CODE-DRIVEN EXPLORATION IN REAL-TO-SIM DIGITAL TWINS FOR MULTI-ARM MANIPULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

General-purpose coding agents can now write, debug, and refine entire robot programs, but only if they have somewhere to practice. Physical hardware is a poor practice ground: exploratory rollouts are slow to reset and, when several arms share a workspace, unsafe. Generic simulation benchmarks are safe but do not match the geometry, assets, or viewpoints of the workspace where the robot must actually work. We present **CoEx** (**Co**de-driven **Ex**ploration), which lets the agent practice in an *executable digital twin of its own deployment workspace*. The twin is built by **CoEnv**, our real-to-sim framework, which reconstructs the physical scene from multi-view RGBD into a physics simulation. Inside it, a centralized code agent explores at the *program level*, writing a complete multi-arm program, running it in the twin, and refining it against grounded verification (privileged state checks, checkpoint predicates on risky steps, and adaptive viewpoints against inter-arm occlusion). Verified programs transfer to hardware through collision-checked deployment. On five real-world multi-arm tasks with up to three heterogeneous robots, CoEx reaches 70% average success, against 50% for the same system without the code agent (where a VLM must choose every primitive step by step), outperforms two prior VLM-planning frameworks, and produces verified episodes at 3.8–11.7$\times$ the ablated rate. Because every verified rollout is also a demonstration, exploration leaves reusable experience behind: a $\pi_{0.5}$ policy fine-tuned purely on 200 agent-generated rollouts reaches 100% and 70% real-robot success on two tasks, with no human demonstrations.

1 INTRODUCTION

Visuomotor policies are bounded by the supply of demonstrations, and nowhere more severely than in multi-arm manipulation, where a single teleoperated demonstration can occupy several synchronized operators and recent systems still keep a human in the loop for every episode (Lin et al., 2026; Mandi et al., 2024). The standard answer is simulation: one line multiplies a few human demonstrations into a large simulated corpus (Mandlekar et al., 2023; Jiang et al., 2025; Qin et al., 2026), another co-trains on simulated and real data (Maddukuri et al., 2025), a third reconstructs the scene and learns inside it (Torne et al., 2024; Dai et al., 2024). All of them place the simulator upstream of the *data* (Fig. 1a,b), with two consequences: the policy learns from simulated experience, so a sim-to-real gap has to be bridged; and the behavior being simulated still originates from a human, as seed demonstrations or as a reward. The bottleneck is displaced rather than removed.

Coding agents suggest a different route. Vision-language and code-generation models decompose tasks, write control programs, inspect outcomes, and correct their own mistakes (Liang et al., 2023; Zhou et al., 2025; Zawalski et al., 2024; Huang et al., 2024), and general-purpose coding agents now write, debug, and refine entire robot programs (Tsui et al., 2026; Elmaaroufi et al., 2026; Lu et al., 2026; Xiao et al., 2026). An agent that can solve a multi-arm task on its own needs neither a demonstration nor a reward. But the edit–run–debug loop that makes such agents effective needs somewhere to run the program, see what went wrong, and try again, cheaply and often. Hardware is neither: an agent that retries spends its budget on resets, risks collisions between arms sharing a workspace, and cannot see past arms that occlude one another. Generic simulation benchmarks are safe but do not match the geometry, assets, or camera views of the deployment workspace, so programs debugged there do not transfer.

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.

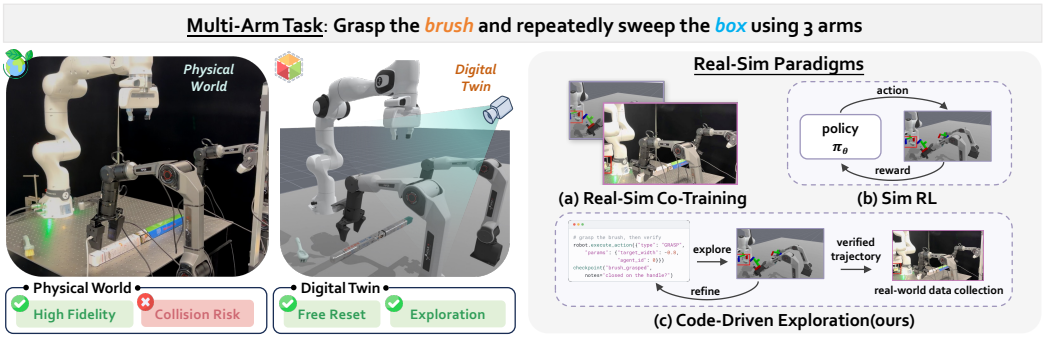

Figure 1: **Where does the simulator sit?** *Left:* we reconstruct the multi-arm workcell into an executable digital twin. The real cell is faithful, but every exploratory attempt risks a collision; the twin resets freely and can be explored at no cost. *Right:* three ways of using that twin. **(a)** co-training mixes simulated trajectories into the training set alongside a few real ones, and **(b)** sim RL optimizes a policy π_θ against simulated rollouts and a reward. Both place the simulator upstream of the *data*, so the policy learns from simulated experience and a sim-to-real gap has to be bridged. **(c)** CoEx places it upstream of the *program*: given only a task description, the code agent writes a multi-arm program, explores and refines it in the twin until it verifies, and the verified trajectory is then executed on the real robot, so the data collected afterwards is real.

Our key idea is to give the agent an *executable digital twin of its own workspace* (Fig. 1). We reconstruct the specific physical scene (objects, poses, robot placements, camera calibration) into a physics simulation (real-to-sim), in which a coding agent can run programs, render arbitrary viewpoints, query privileged state, and verify outcomes at zero physical risk. The twin plays the role of a world model in model-based learning (Hafner et al., 2023; Bruce et al., 2024), an action-conditioned predictor in which behavior is tried before it is executed, but it is *reconstructed rather than learned*: dynamics come from a physics engine and state is explicit. This is deliberate: debugging a program requires ground-truth state (poses, contacts, swept volumes) that tells the agent *what* went wrong, which learned video world models do not expose. Because the twin is metrically aligned with the real workcell, verified programs transfer back to hardware through collision-checked deployment. The twin is therefore used to find a *program*, never to produce training images: once a program is verified, re-running it on the real robot turns one round of exploration into a corpus of real demonstrations, collected without a teleoperator and with no gap left to bridge (Fig. 1c).

We instantiate this idea as **CoEx**, code-driven exploration hosted by **CoEnv**, a real-to-sim framework that builds the twin and returns verified programs to hardware. *(i) Digital twin construction* converts multi-view RGBD observations into a simulator-ready scene via generated 3D assets, open-vocabulary segmentation with VLM disambiguation, 6-DoF pose fusion, and iterative camera calibration (Section 3.1). *(ii) Code-driven exploration (CoEx)*: a centralized code agent (one model controlling several embodiments, not decentralized multi-agent autonomy) writes a complete multi-arm program, runs it in the twin, and refines it against grounded verification, supported by twin-side tools for inter-arm occlusion and risky preconditions; as a baseline we remove the code agent (CoEx w/o code agent), leaving a VLM to choose every primitive step by step on the same twin (Section 3.2). *(iii) Collision-aware deployment* admits trajectories to hardware only after swept-volume checks across all arm pairs (Section 3.3). *(iv) Experience reuse*: verified rollouts persist as exemplars for future planning and as supervision for visuomotor policies (Section 3.4).

