vuong, Tingnan Zhang, Tsang-Wei Edward Lee, Kuang-Huei Lee, et al. PIVOT: Iterative Visual Prompting Elicits Actionable Knowledge for VLMs. In *International Conference on Machine Learning (ICML)*, 2024.

Open X-Embodiment Collaboration, Abby O'Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, Albert Tung, Alex Bewley, et al. Open X-Embodiment: Robotic Learning Datasets and RT-X Models. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

OpenAI. Codex CLI: Lightweight coding agent that runs in your terminal. `https://github.com/openai/codex`, 2025.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, et al. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560*, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models. *Transactions on Machine Learning Research*, 2024a.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, et al. Qwen2-VL: Enhancing Vision-Language Model's Perception of the World at Any Resolution. *arXiv preprint arXiv:2409.12191*, 2024b.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable Code Actions Elicit Better LLM Agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50208–50232, 2024c.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, et al. Emergent Abilities of Large Language Models. *Transactions on Machine Learning Research*, 2022.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, et al. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2024.

Bingxin Xu, Yuzhang Shang, Zhen Dong, and Emilio Ferrara. Coding Agents with an Obstacle-Aware Harness for Safe Robot Manipulation. *arXiv preprint arXiv:2609.20822*, 2026.

Jenai Xuning Yang, Rishit Dagli, Alex Zook, Hugo Hadfield, Ankit Goyal, Stan Birchfield, Fabio Ramos, and Jonathan Tremblay. RoboLab: A High-Fidelity Simulation Benchmark for Analysis of Task Generalist Policies. In *Robotics: Science and Systems (RSS)*, 2026.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652, 2024a.

Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024b.

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.

Takuma Yoneda, Jiading Fang, Peng Li, Huanyu Zhang, Tianchong Jiang, Shengjie Lin, Ben Picker, David Yunis, Hongyuan Mei, and Matthew R. Walter. Statler: State-Maintaining Language Models for Embodied Reasoning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

Haoqi Yuan, Zhixuan Liang, Anzhe Chen, Ye Wang, Haoyang Li, Pei Lin, Yiyang Huang, Zixing Lei, Tong Zhang, Jiazhao Zhang, Jie Zhang, Jingyang Fan, Gengze Zhou, Qihang Peng, Chenxu Lv, Xiaoyue Chen, An Yang, Fei Huang, Junyang Lin, Dayiheng Liu, Jingren Zhou, Chenfei Wu, and Xiong-Hui Chen. Qwen-RobotManip Technical Report: Alignment Unlocks Scale for Robotic Manipulation Foundation Models. *arXiv preprint arXiv:2606.17846*, 2026.

Wentao Yuan, Jiafei Duan, Valts Blukis, Wilbert Pumacay, Ranjay Krishna, Adithyavairavan Murali, Arsalan Mousavian, and Dieter Fox. RoboPoint: A Vision-Language Model for Spatial Affordance Prediction in Robotics. In *Conference on Robot Learning (CoRL)*, 2024.

Michał Zawalski, William Chen, Karl Pertsch, Oier Mees, Chelsea Finn, and Sergey Levine. Robotic Control via Embodied Chain-of-Thought Reasoning. In *Conference on Robot Learning (CoRL)*, 2024.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *International Conference on Learning Representations (ICLR)*, 2024.

A Evaluation tasks

Platforms. The simulators are chosen so that together they cover dual-arm and single-arm platforms, short and long horizons, and instructions that require grounding language beyond naming an object. **RoboDojo** (Chen et al., 2026a) simulates the dual-arm ARX X5 of our real robot and supplies the longest tasks: using a set square as a tool to push blocks into a row, holding a pen holder with one arm while filling it with the other, inserting tubes into a rack, pouring, and two tasks that hinge on memory (uncovering blocks in a color order after they have been hidden, and picking the object that matches one seen earlier on a conveyor). **RoboTwin** (Mu et al., 2025) uses the dual-arm Aloha-AgileX and contributes articulated objects (opening a microwave), pressing tasks (a bell, a stapler, a switch), and relational placements, often with the acting arm named in the instruction. **RoboLab** (Yang et al., 2026) uses a Franka Panda with a Robotiq 2F-85 gripper in the DROID configuration (Khazatsky et al., 2024), and its instructions call for counting (adding one more banana, or making the total three), comparison (the bigger pumpkin), spatial relations, and taking objects out of containers. **LIBERO** (Liu et al., 2023a) uses a single Franka Panda; our ten tasks are drawn from its Spatial, Object, Goal, 90, and 10 suites. The **real robot** is a dual-arm ARX X5 (Appendix G), and its five tasks add what simulation renders poorly, such as a soft doll that slips under a firm grasp and a towel pressed and slid across a stain.

Roster. Every simulator task is run from three initial scenes, giving 30 episodes per simulator and 120 per configuration, and each real task is run five times, giving 25 episodes per configuration. The roster is frozen before evaluation and shared by every model and scaffold. Table 5 lists the 40 simulation tasks, and Table 6 gives the task ID and instruction behind each label in Figure 3, including the five real-robot tasks.

B Evaluation settings

This section gives the settings behind Tables 2, 3, and 4.

Shared protocol. Within each model block the weights, tasks, initial states, sensor access, low-level executor, independent judge, and budget are identical, so only the scaffold changes, and comparisons are valid within a block only. Success is judged independently of the agent: in simulation by the environment's evaluator, and on the real robot by pre-registered observable conditions checked

Table 5: **Evaluation task roster for the full-system comparison** (Table 2). Ten tasks in each of the four simulators, 40 in total, each run over three scenes for 30 episodes per cell. This fixed test set is shared by every scaffold and model in Table 2; the real robot uses a separate five-task set. Task IDs are given verbatim as recorded in the run logs.

Simulator	Tasks (10 each, $\times 3$ scenes)
RoboDojo	`align_blocks`, `cover_blocks`, `fill_pen_holder`, `general_pickup`, `insert_tubes`, `match_and_pick_from_conveyor`, `pour_liquid_into_cup`, `stack_blocks_by_language`, `stack_bowls`, `swap_T`
RoboTwin	`adjust_bottle`, `blocks_ranking_rgb`, `click_bell`, `move_playingcard_away`, `open_microwave`, `place_a2b_left`, `place_a2b_right`, `place_fan`, `press_stapler`, `turn_switch`
RoboLab	`BananaInBowlTask`, `BananaOnPlateTask`, `BananasInBinOneMoreTask`, `BananasInBinThreeTotalTask`, `BigPumpkinInBinTask`, `BowlInBinTask`, `KeyboardOutOfBinTask`, `PlasticBottlesInSquarePailTask`, `RubiksCubeRightOfBowlTask`, `TakeMeasuringSpoonOutTask`
LIBERO	`libero_10-task01`, `libero_90-task28`, `libero_90-task57`, `libero_90-task60`, `libero_goal-task00`, `libero_object-task01`, `libero_object-task04`, `libero_object-task05`, `libero_spatial-task02`, `libero_spatial-task04`

against retained video; a model's claim to have finished never counts. Each episode is a single attempt (pass@1). The agent may recover within the episode but is never reset for another try, and an episode that exhausts its budget on wall-clock time, model calls, or physical execution time fails. The **Average** column is the unweighted mean of the four per-simulator success rates, so every simulator counts equally regardless of its episode share. Cost is measured in output tokens per episode, summed over every model request, failed episodes and retries included. Absolute token counts are never compared across models with different tokenizers.

Scaffold comparison (Table 2). Each per-simulator cell aggregates up to 30 episodes (10 tasks $\times$ 3 scenes; the roster is in Table 5), so the average summarizes the 120 episodes of a configuration. A few cells drop episodes lost to gateway faults, so a cell's denominator can be slightly below 30. The **Raw API loop** row is the null-scaffold control: it keeps the off levels L_0, T_0, and S_0 of Section 3, and its prompt and tool schema are frozen before evaluation, so it cannot be weakened after the fact. It therefore separates a gain that comes merely from removing coding-scaffold overhead from a gain that comes from the robot-centric loop, tools, and skills. Real-robot results are reported on our anonymous website (Appendix G).

CaP-X. We run CaP-X at a pinned commit and record its agent, tier, interaction mode, and tool setting; configurations with different training or inference settings are never merged into one row of Table 2. CaP-X is the only external robot-agent baseline in this round, so we claim a comparison against it and against the general-purpose coding scaffolds, not against the published robot-agent literature as a whole.

Transfer (Table 3). The loop, tool set, skills, and budgets are fixed during development with Opus 5 (the skills are distilled from its trajectories and the tools are selected on development tasks) and transferred without per-model tuning; only mechanical protocol adaptation (tool JSON, image encoding) is allowed. The three open-weight models span a range of scales, from a dense 27B model to MoE models with 6B and 16B active parameters. Each per-simulator cell aggregates 30 episodes (10 tasks $\times$ 3 scenes), 120 per configuration, except the Qwen3.8-Flash-Next LIBERO cells, which aggregate 60 paired episodes (10 tasks $\times$ 6 scenes) from a later rerun. Parameter counts are those reported by each model's release; for DeepSeek-V4.1-Flash the active count is the decode-time figure (8B are active during prefill).

Claude Code on transferred models. In Table 3, Claude Code reaches each model through a compatibility bridge. We use it only after checking that the bridge passes the same model weights and the same visual input as **Qoboter** receives, so that an interface fault is not mistaken for a limit of the model. No model receives per-model tuning or a private vision assistant.

Table 6: **Task labels in Figure 3** and the corresponding task IDs and instructions, in gallery order. Simulation instructions are quoted verbatim; real-robot episodes share the environment ID `arx_realworld`, so their IDs are descriptive names. †: trailing execution hints omitted.

Label	Task ID	Instruction
RoboDojo (ARX X5, dual-arm)		
Align blocks	`align_blocks`	Use the set square to push the three blocks into a straight, aligned row, then reset the robot arm.†
Cover blocks	`cover_blocks`	Cover the blocks from left to right, remember their colors, then uncover them in the order: red, green, and blue.
Fill pen holder	`fill_pen_holder`	Hold the pen holder with one hand, place all pens into it with the other hand, then put it back down.
Pick up scissors	`general_pickup`	Pick up the mint green scissors by 10 cm.
Insert tubes	`insert_tubes`	Insert the three tubes into the rack one by one.
Conveyor pick	`match_and_pick_from_conveyor`	Remember the first object on the conveyor, then pick the matching object when it appears again.
Pour into cup	`pour_liquid_into_cup`	Pour the liquid from the bottle into the cup.
Ordered stack	`stack_blocks_by_language`	stack the blocks from bottom to top in the order of blue, yellow, and orange, then reset the robot arm.
Stack bowls	`stack_bowls`	Stack the three bowls together.
Swap T-blocks	`swap_T`	Pick up the two T-shaped blocks, swap their positions, and place them back with the correct orientations.†
RoboTwin (Aloha-AgileX, dual-arm)		
Adjust bottle	`adjust_bottle`	Pick up the green plastic bottle with ridged bottom carefully using the left arm
Rank blocks	`blocks_ranking_rgb`	Use the left arm to arrange red block first, then use the right arm for green block, and finally use the left arm for blue block in a row.
Click bell	`click_bell`	Click the lightweight bell with smooth surface's top center on the table.
Move card away	`move_playingcard_away`	Grab the rectangular playingcards pack and move it outward.
Open microwave	`open_microwave`	Pull the medium gray microwave with top vents open with the left arm.
A left of B	`place_a2b_left`	Use the right arm to place the teal toycar left of the phone with flat rectangular back.
A right of B	`place_a2b_right`	Put the palm-sized black stapler to the right of the medium irregular woodenblock.
Place fan	`place_fan`	Use the right arm to grab the fan with small round shape and place it on the Silver mat facing the robot.
Press stapler	`press_stapler`	Press down on the blue stapler for holding papers firmly using the right arm.
Turn switch	`turn_switch`	Press the small flat switch tan top using the left arm
RoboLab (Franka Panda, single-arm)		
Banana in bowl	`BananaInBowlTask`	Pick up the banana and place it in the bowl
Banana on plate	`BananaOnPlateTask`	Pick up the banana and put it on the plate
+1 banana	`BananasInBinOneMoreTask`	Put one (1) more bananas in the grey bin.
Three bananas	`BananasInBinThreeTotalTask`	Make sure there are 3 (three) bananas in the grey bin.
Bigger pumpkin	`BigPumpkinInBinTask`	Put the bigger pumpkin in the bin
Bowl in bin	`BowlInBinTask`	put the bowl in the grey bin
Keyboard out	`KeyboardOutOfBinTask`	Take the keyboard out of the bin and put it on the table
Bottles in pail	`PlasticBottlesInSquarePailTask`	Put all the small plastic bottles in the square pail
Cube to right	`RubiksCubeRightOfBowlTask`	Put the rubiks cube to the right of the bowl
Take spoon out	`TakeMeasuringSpoonOutTask`	Take the white colored measuring spoon out of the red bowl and put it on the table.
LIBERO (Franka Panda, single-arm)		
Two to basket	`libero_10-task01`	put both the cream cheese box and the butter in the basket
Close drawer	`libero_90-task28`	close the top drawer of the cabinet
Cheese to tray	`libero_90-task57`	pick up the cream cheese and put it in the tray
Left bowl to tray	`libero_90-task60`	pick up the black bowl on the left and put it in the tray
Open drawer	`libero_goal-task00`	open the middle drawer of the cabinet
Pick cheese	`libero_object-task01`	pick up the cream cheese and place it in the basket
Pick ketchup	`libero_object-task04`	pick up the ketchup and place it in the basket
Pick sauce	`libero_object-task05`	pick up the tomato sauce and place it in the basket
Bowl to plate	`libero_spatial-task02`	pick up the black bowl from table center and place it on the plate
Drawer bowl	`libero_spatial-task04`	pick up the black bowl in the top drawer of the wooden cabinet and place it on the plate
Real robot (ARX X5, dual-arm)		
Stack cylinder	`stack_cylinder_on_cube`	stack the cylinder onto the cube
Doll to plate	`doll_to_plate`	put the doll into the plate that is closer to the water cup
Cylinder to plate	`cylinder_to_plate`	put the cylinder into the plate that is closer to the water cup
Wipe table	`wipe_with_towel`	wipe the water stain in the middle of the table with the towel.†
Open drawer	`open_drawer_place_part`	open the middle drawer and put the gray part into it.