We evaluate CoEx on five real-world multi-arm tasks with heterogeneous embodiments. CoEx reaches 70% average real-world success (up to 90%), ahead of the same system without the code agent (50%) on all five tasks and of two prior VLM-planning frameworks, at $3.8\times$–$11.7\times$ the ablated system's verified-episode throughput; once a CoEx program clears its first milestone it completes the task in 88% of trials, against 57% without the code agent; the twin's adaptive-camera and checkpoint tools prove individually indispensable ($1.7\times$ and $2.5\times$ over their ablations); and no arm–arm collision occurred in 100 deployed trajectories. As a closing study, a $\pi_{0.5}$ policy fine-tuned purely on 200 CoEx rollouts reaches 100% and 70% real-robot success on two tasks with no human demonstrations, up from 80% and 40% with 50 rollouts.

Our main contributions are:

- **A different place for the simulator**: it searches for a *program* rather than generating data, so multi-arm demonstrations can be collected on the real robot with no teleoperator, no reward, and no sim-to-real gap in the training set.
- **CoEx**, code-driven exploration in which a code agent writes, runs, and refines complete multi-arm programs against grounded verification, supported by twin-side tools for inter-arm occlusion (adaptive camera control) and risky preconditions (checkpoint predicates), and compared against the same system without the code agent (CoEx w/o code agent).
- **Real-world validation** on five multi-arm tasks with up to three heterogeneous robots via collision-checked deployment, with comparisons to prior frameworks, tool ablations, and failure analysis.
- **Evidence that the resulting data is what matters**: policies fine-tuned on CoEx rollouts beat those trained on human teleoperation and, at matched size, on rollouts recorded in the twin instead of on the robot.

2 Related Work

Agentic planning and coding agents for robots. Foundation models plan robot behavior through code generation (Liang et al., 2023; Li et al., 2024), execution monitoring (Zhou et al., 2025), keypoint constraints (Huang et al., 2024), chain-of-thought action reasoning (Zawalski et al., 2024; Zhen et al., 2024), and simulation-grounded planning (Liu et al., 2026). A fast-moving line hands the task to general-purpose *coding agents*, which solve benchmark manipulation from privileged state (Tsui et al., 2026), search policy repositories from execution feedback (Elmaaroufi et al., 2026), compound validated fixes into skill libraries (Lu et al., 2026), and run policy improvement on robot fleets (Xiao et al., 2026). CoEx belongs to this line, and practices in a twin of the *actual deployment workspace* rather than in benchmark scenes, with verification built for multi-arm occlusion and contact. LLM coordination has also been extended to multiple robots through dialog or role assignment (Mandi et al., 2024; Singh et al., 2025; Qin et al., 2025; Gong et al., 2024; Palma et al., 2026), largely on textual abstractions without geometric grounding (Gerkey & Matarić, 2004; Liu et al., 2022). CoEx instead uses a deliberately *centralized* planner over all arms; multi-arm manipulation is our testbed, not a claim of decentralized autonomy.

Simulation, digital twins, and where the simulator sits. Because teleoperation does not scale, much work uses simulation to supply training data: multiplying a few demonstrations into thousands of simulated trajectories (Mandlekar et al., 2023; Jiang et al., 2025; Qin et al., 2026), co-training a large simulated corpus with a small real dataset (Maddukuri et al., 2025), or reconstructing the deployment scene, learning inside it, and transferring with domain randomization (Torne et al., 2024; Dai et al., 2024; Lou et al., 2026; Zhu et al., 2025). All place the simulator upstream of the *data* (Fig. 1; Appendix A tabulates them), so a sim-to-real gap must be bridged and the simulated behavior still originates from a human, as seed demonstrations or as a reward. Both bind hardest with multiple arms, where one demonstration occupies several synchronized operators and recent systems keep a human in the loop for every episode (Lin et al., 2026). CoEx places the simulator upstream of the *program* instead: the agent finds and verifies a multi-arm program with neither demonstrations nor rewards, and that program then runs on the real robot, so the data it leaves behind is real. Hence our twin is *reconstructed rather than learned*: world models predict observations from data (Hafner et al., 2023; Bruce et al., 2024; Yang et al., 2024) but expose no privileged state against which success can be checked, whereas workspace-aligned physics simulators (Todorov et al., 2012; Mittal et al., 2025; Tao et al., 2024; Mu et al., 2021; Nasiriany et al., 2024; Chen et al., 2023; Wan et al., 2025; Chen et al., 2025) give the ground-truth poses, contacts, and swept volumes that tell the agent *what* went wrong.

Agent-generated robot data. Foundation models also generate robot training data by synthesizing tasks and scenes (Wang et al., 2024c;b), designing rewards (Ma et al., 2024a;b), collecting autonomously at fleet scale (Ahn et al., 2024), and distilling LLM-guided rollouts into policies (Ha et al., 2023; Duan et al., 2024); language agents likewise improve by accumulating skills and reflections (Wang et al., 2024a; Shinn et al., 2023), and robot foundation models by autonomous practice (Ghasemipour et al., 2025). Demonstration quantity drives policy performance (Lin et al.,

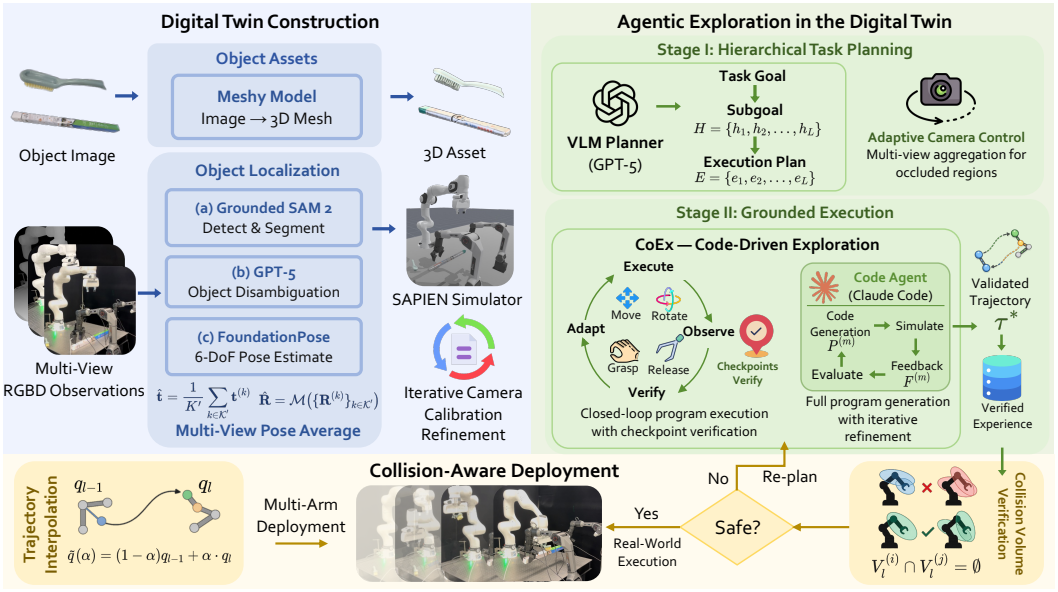

Figure 2: **The CoEnv framework. Top-left:** Digital twin construction converts multi-view RGBD into an executable simulation. **Top-right:** Agentic exploration: after hierarchical planning, CoEx couples a program-level loop, in which a code agent generates, simulates, and refines a complete multi-arm program, with the step-level execute–observe–verify–adapt loop that running the program closes over primitives and checkpoints; validated trajectories are stored as verified experience $\mathcal{D}$. **Bottom:** Collision-aware deployment via interpolation with swept-volume verification.