Ablation (Table 4). All variants run under one controlled API runtime with the loop on, and share the same low-level action guards and the same independent judge. The **Qoboter** (complete) row uses the five tools of Table 1 together with the generic robot-procedure skills (perception, stable grasping). The *with SAM* row adds the open-vocabulary `sam` segmentation tool to this configuration. The *without skills* row removes the skills. The *minimal tools* row keeps only `observation_batch` and `act_batch`, dropping the geometric tools. The *granular tools* row replaces the batched tools with the discrete `get_*` and `act_*` calls, so no call is batched. Each row covers 120 episodes per model (10 tasks $\times$ 3 scenes $\times$ 4 simulators) over a fixed denominator of 30 per simulator, and missing or unbooted runs count as failures. The one exception is the **Qoboter** (complete) row on Qwen3.8-Flash-Next, which is the **Qoboter** row of Table 3, so its LIBERO cell aggregates 60 paired episodes. Out tokens are output tokens per episode, in thousands; input tokens, which are dominated by context, are omitted.

C Qualitative results

This appendix visualizes five episodes, listed in Table 7, all of which end in success. The first four come from the simulators, and each isolates one component of Section 3: the interaction loop (Appendix C.1), the geometric tools (Appendix C.2), and the acting and perceiving skills (Appendices C.3 and C.4). The fifth follows a real-robot episode in which all of them appear together (Appendix C.5). Every figure is assembled from the episode's own record: the panels are images returned to the model by its tool calls or frames from the synchronized recordings, the numbers are copied from tool results, and quoted text is the model's own. The episodes were chosen to show the mechanisms at work, not sampled to be typical; success rates are those of Tables 2, 3, and 4.

Table 7: **Episodes visualized in Appendix C.** The simulated episodes are run by GPT-6-Astra and the real-robot episode by GPT-6-Sol.

Fig.	Component	Platform, task	What the episode shows
4	Interaction loop	LIBERO, two items to basket	A stalled release is re-decided from the wrist view, and the plan is kept
5	Geometric tools	RoboDojo, insert tubes	A ray gap of 67.8 mm exposes a wrong pixel match before any arm motion
6	Acting skill	RoboTwin, turn switch	A "converged" press that stops 4.8 mm short is read as contact
7	Perceiving skill	RoboDojo, lift toy car	The idle arm is moved to a side view to measure pad height
8	All components	Real robot, cylinder to plate	A hand-over, two repaired grasps, and a check after each withdrawal

C.1 Interaction loop: re-deciding one step, keeping the plan

Figure 4 follows the loop of Section 3.1 through a LIBERO episode in which GPT-6-Astra must put the cream cheese and the butter into the basket. Before any motion the model fixes the order of sub-goals: lift the butter with a test lift, place it, then do the same for the cream cheese (a). During placement the descent meets contact just inside the basket rim, and the model backs off instead of pushing (b). The release then stalls: repeated open commands leave the jaws at about 0.52, and the wrist view shows the butter still following the hand (c). The model marks the sub-goal as incomplete and re-decides only that step from the wrist view, moving the release to the clear centre of the basket rather than repeating the stalled descent. The jaws then open to 0.978 and the butter rests in the basket (d). The plan itself is never rewritten: the model continues with the cream cheese as planned, and the episode succeeds (e).

C.2 Geometric tools: a wrong correspondence caught by the ray gap

Figure 5 follows the geometric tools (Section 3.2; implementation in Appendix E) through a RoboDojo tube-insertion episode with GPT-6-Astra. In its first `triangulate_batch` call the model localizes the left tube cap and one rack corner, marking each in two of the initial images (a). The cap is matched correctly, and its rays meet within $0.29\,\mathrm{mm}$. The rack pair is not: the head-camera

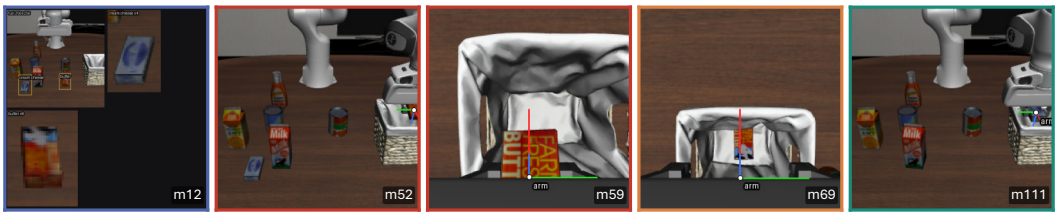


Figure 4: **Re-deciding one step, keeping the plan** (LIBERO, putting the cream cheese and the butter in the basket, GPT-6-Astra). Panels (a), (b), and (e) are the front view and (c), (d) the wrist view, each returned with the tool call at the labelled message. (a) to (e) follow the steps described in Appendix C.1.

pixel lies on the rack's left-front corner, but that corner is outside the initial right-wrist view, so the right-wrist pixel marks another part of the rack. The tool does not turn the pair into a point to be trusted; it reports that the solved point lies $67.8\,\mathrm{mm}$ from one of the rays. The model reads this signal and discards the estimate ("The left cap triangulates cleanly; the rack-corner correspondence does not, so I am discarding that rack estimate."), before any arm command has been issued; the discarded point is never used later in the episode. The model does not say why the pair fails; the out-of-frame diagnosis is ours, checked against the logged calibration. It then carries out the step that is already reliable, grasping the left tube, and only when the rack is needed moves the right arm to a side view that shows the whole rack (b). There it marks three points on the rack's back edge that both views contain; all three sets of rays meet within $2\,\mathrm{mm}$ and the three heights agree within $3\,\mathrm{mm}$, which gives the slot-mouth height it uses before carrying the first tube. Later the right tube cap is triangulated from the start-of-episode images, passing the stored wrist view together with the pose at which it was taken (c). Throughout, which pixel to use is the model's decision and the arithmetic is the tool's, and the ray gap is what lets the model notice its own wrong match before the arm moves.

C.3 Acting skill: a converged controller that stops short

Figure 6 follows the acting skill (Section 3.3) through a RoboTwin `turn_switch` episode run by GPT-6-Astra. With the idle right arm serving as a side camera, the model aligns the closed fingertips with the upper half of the rocker switch and advances in 5 mm steps (a). Each step returns "converged", but the reported fingertip position stops advancing: a command to $y = 0.0475\,\mathrm{m}$ ends at $0.0427\,\mathrm{m}$, a residual of 4.8 mm against about 1.3 mm on the free steps before it (b). The model ranks the measured position above the controller's status and records the stall as resistance at the upper half rather than pushing harder ("I won't keep pushing there"). It withdraws, lowers the fingertip by about 3 cm to the protruding lower half (c), and presses it in 3.5 mm steps; the next step triggers success (d).

C.4 Perceiving skill: the idle arm as a side camera

Figure 7 follows the perceiving skill (Section 3.3; further rules in Appendix D) through a RoboDojo episode in which GPT-6-Astra must lift a toy car by 10 cm with the right arm. With the open gripper hovering above the car (a), neither available view carries the dimension the grasp depends on. The head camera looks down, so the gripper overlaps the car and pad height cannot be read, and the executing wrist camera looks along the approach axis, which fixes the lateral offset but not the depth. The model names the missing dimension ("the remaining uncertainty is pad height"), holds the right arm still, screens reachable viewpoints for the idle left arm, and moves it to a lateral pose (b). The new view shows the pads still above the roof, so the model concludes that "closing now would be too shallow" (c) and lowers the gripper in steps checked against the side view. When the first test lift pitches the car nose-down in that view, the model reads it as a rear-biased grasp, releases the car, and grasps again with the pads flanking the central body (d). The second test lift raises both

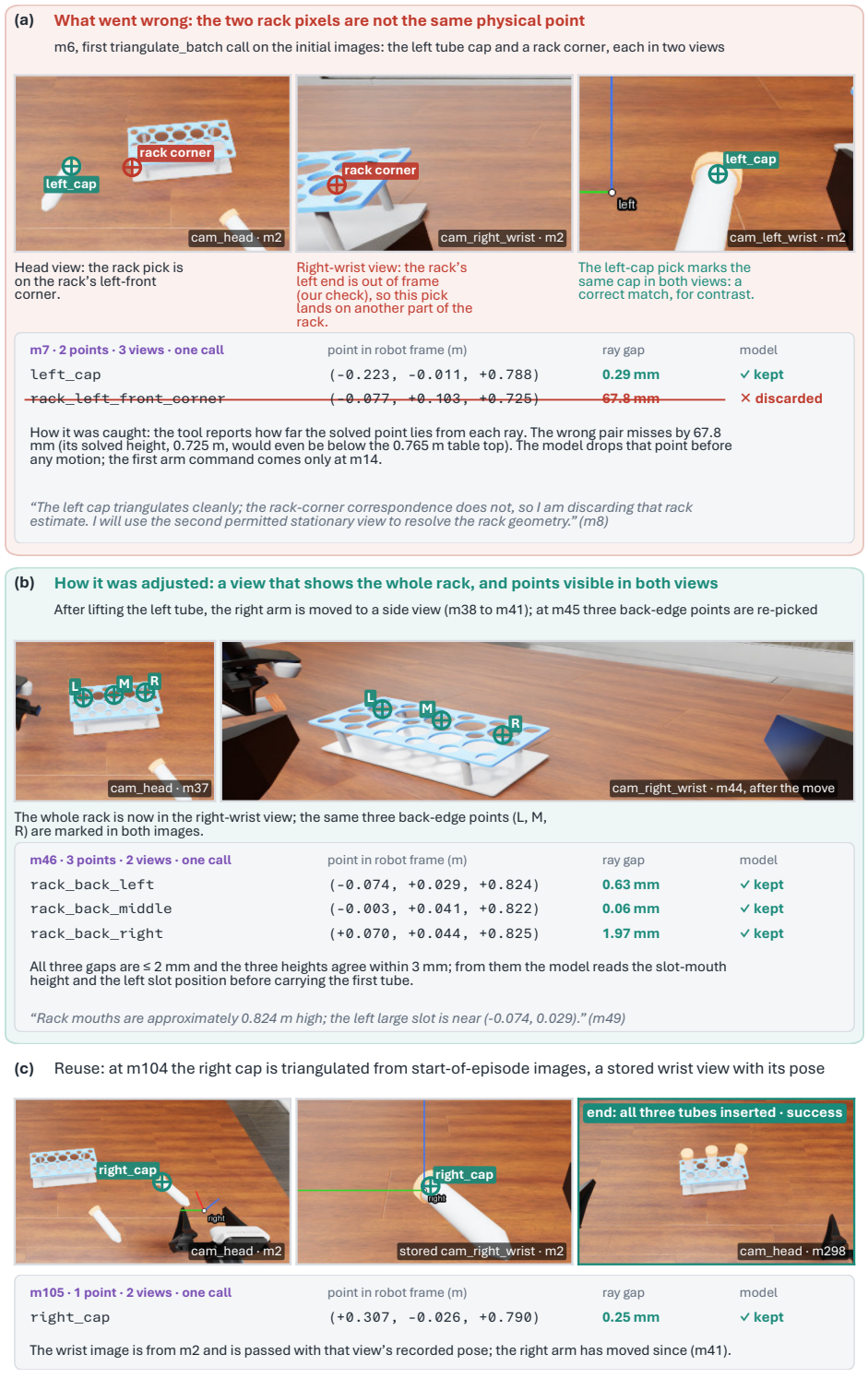


Figure 5: **A wrong correspondence caught by the ray gap** (RoboDojo, tube insertion, GPT-6-Astra). Circles mark the pixels the model passed to `triangulate_batch`, teal for points it kept and red for the point it discarded. Each table lists the robot-frame points and ray gaps returned by one call; mk is the message index in the episode log. (a) The first call. (b) The rack re-localised from a side view. (c) A later call that reuses a stored wrist view. The episode ends with all three tubes inserted.

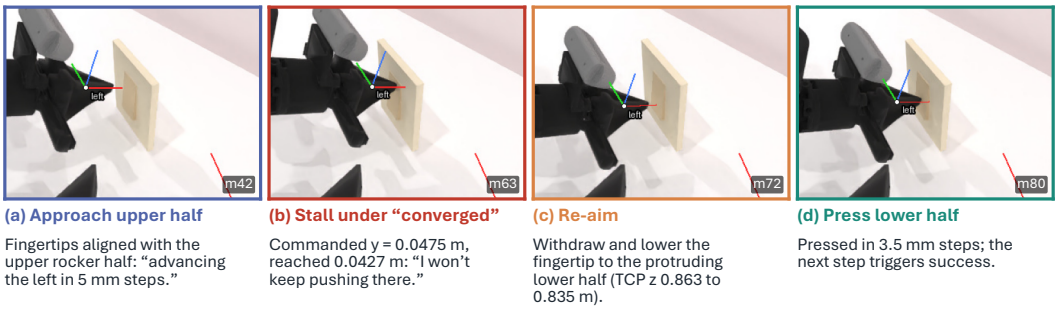


Figure 6: **A converged controller that stops short** (RoboTwin, `turn_switch`, GPT-6-Astra). Each panel is the side view returned with the action at the labelled message. (a) to (d) follow the steps described in Appendix C.3.

ends together with a gap visible below (e); only then does the model return the observation arm to its initial pose and finish the lift.