2025), so such generation is increasingly attractive. In CoEx, data is a by-product rather than the goal: the rollouts the agent verifies while solving tasks are themselves demonstrations.

3 Method

CoEnv couples a reconstructed digital twin with code-driven agentic exploration and collision-checked deployment; verified rollouts persist as reusable experience (Fig. 2). We denote the twin $\mathcal{W} = (\mathcal{S}, f, \textsc{Render})$, comprising the structured state space $\mathcal{S}$, the physics transition function $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$, and a rendering function over arbitrary camera poses.

3.1 Digital Twin Construction from Real Scenes

From K calibrated RGBD views $\mathbf{o}_t$ we build a structured simulator state $s_t = \Phi(\mathbf{o}_t)$. Simulation-ready meshes are generated from a few reference images per object and paired with convex collision proxies and per-category physical attributes; Grounded SAM2 (Ren et al., 2024) segments the objects, GPT-5.6 Sol (OpenAI, 2026) disambiguates visually similar instances, and FoundationPose (Wen et al., 2024) gives a 6-DoF pose per view, which we fuse across views by averaging translations and quaternions (Markley et al., 2007). We implement Φ in ManiSkill (Tao et al., 2024) on SAPIEN (Xiang et al., 2020), whose contact resolution is stable at the millimeter scale, and refine camera extrinsics by rendering and comparing against the real capture until reprojection error stabilizes (Appendix C.1). What matters downstream is not photorealism but that $\mathcal{W}$ is resettable, queryable, and metrically aligned with the workcell: the agent debugs against ground-truth state, and the training images come from the real cameras, never from $\mathcal{W}$.

3.2 Code-Driven Exploration with Grounded Verification

Given state $\mathbf{s}_t$, a centralized planner synthesizes a joint action $\mathbf{a}_t = [a_t^{(1)}, \ldots, a_t^{(N)}]$ for N arms evolving under f. Given a goal $\mathcal{G}$, exploration proceeds in two stages (Fig. 2, top-right): *hierarchical planning* decomposes $\mathcal{G}$ into a structured plan, and *grounded execution* realizes it through Code-driven Exploration (CoEx), in which a program is written, run, and refined against verification. Both

Algorithm 1 CoEx: code-driven exploration in the digital twin

Require: Twin $\mathcal{W}$ with initial state $\mathbf{s}_0$, goal $\mathcal{G}$, round budget $M_{\max}$, experience set $\mathcal{D}$
Ensure: Verified program $\mathcal{P}^*$ and trajectory τ^*

$\hat{o}_0 \leftarrow \textsc{Aggregate}(\{\textsc{Render}(\mathbf{s}_0, c_j)\}_{j=1}^{J})$ ▷ adaptive multi-view observation
$\mathcal{H} \leftarrow \textsc{Decompose}(\mathcal{G}, \hat{o}_0)$; $\mathcal{E} \leftarrow \textsc{Assign}(\mathcal{H}, \mathbf{s}_0)$ ▷ Stage I: hierarchical plan
$\mathcal{F}^{(0)} \leftarrow \textsc{Retrieve}(\mathcal{G}, \mathbf{s}_0, \mathcal{D})$ ▷ hints from past verified episodes
for $m = 1, \ldots, M_{\max}$ **do**
  $\mathcal{P}^{(m)} \leftarrow \textsc{CodeGen}(\mathbf{s}_0, \mathcal{E}, \mathcal{F}^{(m-1)})$ ▷ complete multi-arm program
  $\tau^{(m)} \leftarrow \textsc{Execute}(\mathcal{P}^{(m)}, \mathcal{W})$ ▷ free reset; checkpoints snapshot state and views
  if $\textsc{Verified}(\tau^{(m)}, \mathcal{G})$ **then** $\mathcal{P}^* \leftarrow \mathcal{P}^{(m)}$, $\tau^* \leftarrow \tau^{(m)}$; **break**
  else $\mathcal{F}^{(m)} \leftarrow \textsc{Analyze}(\tau^{(m)}, \text{logs}, \text{checkpoints})$ ▷ structured failure feedback
  end if
end for
$\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_0, \mathcal{G}, \tau^*)\}$; re-execute $\mathcal{P}^*$ from new initial states to add further verified rollouts
Deploy τ^* to hardware after swept-volume collision checks (Section 3.3)

```
# privileged state of the twin
robot.get_robot_state(robot_id)  -> {position, orientation, euler_rpy, gripper}
robot.get_object_state(name)     -> {position, orientation}
# primitives via robot.execute_action(a); agent_id: int or list (synchronized arms)
a = {"type": "MOVE",    "params": {delta_x, delta_y, delta_z, agent_id}}
a = {"type": "ROTATE",  "params": {delta_yaw, delta_pitch, delta_roll, agent_id}}
a = {"type": "GRASP",   "params": {target_width, agent_id}}
a = {"type": "RELEASE", "params": {agent_id}}
# grounded verification: snapshot state + multi-view renderings at a critical step
checkpoint(name, checkpoint_type, notes, suggested_views)
```

Figure 3: **Interface exposed to CoEx in the twin** (abridged from Appendix F.3). Programs read privileged state, issue delta-pose primitives to one or several arms at once, and place checkpoints whose snapshots drive failure analysis.

exploit affordances unavailable in the physical world: free resets, arbitrary rendered viewpoints, and privileged state queries.

Stage I: Hierarchical task planning. A VLM planner (GPT-5.6 Sol) decomposes the goal into a two-level structure $\mathcal{G} \to \mathcal{H} = \{h_l\}_{l=1}^{L} \to \mathcal{E} = \{e_l\}_{l=1}^{L}$, where each h_l is a semantic sub-goal (e.g., "pick up the red cube") and each $e_l = (i_l, \rho_l, \mathbf{p}_l)$ assigns arm i_l an action primitive $\rho_l \in \{\textsc{Move}, \textsc{Rotate}, \textsc{Grasp}, \textsc{Release}\}$ with parameters $\mathbf{p}_l$, executed via inverse kinematics.