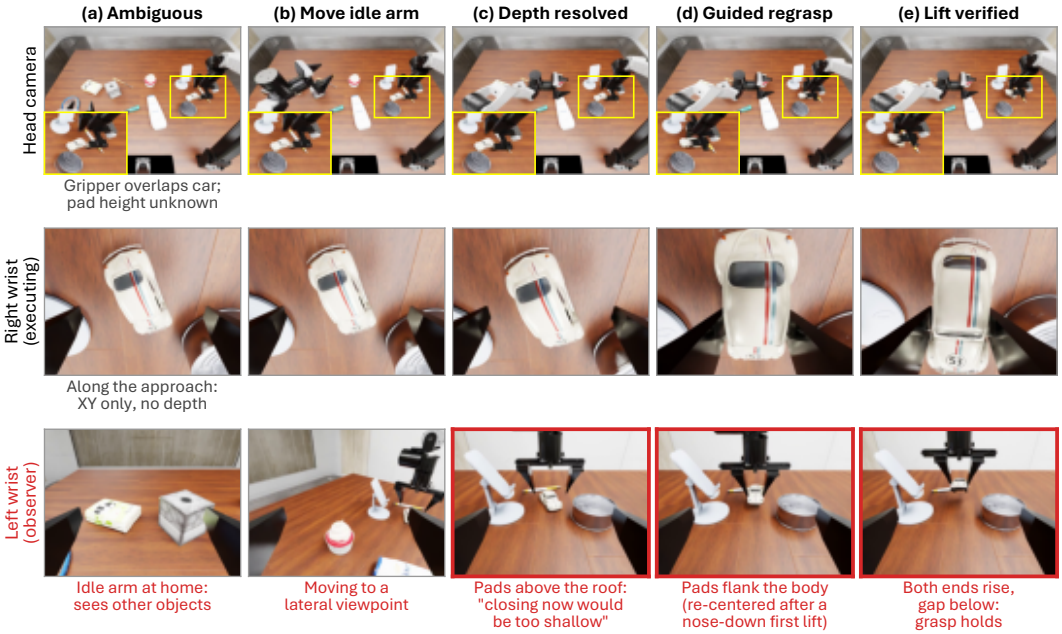


Figure 7: **The idle arm as a side camera** (RoboDojo, lifting a toy car, GPT-6-Astra). Rows show the head camera, the wrist camera of the executing right arm, and the wrist camera of the idle left arm, taken from the episode's synchronized recordings; in the head row the inset magnifies the boxed region around the gripper. Red frames mark the side view once the left arm holds its observation pose. Columns (a) to (e) follow the steps described in Appendix C.4.

C.5 Real robot: the same mechanisms on hardware

Figure 8 follows one *Cylinder to plate* episode run by GPT-6-Sol under Qoboter on the ARX X5 (setup in Appendix G) and judged a success by the operator. It shows that each mechanism of the simulated episodes above has a counterpart on hardware. *Loop.* Before any motion the model resolves the referring expression (the left plate is nearer the cup), and a reachability check shows that the right arm, which is next to the cylinder, cannot reach that plate; the plan therefore hands the cylinder over through a clear table spot (a). Both later failures are repaired at the grasp where they occur, and the plan is never rewritten, as in Figure 4. *Acting.* A gripper reading of 0.042 after the first close exposes an empty grasp before any lift; the gripper shifts 1.5 cm forward and closes on the

cylinder at about 31 mm (b). The first left grasp passes the close but narrows from 0.288 to 0.261 during a 1 cm test lift, so the model sets the cylinder back down and grasps 1 cm deeper (e). When the cylinder is lowered at the hand-over spot, a 5 mm command that completes only in part is read as table contact and triggers the release, the real-robot analogue of Figure 6. *Perceiving.* With the right arm occluding the gap to the table in the head view, the model sends the idle left arm towards a side view, as in Figure 7 (c). Before the left grasp, the head view shows the gripper over the cylinder, but the wrist view, which carries the lateral offset, shows it biased toward one pad and prompts a 1 cm correction (d). *Progressing.* Each hand is withdrawn before its stage counts as done: the hand-over is accepted only once the cylinder stays put after the right gripper lifts away, and the placement only once it remains in the plate after the left hand clears (f). The episode takes 61 motion commands and about 29 minutes, with no over-current event on either gripper.

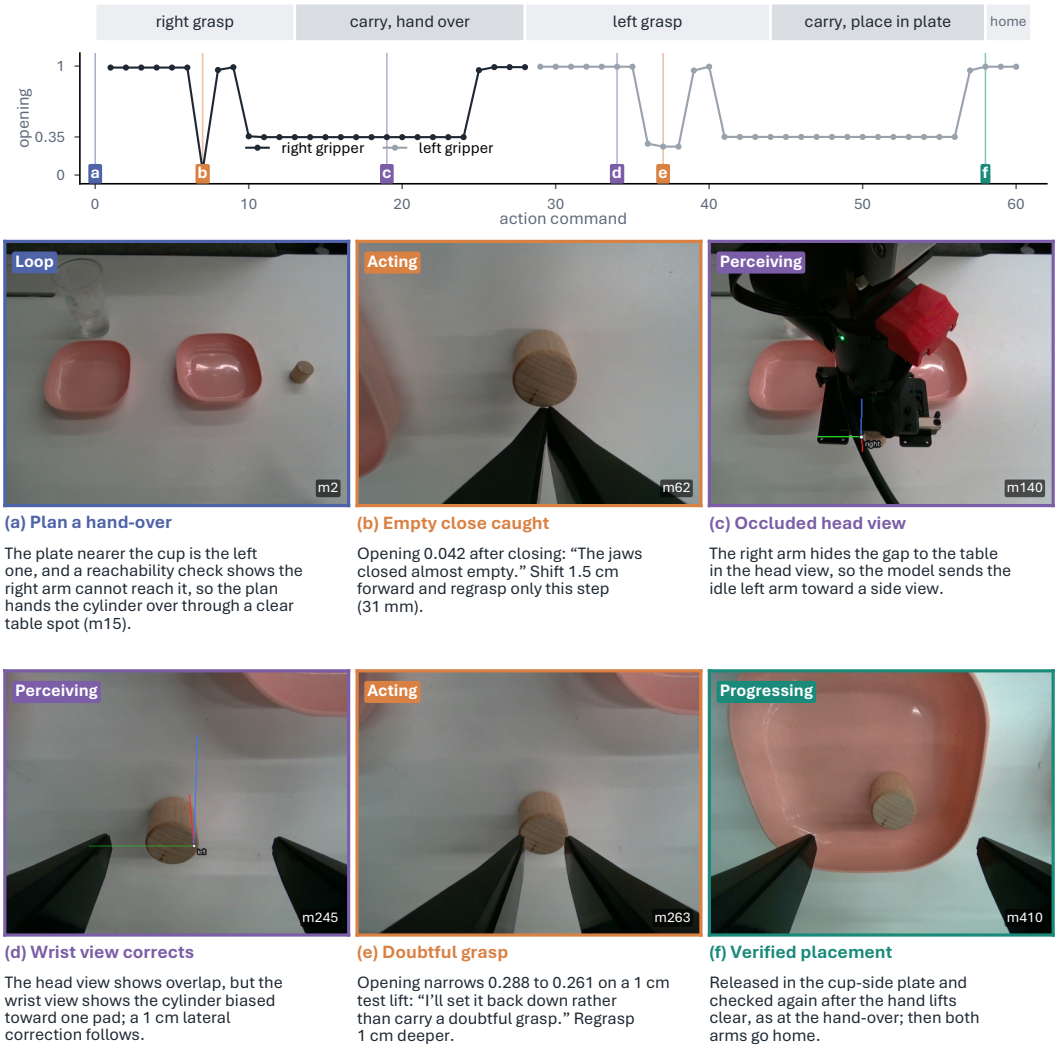


Figure 8: **One real-robot episode across the four principles** (ARX X5, *Cylinder to plate*, GPT-6-Sol, success judged by the operator). *Top:* the opening of the right (dark) and left (grey) gripper after each of the 61 motion commands, under our summary of the episode’s stages; letters mark the commands at which panels (a) to (f) are taken. *Bottom:* each panel is a camera image returned to the model at the labelled message and is tagged with the principle it illustrates; the steps are described in Appendix C.5.

D SKILL DETAILS

This appendix records rules of the acting and perceiving skills (Section 3.3) that the main text omits. For acting, a `converged` report is checked against the measured pose and does not certify a grasp, and a blocked gripper shows only that something resists between the fingers, so an image must still identify the object. For perceiving, on a single-arm platform, where no idle arm can serve as a side camera, the model prefers triangulation or a static stereo pair and moves the arm to look only when its hand is empty. An unchanged state may be inspected at most twice; after that, remaining uncertainty must be resolved by an action that produces new evidence. A triangulation whose rays fail to meet is discarded rather than refined, since refinement would rest on a faulty correspondence. Example episodes of the acting and perceiving skills are in Appendices C.3 and C.4.

E TOOL INTERFACE

This appendix summarizes the five core tools in Table 1, using RoboDojo as the reference implementation. Platform-specific support is described in Appendix F.

Shared conventions. Poses refer to the grasp point between the gripper pads, with positions in metres and orientations as rotation vectors in radians in a fixed world frame. Camera calibration provides intrinsics and the camera-to-world transform. Gripper opening is normalized, with 0 closed and 1 fully open; feedback reports the measured opening rather than the last command.

Tool surfaces. The server provides three interfaces (Table 8). The granular interface exposes separate reads and actions; the batch interface groups recurring operations; the minimal interface retains only batched observation and motion. The *granular tools* and *minimal tools* ablations use the granular and minimal interfaces, respectively. SAM can be added as a separate segmentation tool, as in the *with SAM* variant. The granular and batch surfaces also expose instruction reads and an insertion-search primitive outside the five-tool core.

Table 8: **Core operations on the RoboDojo tool surfaces.** The minimal interface removes geometric queries while retaining batched observation and motion.

Operation	Granular	Batch	Minimal
Observation	Separate reads	Synchronized read	Synchronized read
Motion	Separate commands	Unified action call	Unified action call
Pixel back-projection	Available	Available	Omitted
Triangulation	Two views	Multiple views	Omitted
Reachability query	Available	Available	Omitted

`observation_batch`. One call returns the selected camera images, their calibration, and each arm's TCP pose, joint positions, and measured gripper opening as a synchronized snapshot. Calibration is recorded with each image, allowing a stored wrist view to be used at its capture pose after the camera moves. Optional TCP overlays and magnified image regions support visual alignment; magnified views include the mapping back to original image pixels.

`unproject_pixels`. Given a camera and a list of pixels, the tool casts calibrated rays into the world frame. If the model supplies a plane height, it also returns forward ray-plane intersections where defined. No surface plane is assumed by default, and no depth image is used. A stored camera pose can replace the live pose when the pixels come from an earlier image.

`triangulate_batch`. The model supplies matched pixels from at least two calibrated views for each target point. The tool estimates the 3D point by minimizing its squared distances to the viewing rays. Views may come from different cameras or from one wrist camera at different poses, provided the target point remains stationary. The reply includes ray-gap diagnostics: the largest distance from the estimated point to an input ray indicates correspondence consistency. Insufficient parallax or a solution behind a camera produces no usable point. The model decides whether to accept other estimates from their diagnostics; Appendix C.2 shows it rejecting a mismatched pair before moving the arm.

`check_tcp_pose_reachable`. Given an arm and candidate poses, the tool returns a feasibility result and any failure reason for each pose without moving the robot. On RoboDojo, it queries the planner's inverse kinematics from the current joint state. The result is conditional on that configuration and does not certify a collision-free path or successful contact. Other platforms use the feasibility checks listed in Table 9.

`act_batch`. The model selects an arm, specifies the gripper command, and requests gripper control, absolute or relative end-effector motion, or a joint reset. Relative motion can use the world or tool frame; a joint reset returns to the episode's initial configuration. A path may contain several poses executed in order with one gripper command. The reply reports the measured arm state, the termination reason, and any interruption. A whole path counts as one motion command under the server's command budget (Appendix F.2).

Batching and visual feedback. Batched reads and geometric queries collect multiple requested results in one reply, reducing model round trips. Motion paths execute sequentially. The batch interface separates action from visual inspection: `act_batch` returns no image, so the model obtains fresh images through `observation_batch`. Granular actions instead include camera images in their replies.

F PLATFORM INTEGRATION

Every platform is reached through the same path: the agent loop calls tools of an MCP server, the MCP server forwards each call over HTTP to an environment server, and the environment server runs the call against the platform. This appendix describes that path, what a platform must supply to be plugged in, the adapter written for each platform, and the parts of the episode lifecycle that all platforms share. The computations behind the tools are described in Appendix E, and the real-robot hardware and guards in Appendix G.

F.1 ARCHITECTURE

Environment server. Each platform runs one environment server process per evaluation slot, built on a small standard-library bridge package. The server speaks a JSON protocol over HTTP (`agent-env`, version 1.1) with endpoints for a health check, a description of the served tools and platform metadata, episode start, restart, and end, and a single tool call; images are served separately as artifacts. The platform side is a runtime object with a fixed lifecycle (start-up, start of an episode, restart, end of an episode, an optional idle tick, and shutdown) and a set of tool handlers registered before the server starts. HTTP requests arrive on background threads, but every lifecycle call and every tool call is queued and executed serially on the one thread that owns the simulator or the robot, so thread-bound simulator state is never touched concurrently and each call observes the state the previous one left. A watchdog marks the server unhealthy if a command stalls on that thread (300 s by default), and renderer or physics errors that require a fresh process, such as a lost GPU device, are propagated instead of being returned as tool errors. The tool list and the description are frozen when the server starts. Episode identifiers are issued by the server, only the active episode accepts tool calls, and each response is cached under its request identifier so that a retried request is answered from the cache rather than executed twice.

MCP adapter. A runner-side adapter speaks this protocol and exposes the tools of the server's description to the model, with the episode lifecycle kept out of the tool list: the runner, not the model, starts and ends episodes. It has two front ends over one core, an in-process MCP server for the Claude Agent SDK runner and a stdio MCP server for scaffolds that spawn MCP subprocesses; for Codex, the evaluation service passes the same tool descriptions to the Codex app server as dynamic tools in a `robot` namespace. Tools can be filtered by name or tag on the SDK path. A tool result becomes a text block with the JSON payload plus one image block per image artifact, fetched from the server as a preview rendition. Because every platform publishes the same protocol, a scaffold reaches each platform, including the real robot, through exactly the interface it uses on the others.

What a platform supplies. To be plugged in, a platform provides an environment object with the following reads and actions, which the shared tool handlers call.

- *Cameras*: the camera names, RGB frames for any subset of them, and, optionally, an overlay that draws each arm's TCP axes onto a frame.
- *Calibration*: per-camera intrinsics (f_x, f_y, c_x, c_y) with the resolution, and the live camera-to-world extrinsic in the OpenCV convention, which for a wrist camera moves with the arm.
- *Arm state*: for every arm, the grasp-point pose, the joint positions, and the measured gripper opening, returned together as one state snapshot.
- *Motion*: a gripper command, an absolute or relative grasp-point command, and a path through several absolute poses, each returning a termination reason and the state afterwards.
- *Geometry*: pixel back-projection, triangulation, and a reachability query that uses the platform's IK, motion planner, or workspace model without commanding motion.
- *Episode control*: reset to a selected initial scene, an episode status with the success flag of the platform's own success judge, and a final verdict when the episode ends.

Given these, the shared module that implements `observation_batch` and `act_batch` needs from each server only two small hooks, for composing the magnified contact sheet and for encoding the preview image.