Stage II: Grounded execution with CoEx. We expose the primitive library as a code interface (Fig. 3) and perform **Co**de-driven **Ex**ploration (CoEx), which couples the two loops drawn in Fig. 2 (Algorithm 1). The *outer, program-level loop* belongs to a code agent (Claude Code; Anthropic, 2025): it generates a complete program $\mathcal{P}^{(m)} = \textsc{CodeGen}(\mathbf{s}_0, \mathcal{E}, \mathcal{F}^{(m-1)})$ encoding the full multi-arm execution logic, simulates it in $\mathcal{W}$ to obtain $\tau^{(m)}$, evaluates the outcome, and turns failures into structured feedback $\mathcal{F}^{(m)}$ (collision locations, unachieved sub-goals) that conditions the next program, up to $M_{\max}$ rounds. The *inner, step-level loop* is what simulating a program means: the program *executes* primitives on one or several arms, *observes* privileged state and rendered views, *verifies* outcomes and checkpoint predicates at critical junctures, and *adapts* the following motions to what it observed; the logs and checkpoint snapshots it leaves behind are exactly what the outer loop evaluates. Because each round resets $\mathcal{W}$ for free, the agent can afford genuine trial-and-error: the twin converts exploration from a physical liability into cheap test-time search. And because a verified program is a reusable artifact, re-executing it from new initial states yields further verified rollouts without further deliberation, so exploration doubles as a data engine (Section 4.5).

Why the program is the unit of exploration. The two loops differ in what one act of deliberation commits to. A step-level agent commits to a single primitive, observes, and deliberates again, so an episode costs one model call per step and the reasoning evaporates when the episode ends. A program-level agent commits to an entire multi-arm execution: deliberation is amortized over the whole task, a failure is diagnosed against the program that produced it rather than against an isolated step, and what survives the episode is source code. That artifact is why exploration compounds

here. It can be re-executed from new initial configurations without further deliberation, retrieved as an exemplar when a related goal appears, and read by a person before it ever reaches hardware.

Grounded verification tools. The twin exposes three tools to any agent acting in it. A verifier $v_l = V(e_l, \mathbf{s}_t, \mathbf{s}_{t'}) \in \{\textsc{Success}, \textsc{Fail}\}$ checks each outcome against privileged state. *Checkpoints* e_l^{CKPT} are non-action steps at critical junctures (e.g., gripper alignment before a grasp) that snapshot state and multi-view renderings and evaluate a predicate $\phi(e_l^{\text{CKPT}}, \mathbf{s}_t) \in \{0, 1\}$. And because multi-arm workspaces are frequently occluded by the arms themselves, *adaptive camera control* queries multiple poses $\{c_j\}_{j=1}^{J}$ and aggregates renderings, $\hat{o}_t = \textsc{Aggregate}(\{\textsc{Render}(\mathbf{s}_t, c_j)\}_{j=1}^{J})$, to expose occluded regions.

Ablated baseline (CoEx w/o code agent). To measure what the program-level loop buys, we remove the code agent: CoEx w/o code agent keeps the same twin, primitives, and tools but runs the inner loop alone, with a VLM (GPT-5.6 Sol) choosing every primitive step by step: it re-parameterizes e_l or inserts corrective steps when $v_l = \textsc{Fail}$, and proceeds past a checkpoint only when $\phi = 1$. This design reacts within an episode, but pays for deliberation at every primitive and leaves no reusable artifact.

3.3 Collision-Aware Real-World Deployment

We bridge the kinematics gap between $\mathcal{W}$ and physical robots via *trajectory interpolation* and *collision volume verification*. For each arm i, we record joint configurations $\mathbf{q}_l^{(i)} \in \mathbb{R}^d$ at the completion of every primitive in simulation, and densify the trajectory by linear interpolation $\tilde{\mathbf{q}}^{(i)}(\alpha) = (1-\alpha)\mathbf{q}_{l-1}^{(i)} + \alpha\,\mathbf{q}_l^{(i)}$, discretized into S uniform steps for the real-world controller. Before each interpolated action executes, a forward-kinematics pass derives the swept collision volume $\mathcal{V}_l^{(i)}$ for every arm from its link geometries along the interpolated path. An action is admitted only if $\mathcal{V}_l^{(i)} \cap \mathcal{V}_l^{(j)} = \emptyset$ for all $i \neq j$; otherwise it is discarded and re-planning is triggered from the current state. This pre-execution check keeps multi-arm operation collision-free without overly conservative motion constraints.

3.4 Reusing Verified Experience

Exploration leaves a by-product: every verified rollout enters an experience set ("Verified Experience" in Fig. 2) $\mathcal{D} = \{(\mathbf{s}_0^{(j)}, \mathcal{G}^{(j)}, \tau^{(j)})\}_{j=1}^{|\mathcal{D}|}$, which can be reused in two ways. *As exemplars:* the framework stores every verified episode, and when CoEx faces goal $\mathcal{G}$ in state $\mathbf{s}_0$ it receives relevant past experience as in-context hints (strategies and grasp parameters that survived verification, and feedback from failed rounds). This mechanism is active in all our experiments; we do not ablate it separately. *As supervision:* each verified trajectory unrolls into pairs $\{(o_t, \mathbf{a}_t)\}_{t=1}^{T}$, where o_t comprises the camera images and proprioception recorded while the verified program executes, and we fine-tune a visuomotor policy π_θ by behavior cloning on action chunks,

$$\mathcal{L}_{\text{BC}}(\theta) = \mathbb{E}_{(o_t,\, \mathbf{a}_{t:t+H}) \sim \mathcal{D}} \big[\, \ell\big(\pi_\theta(o_t),\, \mathbf{a}_{t:t+H}\big)\big], \tag{1}$$

where ℓ is the policy's native imitation loss and H the chunk horizon. Unlike the agent, whose programs query privileged simulator state, π_θ acts from raw observations alone and replaces per-episode deliberation with a forward pass. We test this use of the data by fine-tuning $\pi_{0.5}$ (Physical Intelligence et al., 2025) in a small study (Section 4.5); re-verifying policy rollouts with the same tools to grow $\mathcal{D}$ further is a natural extension we leave to future work.

4 Experiments

Our experiments answer four questions. **Q1:** Does code-driven exploration in the twin solve real multi-arm tasks, and how does it compare with step-wise deliberation (Section 4.2)? **Q2:** How does CoEx compare with prior agentic planning frameworks (Section 4.3)? **Q3:** Which tools make exploration reliable, and where does it fail (Section 4.4)? **Q4:** Is the verified experience that exploration leaves behind reusable (Section 4.5)?

Table 1: **Real-world multi-arm evaluation.** 10 trials per agent per task. S_i: i-th subtask milestone (x/10). SR: task success. Bold: higher SR per task.

	CoEx (ours)				CoEx w/o code agent (baseline)			
Task	S_1	S_2	S_3	SR	S_1	S_2	S_3	SR
Two-Arm (Franka × 2)								
Cube Stacking	10/10	9/10	–	**9/10**	7/10	6/10	–	6/10
Ball Pickup	6/10	6/10	–	**6/10**	9/10	4/10	–	4/10
Transfer Cylinder	8/10	6/10	6/10	**6/10**	9/10	4/10	4/10	4/10
Three-Arm (Franka + AgileX Piper × 2)								
Place Cucumber	8/10	8/10	6/10	**6/10**	9/10	7/10	4/10	4/10
Brush Box	8/10	8/10	8/10	**8/10**	10/10	9/10	7/10	7/10
Average	80%	74%	67%	**70%**	88%	60%	50%	50%

4.1 SETUP

Hardware and tasks. We evaluate on five multi-arm manipulation tasks in two workcell configurations (Fig. 7; full milestone definitions in Appendix B). *Two-arm* (Franka Research 3 ×2): **Cube Stacking**, **Ball Pickup** (bimanual lift), and **Transfer Cylinder** (handover). *Three-arm* (Franka + AgileX Piper dual-arm): **Place Cucumber** (lid lifting + insertion) and **Brush Box** (coordinated sweeping). Robot bases are fixed; object poses vary across trials. Each workcell uses 2–3 calibrated RealSense D435i cameras, and scenes are reconstructed in ManiSkill (Section 3.1).