F.2 Shared episode lifecycle

Evaluation service and runner. An evaluation service owns the evaluation slots and their environment servers, and an evaluation runner drives the evaluation on top of it. The runner reads the task roster of the selected simulator from the service's catalog, divides the tasks round-robin over the available slots with one worker thread per slot, and runs each task through a driver. For each initial scene the driver starts the slot's environment server for the task, waits until it is ready, loads the scene, and starts an experiment. The initial-scene selector is a layout identifier on RoboDojo, taken from pre-sampled whitelists of stable layouts; on the other simulators, whose servers advertise no layout option, it is sent as the episode seed, using pre-sampled stable seeds where they exist and seeds $0, \ldots, K-1$ otherwise. The service then starts the episode on the environment server, with video recording options, and collects an initial workspace observation (instruction, arm state, and images) before the first model turn. These setup reads are made by the runtime on the raw client, so they are not counted as model decisions. The skills are placed into the system prompt, and the agent runner is launched against the server.

Budget. The budget is enforced at two levels. The driver polls the experiment every $20\,\mathrm{s}$ and stops it when the number of model responses exceeds the round cap, which counts as a failure, or when the wall-clock limit is reached; the runner defaults to 180 minutes per task. Inside the environment server, RoboTwin, RoboLab, LIBERO, and the real robot additionally cap the number of motion commands per episode (60 by default): every action call, including a whole path, counts as one command, and exhausting the cap ends the episode as a failure with reason `max_steps`. RoboDojo imposes no command cap; its own step limit is raised so that the episode never truncates.

Success and termination. Success is taken only from the platform's judge. The simulator adapters poll the simulator's own success check while a command executes and latch the first success; the episode then ends, further actions are refused with `episode_finished`, and the reply that fired it carries the success flag and a request to stop the agent. When the episode is ended without a latched success, RoboTwin and LIBERO take a final verdict on the terminal state. On RoboDojo and RoboTwin the success criteria of the task are appended to the instruction. On the real robot the server never reports success, and the episode is judged by the operator (Appendix G).

Retries and logging. A run that ends through an infrastructure fault (a failed model API call, a runner error, or an environment that did not start) is re-run with linear back-off, up to five times by default; a clean failure, including one that exhausts the round cap, is recorded once and never re-run. Each episode writes a run directory with the experiment configuration, the metadata, every model message and tool call, the outcome with the token usage, the returned images, and the video of the recorded cameras. The driver keeps a resumable state file per task, so an interrupted evaluation continues without re-running finished episodes.

Table 9: **Platform adapters.** Surfaces as in Table 8; "Default surface" is the surface served when no surface is selected, and the command cap is the default per-episode limit on motion commands.

	RoboDojo	RoboTwin	RoboLab	LIBERO	Real robot
Backend	Isaac Sim / Isaac Lab task	SAPIEN task class	Isaac Lab task, DROID config	robosuite (MuJoCo) off-screen env	ARX X5 SDK and RealSense
Arms	`left`, `right`	`left`, `right`	`arm`	`arm`	`left`, `right`
Cameras	head, two wrist	head, left, right, front	over-shoulder, wrist, viewport	`agentview`, eye-in-hand	head, two wrist
World frame	env frame, z up	SAPIEN world, z up	env-local, z up	MuJoCo world, z up	left-arm base
Calibration	computed from camera optics and prim pose	simulator camera config	sensor data	robosuite camera utilities	calibration files, hand-eye
Grasp-point motion	cuRobo plan, joint waypoints	mplib plan, dense replay	differential IK servo	OSC delta servo	Cartesian controller, walked steps
Reachability	cuRobo IK	mplib planning	reach annulus	FK-sampled box	shell, IK, arm separation
Success judge	reward-manager gates	`check_success`	termination signal	BDDL goal check	operator
Default surface	batch	granular	granular	granular	batch
Joint reset (`act_batch`)	reset to start	refused	refused	refused	refused
Command cap	none	60	60	60	60

F.3 Platform adapters

Table 9 summarizes the adapters. All of them serve the conventions of Appendix E; they differ in how a grasp-point command becomes motion, how reachability is decided, and which tool surface they serve.

Tool surface served. The RoboDojo and real-robot servers publish all three surfaces of Table 8 and serve the batch surface by default. The RoboTwin, RoboLab, and LIBERO servers implement `observation_batch` and `act_batch` through the shared module but publish only two surfaces, the minimal surface or the granular surface, and serve the granular surface whenever the minimal one is not selected. On these three simulators the geometry is therefore offered through `unproject_pixels`, the two-camera `triangulate_pixels`, and `check_tcp_pose_reachable`, next to the discrete reads and actions.

RoboDojo. The adapter builds a RoboDojo evaluation environment with the ARX X5 configuration and a single environment instance, replaces the policy client with a stub, and drives the task directly. The arms are the robots of the scene's robot manager, named `left` and `right`. The grasp point is measured at start-up from the finger bodies and lies ahead of the flange along the approach axis, while IK is solved for the flange, so every command is converted from grasp point to flange before it reaches the solver. Camera names are read from the scene at every reset (`cam_head`, `cam_left_wrist`, `cam_right_wrist` in our tasks), and the resolution is read from the camera. The intrinsics are computed from the camera optics as $f_x = f_y = Wf/h$ with $c_x = W/2$ and $c_y = H/2$, where f is the focal length and h the horizontal aperture, because the renderer ignores the vertical aperture that the simulator's own intrinsic matrix uses. The extrinsic is the camera prim's world pose relative to the environment origin, converted from the USD to the OpenCV camera axes. An absolute command first passes a cuRobo IK check, which refuses an unsolvable pose before anything moves (`ik_unreachable`); the move is then planned with the cuRobo planner, the planned joint path is thinned to waypoints at most $0.20\,\mathrm{rad}$ apart ($0.05\,\mathrm{rad}$ while holding an object), and each waypoint is driven as a joint target, followed by a Cartesian settle at the goal. Without a plan, the adapter falls back to straight-line Cartesian interpolation in small steps. Each target is held for at most 25 simulator actions and ends as `converged` (within $3\,\mathrm{mm}$ and $3°$), `arm_stalled` or `gripper_blocked` (progress below a threshold), `max_steps`, or `episode_end`. While one arm moves, the other is held at its current pose and gripper value. The reachability query runs the same cuRobo IK check on the action a real command would send, with the other arm held, and simulates no path. The joint reset drives the arm back to the joint configuration recorded at the start of the episode. Success comes from RoboDojo's reward-manager gates, re-armed after every reset and read after every simulator action. The initial scene is a pre-generated evaluation layout selected by

identifier, and a reset that settles into an unstable scene is reported as an error rather than started. The server also publishes the nominal table-top height from the scene configuration in its description.

RoboTwin. The adapter loads the RoboTwin task class and its configuration files as RoboTwin's own policy evaluation does, with a single dual-arm embodiment, and builds each scene with the task's demo setup in test mode. Camera names are read from the scene (`head_camera`, `left_camera`, `right_camera`, and `front_camera`, which is not recorded to video), and intrinsics and extrinsics come from the simulator's camera configuration after a render refresh, with the world-to-camera extrinsic inverted to camera-to-world. A grasp-point target is shifted $0.12\,\mathrm{m}$ back along its approach axis to the planner's reference point and planned with RoboTwin's mplib planner; a failed plan is refused before motion, and a successful joint trajectory is replayed step by step, followed by a short settle. The gripper is driven along a planned gripper trajectory only when its command changes, so a squeeze is held across later motions. The reachability query calls the same planner from the current joint state for each candidate and executes nothing. Joint actions are time-parameterized along the joint-space segment. Success is RoboTwin's `check_success`, polled every ten control steps during replay, during settling, and after each path leg. At reset, with the expert check on by default, the adapter walks seeds from the requested one and keeps the first scene that RoboTwin's scripted expert solves, trying at most 20, and then rebuilds that scene fresh for the agent; the instruction is drawn from RoboTwin's generated episode descriptions.