Agents and protocols. CoEx is driven by Claude Code as the code agent, with a budget of $M_{\max}$=10 refinement rounds per task. The CoEx w/o code agent baseline shares the twin, primitives, and verification tools but lets GPT-5.6 Sol choose each primitive, so the two differ in backbone model as well as in loop structure. Real-world evaluations use 10 trials per agent per task. We report subtask milestones S_i and task success rate (SR).

4.2 REAL-WORLD MULTI-ARM EVALUATION (Q1)

Table 1 reports real-robot results. CoEx reaches 70% average SR, against 50% without the code agent, and is ahead on all five tasks, peaking at 90% on *Cube Stacking*; with 10 trials per cell, per-task gaps are best read as trends. The more telling pattern is *where* each system loses trials. The step-wise ablation clears the first milestone more often (88% vs. 80%) but sheds successes at every later stage, finishing only 25 of the 44 trials that pass S_1 (57%). CoEx finishes 35 of 40 (88%): every *Brush Box* trial that grasps the brush completes the sweep (8/8/8), and the two three-stage tasks that involve a handover or a held lid keep 6 of 8. Two thirds of CoEx's failures (10 of 15) occur before the first milestone: on *Ball Pickup*, all four failures happen at the initial bimanual contact, and every trial that establishes it completes the lift. A program that has been verified end to end in the twin encodes the whole multi-arm sequence coherently, so it either fails early, at the first grasp, or carries the task through, whereas per-step decisions let small errors compound along the horizon. Fig. 4 shows the correspondence between twin rollouts and physical execution, and no arm–arm collision occurred across all 100 deployed trajectories.

4.3 COMPARISON WITH PRIOR AGENTIC PLANNING FRAMEWORKS (Q2)

Fig. 5(a) compares against two prior VLM-planning frameworks and our CoEx w/o code agent baseline on three representative tasks. Code-as-Monitor (Zhou et al., 2025) performs execution-time failure detection without sim-grounded planning; SimPACT (Liu et al., 2026) grounds planning in simulation but lacks an iterative verify–refine loop. CoEx reaches 77% on these tasks, against 57% without the code agent and 20% and 40% for the prior frameworks. The gap is widest on the *Transfer Cylinder* handover, where both prior frameworks collapse (1/10 and 0/10) while CoEx completes 6/10. The two fail for complementary reasons: Code-as-Monitor detects a failure only once it has happened on hardware, and SIMPACT commits to the first plan its simulator accepts. CoEx meets the same failures inside the twin, where retrying is free.

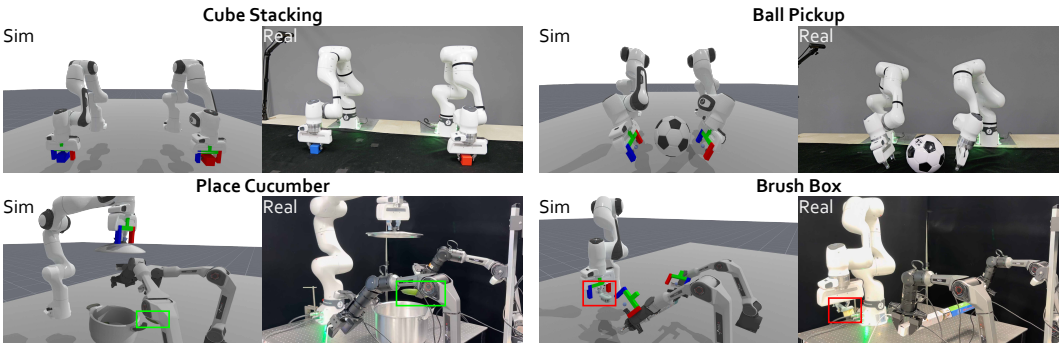

Figure 4: **Digital twin to real world.** Planned twin rollouts beside real executions on four tasks.

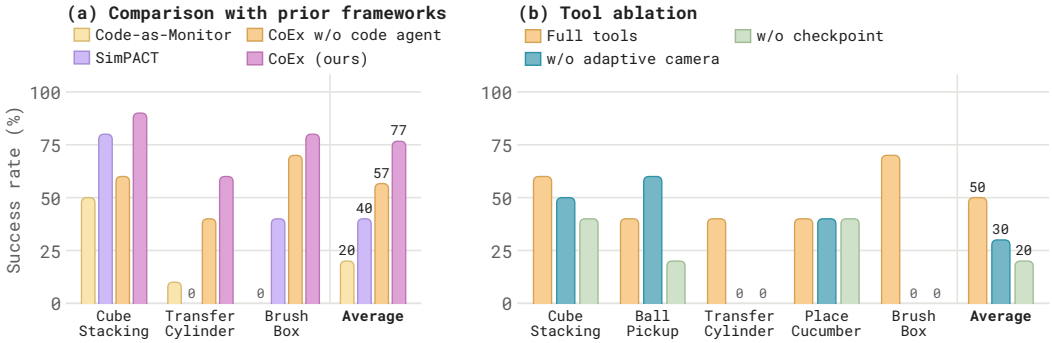

Figure 5: **Real-world success rates** (10 trials per task). **(a)** CoEx and our CoEx w/o code agent baseline against two prior VLM-planning frameworks on three representative tasks (easy two-arm, reactive two-arm handover, three-arm). **(b)** Removing either twin-side tool from CoEx w/o code agent. Per-task counts are tabulated in Appendix C.4.

4.4 What Makes Exploration Reliable, and Where It Fails (Q3)

Tool ablation. Fig. 5(b) isolates the two twin-side tools on CoEx w/o code agent, where each tool's effect on individual decisions is directly observable. Removing **adaptive camera control** drops average SR from 50% to 30%, concentrated on occlusion-heavy tasks (*Transfer Cylinder* and *Brush Box* both fall to 0/10) where acting arms obstruct the contact region from any fixed viewpoint. Removing **checkpoint verification** is more damaging (50%→20%): without precondition gating, early errors compound through long horizons. The two address complementary failure modes of exploration: *observability* and *precondition reliability*.

Failure analysis. Failures fall into three recurring modes: residual sim-to-real pose offsets at grasp or placement, re-planning that stagnates on a failed strategy, and agent-specific limitations (drift over long step-wise sequences, and handovers that a single program must anticipate). The most direct remedies are a residual correction at contact and a hybrid that hands reactive sub-steps to step-wise control while the rest of the task stays at the program level; Appendix E discusses representative cases.