RoboLab. The adapter registers the RoboLab task with the DROID robot (Franka Panda with a Robotiq 2F-85), its wrist and over-the-shoulder cameras, and an egocentric viewport, and replaces the action space: the arm uses Isaac Lab's differential IK on the Robotiq base flange with absolute pose commands, and the finger uses a continuous joint-position target instead of RoboLab's binary gripper action. A grasp-point target is converted to a flange pose by a fixed offset and sent as one absolute IK target, and the environment steps until the grasp point converges ($1\,\mathrm{cm}$ and $5°$), stops improving over ten steps, or uses 60 steps; there is no motion planner. Camera names and resolutions are read from the live observation groups, intrinsics from the sensor data (or from the pinhole model of the camera configuration when absent), and the extrinsic from the sensor pose relative to the environment origin. The reachability query is a geometric screen: the flange point must lie between $0.10\,\mathrm{m}$ and $0.95\,\mathrm{m}$ from the arm base. The same screen runs before every grasp-point command, so a pose outside it is refused before motion, while a pose inside it that cannot be reached returns `arm_stalled`. Success is RoboLab's termination signal, latched after every environment step, and a truncation counts as failure. The task and scene are fixed per process, and an episode is seeded rather than chosen from a layout table.

LIBERO. The adapter builds LIBERO's off-screen environment for one task of a suite with the default operational-space controller, and serves the Panda as a single arm `arm`, with the robosuite end-effector site as the grasp point. The two cameras are `agentview` and `robot0_eye_in_hand` at 256×256 by default; frames are flipped vertically once to undo the renderer's row order, and the intrinsics and camera-to-world extrinsics come from robosuite's camera utilities, which already use the OpenCV axes. A grasp-point command is executed by closed-loop servoing without IK or planning: at each $20\,\mathrm{Hz}$ control step the adapter measures the end effector, converts the remaining position error and the rotation vector of $R_{\mathrm{target}} R^{\top}$ into a controller delta scaled by the controller's action scale and clipped to $[-1, 1]$, and stops at $1\,\mathrm{cm}$ and $5°$, at the end of the episode, or after 20 steps, reporting an unreachable target as `arm_stalled` after a partial approach. LIBERO's gripper command is binary, so the adapter drives it by sign in pulses until the measured opening is within tolerance or the fingers stop, which is reported as `gripper_blocked`. LIBERO has no joint control, and joint positions are read-only. The reachability query tests the position against an axis-aligned box measured at start-up from 256 random joint configurations through forward kinematics, and does not judge orientation. Success is the task's BDDL goal check, which LIBERO reports on every step, with a final check when the episode ends. The initial scene is LIBERO's fixed initial state indexed by the seed, followed by a few settling steps with the gripper open.

Real robot. The real-robot server runs the same protocol and the batch surface. It controls each ARX X5 arm through the vendor's Cartesian controller, reads the three RealSense cameras, and loads the camera calibration from files; a wrist camera's extrinsic is its calibrated pose in the last link composed with the live pose of that link. Absolute moves are walked in bounded steps, the reach-

ability query runs the same workspace-shell, 6-DoF IK, and arm-separation checks as an action, and success is judged by the operator; these and the gripper handling are described in Appendix G. Like the other non-RoboDojo platforms, it refuses `target=joint`, and it reports camera extrinsic poses with Euler angles rather than rotation vectors.

G REAL-ROBOT SETUP

Hardware and sensing. The platform is a dual-arm ARX X5: two 6-DoF arms, each with a position-controlled parallel gripper, mounted side by side in front of a tabletop. Three Intel RealSense D405 cameras provide the observations: a fixed head camera looking down at the table and one camera on each wrist, all streaming 640×480 RGB. Intrinsics are calibrated once and shared across the three cameras; the head extrinsics are calibrated for the current mounting, and the wrist extrinsics are given in the frame of the last link. All poses the agent reads or commands are expressed in a single world frame, the base of the left arm. The robot runs the same environment server contract as the simulators, so every scaffold reaches it through exactly the interface it uses in simulation; the scaffold runs on a separate machine and talks to the server over the network.

Safety guards. The guards concern only the robot itself and encode no scene geometry. Before any motion, a command must pass a workspace-shell check, a 6-DoF inverse kinematics check, and a check that keeps each gripper at least $0.12\,\mathrm{m}$ from the other arm's grasp point; a refused command returns the reason and the arm state to the agent without moving the hardware. The reachability tool `check_tcp_pose_reachable` runs the same checks, so a pose it accepts is a pose the executor accepts. An absolute move is executed in steps of at most $5\,\mathrm{cm}$ and $0.35\,\mathrm{rad}$ ($2\,\mathrm{cm}$ and $0.15\,\mathrm{rad}$ while holding an object), each step checked separately; if a step stalls, the remaining steps are dropped and the controller is re-targeted to the pose actually reached rather than left pressing. The gripper, which has no force control, closes at $20\,\mathrm{mm/s}$ with each command clamped to within $4\,\mathrm{mm}$ of the measured finger position, and it freezes its hold once the fingers stop on an object. We deliberately impose no lower bound on the height of the end effector. The table height is exactly the quantity the skills ask the agent to measure by probing or triangulation, and a floor guard would leak it: even if the bound were never stated, the agent could recover it by submitting poses and observing which are refused. What keeps the arm off the table is the stepwise execution and stall handling above, together with an operator who holds the physical emergency stop throughout every episode.

Tasks. The five tasks and their instructions are listed in Table 6. They are chosen to probe what the simulators render poorly or not at all. *Stack cylinder* requires placing one rigid object on another with little margin. *Doll to plate* and *Cylinder to plate* share a referring expression (the plate closer to the water cup) but differ in the object: the soft doll deforms and slips under a grasp that holds the rigid cylinder. *Wipe table* is contact-rich: the agent must press a towel onto the table and slide it across a stain, which requires finding the table surface by probing rather than by reading it from a model. *Open drawer* chains an articulated interaction with a placement into the opened drawer.

Episode protocol. Each episode starts with both arms at their home configuration and the scene arranged to the task's pre-registered layout. The episode ends when the agent declares completion or its budget runs out. As in RoboDojo, completion includes returning each arm to the pose it held at the start of the episode, after releasing the object and retreating along the approach direction. The budget caps the number of motion commands the executor accepts, the number of tool rounds, and the wall-clock time. Episodes are blocked by layout and session, and the order of configurations is randomized within a block, so that drift in lighting, calibration, or hardware state does not align with a single scaffold. Each task is run five times per configuration, giving 25 episodes per configuration.

Success judgment. There is no simulator evaluator on the real robot, and the environment never reports success itself. An on-site operator judges each episode against the pre-registered observable success condition of its task, and the video of all three cameras is retained for review.

Real-robot results. Real-robot results on these tasks are reported on our anonymous project website: `https://anonymous.4open.science/w/RoCS-F1CF/`. Appendix C.5 follows one *Cylinder to plate* episode step by step.