4.5 Reusing Verified Experience (Q4)

Throughput. Table 2 reports verified episodes per agent session: CoEx yields 17.5 and 9.5 on *Cube Stacking* and *Brush Box*, 11.7× and 3.8× the CoEx w/o code agent baseline (1.5 and 2.5), because one verified program amortizes reasoning across many rollouts, whereas a step-wise agent pays per primitive. Twin resets are instantaneous.

A small distillation study. To test whether these rollouts are useful supervision, we collect them in a closed loop (Fig. 6): once a program is verified, the frozen agent re-issues it from each new initial configuration, so a whole session reuses one round of exploration (scene resets were manual here;

Table 2: **Reusable experience.** *Left:* verified episodes per agent session. *Right:* real-robot SR (10 trials) of a $\pi_{0.5}$ policy fine-tuned on human teleoperated demonstrations, on those demonstrations co-trained with 200 rollouts of the same verified program recorded *in the twin*, or on 50 and 200 rollouts of that program recorded *on the robot*. Bold: best per row.

	Verified episodes / session		**$\pi_{0.5}$ policy SR by training data**			
Task	w/o code agent	CoEx	20 teleop.	200 in twin + 20 teleop.	50 CoEx	200 CoEx
Cube Stacking	1.5	**17.5**	4/10	3/10	8/10	**10/10**
Brush Box	2.5	**9.5**	2/10	0/10	4/10	**7/10**

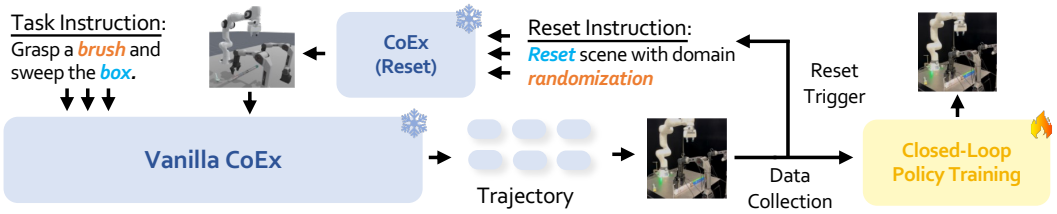

Figure 6: **Closed-loop data collection.** The agent solves the task (*Vanilla CoEx*) and its trajectory is recorded; after a reset, a reset instruction makes the same frozen agent re-issue the task from the new configuration (*CoEx (Reset)*), so the same program is reused across episodes. The recorded rollouts fine-tune a visuomotor policy.

Appendix D). We fine-tune a $\pi_{0.5}$ policy (Physical Intelligence et al., 2025) on 50 and on 200 CoEx rollouts recorded on the real robot, against two references: 20 teleoperated demonstrations, and the standard co-training recipe of pairing those demonstrations with 200 rollouts of the *same verified program* recorded in the twin rather than on the robot. Success grows steadily with the number of CoEx rollouts, reaching 10/10 and 7/10 at 200 (Table 2). Co-training on the simulated rollouts does not help here: at 3/10 and 0/10 it falls below the teleoperated demonstrations alone. With the program, the episode count and the evaluation protocol held fixed, the only thing separating that column from the last one is where the rollouts were recorded. The gap is consistent with what the policy consumes: $\pi_{0.5}$ acts from camera images, and rendered views differ from the real cameras in texture, lighting, and sensor noise, so simulated episodes train the visuomotor mapping on a distribution it never meets at test time. The program crosses that gap because it is written in terms of object poses; the pixels do not.

5 Conclusion and Limitations

CoEx, hosted by the CoEnv framework, gives a coding agent what it needs to program real multi-arm systems: an executable digital twin of its own deployment workspace in which to write, run, and debug programs, grounded verification tools that tell it what went wrong, and a collision-checked path back to hardware. Real-robot experiments on five multi-arm tasks validate the pipeline against prior frameworks and against the same system without its code agent, and a small distillation study shows that the verified rollouts exploration leaves behind already suffice to fine-tune a visuomotor policy, and that recording them on the robot rather than in the twin is what makes them useful, so the agent's practice is not wasted. More broadly, the bottleneck is less the agent than the environment it can practice in: given a twin with inspectable state, an off-the-shelf code agent already programs heterogeneous multi-arm systems, and the data it leaves behind needs no sim-to-real bridge.

Limitations. The twin covers rigid objects with pre-generated assets, fixed bases, and static scenes, and coordination is centralized, so we claim nothing about decentralized autonomy or scaling beyond three arms. Real-world execution is open-loop replay, so residual pose offsets remain a main source of failure. Our evaluation is modest in scale, and CoEx and its ablation differ in backbone model as well as in loop structure.

Reproducibility Statement

We describe the full system in Section 3 and Appendix C: the perception models used for digital twin construction, the simulator, the foundation models, the primitive and verification interface exposed to the code agent (Fig. 3), and the hardware and control stack. The complete prompt structures of CoEx and of the ablated baseline are reproduced in Appendix F, task and milestone definitions in Appendix B, the evaluation protocol in Section 4.1, and the data collection and policy fine-tuning details of the distillation study in Appendix D. Per-task counts behind every figure are tabulated in Appendix C.4. We will release our code, including the twin construction pipeline, the agent interface, and all prompts, upon publication. Because the agents rely on proprietary foundation models (GPT-5.6 Sol and Claude Code) whose behavior may change across versions, we will document the model versions used in our experiments together with the code.

Ethics Statement

This work studies tabletop manipulation with fixed-base robot arms in a controlled laboratory setting. It involves no human-subject experiments and no personal or sensitive data. Letting a language-model agent write programs that move physical robots raises a safety concern, which our design addresses directly: every program is first executed and verified in a digital twin, and a trajectory is admitted to hardware only after a pre-execution swept-volume collision check. These mechanisms reduce but do not remove risk, so all real-robot experiments were run under human supervision with standard physical safeguards (workspace barriers and emergency stops), and we recommend the same for any deployment of this kind of system. We foresee no negative societal impact specific to this method beyond those common to robot automation in general.

AI Use Statement

Foundation models are components of the method studied in this paper: GPT-5.6 Sol serves as the task planner, disambiguates object instances during digital twin construction, and drives the ablated baseline (CoEx w/o code agent), and Claude Code is the code agent inside CoEx (Section 3). Their prompts are reproduced in Appendix F, and the rollouts used in the distillation study are, by design, generated by these agents. Beyond the method itself, we used an LLM-based assistant as a writing aid during manuscript preparation: for grammar checking and language polishing, including help with tightening and reorganizing the text and feedback on its presentation, for verifying the bibliography against the original sources, for cross-checking the consistency of reported numbers between tables, figures, and prose, and for writing the plotting script of Fig. 5. The research ideas, the system design and implementation, all experiments, and the analysis and interpretation of the results are the work of the authors. All AI-assisted text was reviewed and revised by the authors, who take full responsibility for the content of this paper; no AI system is listed as an author.

References

AgileX Robotics. Piper SDK. `https://github.com/agilexrobotics/piper_sdk`, 2024.

Michael Ahn, Debidatta Dwibedi, Chelsea Finn, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Karol Hausman, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, et al. AutoRT: Embodied foundation models for large scale orchestration of robotic agents. *arXiv preprint arXiv:2401.12963*, 2024.

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

Jake Bruce, Michael Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative interactive environments. In *International Conference on Machine Learning (ICML)*, 2024.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

Ziyang Chen, Aldo Marzullo, Davide Alberti, Elena Lievore, Matteo Fontana, Ottavio De Cobelli, Gennaro Musi, Giancarlo Ferrigno, and Elena De Momi. FRSR: Framework for real-time scene reconstruction in robot-assisted minimally invasive surgery. *Computers in Biology and Medicine*, 163:107121, 2023.

Tianyuan Dai, Josiah Wong, Yunfan Jiang, Chen Wang, Cem Gokmen, Ruohan Zhang, Jiajun Wu, and Li Fei-Fei. Automated creation of digital cousins for robust policy learning. *Conference on Robot Learning (CoRL)*, 2024.

Jiafei Duan, Wentao Yuan, Wilbert Pumacay, Yi Ru Wang, Kiana Ehsani, Dieter Fox, and Ranjay Krishna. Manipulate-Anything: Automating real-world robots using vision-language models. In *Conference on Robot Learning (CoRL)*, 2024.

Karim Elmaaroufi, Justin Svegliato, Sarunas Kalade, Graham Schelle, Sanjit A. Seshia, and Matei Zaharia. RHO: Your coding agent is secretly a roboticist. *arXiv preprint arXiv:2606.16458*, 2026.

Brian P Gerkey and Maja J Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International journal of robotics research*, 23(9):939–954, 2004.

Seyed Kamyar Seyed Ghasemipour, Ayzaan Wahid, Jonathan Tompson, Pannag Sanketi, and Igor Mordatch. Self-improving embodied foundation models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Ran Gong, Qiuyuan Huang, Xiaojian Ma, Yusuke Noda, Zane Durante, Zilong Zheng, Demetri Terzopoulos, Li Fei-Fei, Jianfeng Gao, and Hoi Vo. MindAgent: Emergent gaming interaction. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 3154–3183, 2024.

Huy Ha, Pete Florence, and Shuran Song. Scaling up and distilling down: Language-guided robot skill acquisition. In *Conference on Robot Learning (CoRL)*, 2023.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. ReKep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. *Conference on Robot Learning (CoRL)*, 2024.

Zhenyu Jiang, Yuqi Xie, Kevin Lin, Zhenjia Xu, Weikang Wan, Ajay Mandlekar, Linxi Fan, and Yuke Zhu. DexMimicGen: Automated data generation for bimanual dexterous manipulation via imitation learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

Jingyao Li, Pengguang Chen, Sitong Wu, Chuanyang Zheng, Hong Xu, and Jiaya Jia. RoboCoder: Robotic learning from basic skills to general tasks with large language models. *arXiv preprint arXiv:2406.03757*, 2024.

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 *2023 IEEE International conference on robotics and automation (ICRA)*, pp. 9493–9500. IEEE, 2023.

Fanqi Lin, Yingdong Hu, Pingyue Sheng, Chuan Wen, Jiacheng You, and Yang Gao. Data scaling laws in imitation learning for robotic manipulation. In *International Conference on Learning Representations (ICLR)*, 2025.

Zesen Lin, Jian-Jian Jiang, Haoming Cen, Xiao-Ming Wu, Dandan Zhang, and Wei-Shi Zheng. HATS: A human-agent teleoperation system for multi-arm data collection. *arXiv preprint arXiv:2606.16491*, 2026.

Haowen Liu, Shaoxiong Yao, Haonan Chen, Jiawei Gao, Jiayuan Mao, Jia-Bin Huang, and Yilun Du. SIMPACT: Simulation-enabled action planning using vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Xinzhu Liu, Xinghang Li, Di Guo, Sinan Tan, Huaping Liu, and Fuchun Sun. Embodied multi-agent task planning from ambiguous instruction. In *Robotics: Science and Systems*, pp. 1–14, 2022.

Haozhe Lou, Mingtong Zhang, Haoran Geng, Hanyang Zhou, Sicheng He, Zhiyuan Gao, Siheng Zhao, Jiageng Mao, Pieter Abbeel, Jitendra Malik, Daniel Seita, and Yue Wang. D-REX: Differentiable real-to-sim-to-real engine for learning dexterous grasping. In *International Conference on Learning Representations (ICLR)*, 2026.

Runyu Lu, Yubo Wu, Ethan Kou, Letian Fu, Wenli Xiao, Ajay Mandlekar, Yinzhen Xu, Guanya Shi, Ken Goldberg, Ang Chen, Mosharaf Chowdhury, Yuke Zhu, Linxi Fan, and Guanzhi Wang. ASPIRE: Agentic skills discovery for robotics. *arXiv preprint arXiv:2607.00272*, 2026.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations (ICLR)*, 2024a.

Yecheng Jason Ma, William Liang, Hung-Ju Wang, Sam Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. DrEureka: Language model guided sim-to-real transfer. In *Robotics: Science and Systems (RSS)*, 2024b.

Abhiram Maddukuri, Zhenyu Jiang, Lawrence Yunliang Chen, Soroush Nasiriany, Yuqi Xie, Yu Fang, Wenqi Huang, Zu Wang, Zhenjia Xu, Nikita Chernyadev, Scott Reed, Ken Goldberg, Ajay Mandlekar, Linxi Fan, and Yuke Zhu. Sim-and-real co-training: A simple recipe for vision-based robotic manipulation. In *Robotics: Science and Systems (RSS)*, 2025.

Zhao Mandi, Shreeya Jain, and Shuran Song. RoCo: Dialectic multi-robot collaboration with large language models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 286–299. IEEE, 2024.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. MimicGen: A data generation system for scalable robot learning using human demonstrations. In *Conference on Robot Learning (CoRL)*, 2023.

F Landis Markley, Yang Cheng, John L Crassidis, and Yaakov Oshman. Averaging quaternions. *Journal of Guidance, Control, and Dynamics*, 30(4):1193–1197, 2007.

Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, Octi Zhang, Peter Du, Antonio Serrano-Munoz, Xinjie Yao, René Zurbrügg, Nikita Rudin, et al. Isaac Lab: A GPU-accelerated simulation framework for multi-modal robot learning. *arXiv preprint arXiv:2511.04831*, 2025.

Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. ManiSkill: Generalizable manipulation skill benchmark with large-scale demonstrations. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2021.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. RoboCasa: Large-scale simulation of everyday tasks for generalist robots. In *Robotics: Science and Systems (RSS)*, 2024.

OpenAI. GPT-5.6 preview system card. `https://deploymentsafety.openai.com/gpt-5-6-preview`, 2026. Released 2026-06-25; covers GPT-5.6 Sol, Terra, and Luna.

Alessio Palma, Indro Spinelli, Vignesh Prasad, Luca Scofano, Yufeng Jin, Georgia Chalvatzaki, and Fabio Galasso. Bimanual robot manipulation via multi-agent in-context learning. *arXiv preprint arXiv:2604.20348*, 2026.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Yiran Qin, Li Kang, Xiufeng Song, Zhenfei Yin, Xiaohong Liu, Xihui Liu, Ruimao Zhang, and Lei Bai. RoboFactory: Exploring embodied agent collaboration with compositional constraints. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10075–10085, 2025.

Yiran Qin, Jiahua Ma, Li Kang, Wenzhan Li, Yihang Jiao, Xin Wen, Xiufeng Song, Heng Zhou, Jiwen Yu, Zhenfei Yin, Xihui Liu, Philip Torr, Yilun Du, and Ruimao Zhang. ComSim: Building scalable real-world robot data generation via compositional simulation. *arXiv preprint arXiv:2604.11386*, 2026.

Tianhe Ren, Shilong Liu, Ailing Zeng, Jing Lin, Kunchang Li, He Cao, Jiayu Chen, Xinyu Huang, Yukang Chen, Feng Yan, et al. Grounded SAM: Assembling open-world models for diverse visual tasks. *arXiv preprint arXiv:2401.14159*, 2024.

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.

Harsh Singh, Rocktim Jyoti Das, Mingfei Han, Preslav Nakov, and Ivan Laptev. MALMM: Multi-agent large language models for zero-shot robotic manipulation. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 20386–20393. IEEE, 2025.

Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse-kai Chan, et al. ManiSkill3: GPU parallelized robotics simulation and rendering for generalizable embodied AI. *arXiv preprint arXiv:2410.00425*, 2024.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033. IEEE, 2012.

Marcel Torne, Anthony Simeonov, Zechu Li, April Chan, Tao Chen, Abhishek Gupta, and Pulkit Agrawal. Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation. In *Robotics: Science and Systems (RSS)*, 2024.

Brian Y. Tsui, Alan Y. Fang, and Tiffany J. Hwu. Demonstration-free robotic control via LLM agents. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2026.

Weikang Wan, Jiawei Fu, Xiaodi Yuan, Yifeng Zhu, and Hao Su. LodeStar: Long-horizon dexterity via synthetic data augmentation from human demonstrations. In *9th Annual Conference on Robot Learning*, 2025.

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 (TMLR)*, 2024a.

Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. GenSim: Generating robotic simulation tasks via large language models. In *International Conference on Learning Representations (ICLR)*, 2024b.

Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. RoboGen: Towards unleashing infinite data for automated robot learning via generative simulation. In *International Conference on Machine Learning (ICML)*, 2024c.

Bowen Wen, Wei Yang, Jan Kautz, and Stan Birchfield. FoundationPose: Unified 6D pose estimation and tracking of novel objects. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 17868–17879, 2024.

Philipp Wu, Yide Shentu, Zhongke Yi, Xingyu Lin, and Pieter Abbeel. GELLO: A general, low-cost, and intuitive teleoperation framework for robot manipulators. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. SAPIEN: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11097–11107, 2020.

Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, Jimmy Wu, Guanzhi Wang, S. Shankar Sastry, Ken Goldberg, Linxi Fan, Yuke Zhu, and Guanya Shi. ENPIRE: Agentic robot policy self-improvement in the real world. In *Conference on Robot Learning (CoRL)*, 2026.

Sherry Yang, Yilun Du, Kamyar Ghasemipour, Jonathan Tompson, Leslie Kaelbling, Dale Schuurmans, and Pieter Abbeel. Learning interactive real-world simulators. In *International Conference on Learning Representations (ICLR)*, 2024.

Michał Zawalski, William Chen, Karl Pertsch, Oier Mees, Chelsea Finn, and Sergey Levine. Robotic control via embodied chain-of-thought reasoning. *Conference on Robot Learning (CoRL)*, 2024.

Haoyu Zhen, Xiaowen Qiu, Peihao Chen, Jincheng Yang, Xin Yan, Yilun Du, Yining Hong, and Chuang Gan. 3D-VLA: A 3D vision-language-action generative world model. *International Conference on Machine Learning (ICML)*, 2024.

Enshen Zhou, Qi Su, Cheng Chi, Zhizheng Zhang, Zhongyuan Wang, Tiejun Huang, Lu Sheng, and He Wang. Code-as-Monitor: Constraint-aware visual programming for reactive and proactive robotic failure detection. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 6919–6929, 2025.

Shaoting Zhu, Linzhan Mou, Derun Li, Baijun Ye, Runhan Huang, and Hang Zhao. VR-Robo: A real-to-sim-to-real framework for visual robot navigation and locomotion. *IEEE Robotics and Automation Letters*, 2025.

Yifeng Zhu, Abhishek Joshi, Peter Stone, and Yuke Zhu. VIOLA: Imitation learning for vision-based manipulation with object proposal priors. In *Conference on Robot Learning (CoRL)*, 2022.

A Positioning with Respect to Prior Data Pipelines

Table 3 expands Fig. 1. The column that matters is *what a human must supply per task*: every prior pipeline needs either demonstrations or a reward, because the simulator can only replay or optimize behavior that something else has specified. CoEx supplies that behavior itself, which is why the human input reduces to a task description, and why the simulator is never asked to produce a training image.

Table 3: **Where the simulator sits in sim-assisted manipulation data pipelines.** "Policy trains on" refers to the data the deployed policy is fitted to. Rows: MimicGen (Mandlekar et al., 2023), DexMimicGen (Jiang et al., 2025), ComSim (Qin et al., 2026), sim-and-real co-training (Maddukuri et al., 2025), RialTo (Torne et al., 2024), ACDC (Dai et al., 2024), HATS (Lin et al., 2026).

Method	Human input per task	What the simulator does	Policy trains on	Arms
MimicGen	∼10 source demos	replays and adapts them	sim	1–2
DexMimicGen	∼5 source demos	replays and adapts them	sim	2
ComSim	10 real demos	classical sim + neural rendering	real + pseudo-real	2
Co-training	10–30 real demos	generates a large corpus	sim + real (tuned ratio)	1–2
RialTo	demos + success condition	environment for RL	sim (+ randomization)	1
ACDC	demos or reward	generates digital cousins	sim	1
HATS	an operator every episode	not used	real	4
CoEx (ours)	**a task description**	**searches for and verifies a program**	**real**	**2–3**

B Task Descriptions

Fig. 7 shows keyframes from successful real-world executions of all five tasks. Table 4 summarizes the five evaluation tasks and their objectives. Table 5 details the ordered subtask milestones (S_i) used for evaluation. Each milestone must be satisfied in order; the task succeeds only when the final milestone is achieved.

Table 4: **Task descriptions.** Each task involves 2–3 robotic arms.

Task	**Description**
Two-Arm (Franka × 2)	
Cube Stacking	Two arms stack a blue cube and a red cube separately.
Ball Pickup	Bimanual coordination to lift a soccer ball with balanced contact.
Transfer Cylinder	Pick up a cylinder, perform a bimanual handover and place it at the target location.
Three-Arm (Franka + AgileX Piper × 2)	
Place Cucumber	Pick up the pot lid and place the cucumbers into the pot.
Brush Box	Grasp a brush and repeatedly sweep the box.

C Implementation Details

Algorithm 1 in the main text formalizes CoEx. The CoEx w/o code agent baseline shares Stage I and replaces the program loop (lines 4–9) with a per-primitive *execute* → *observe* → *verify* → *adapt* loop, described in Section C.3.2.

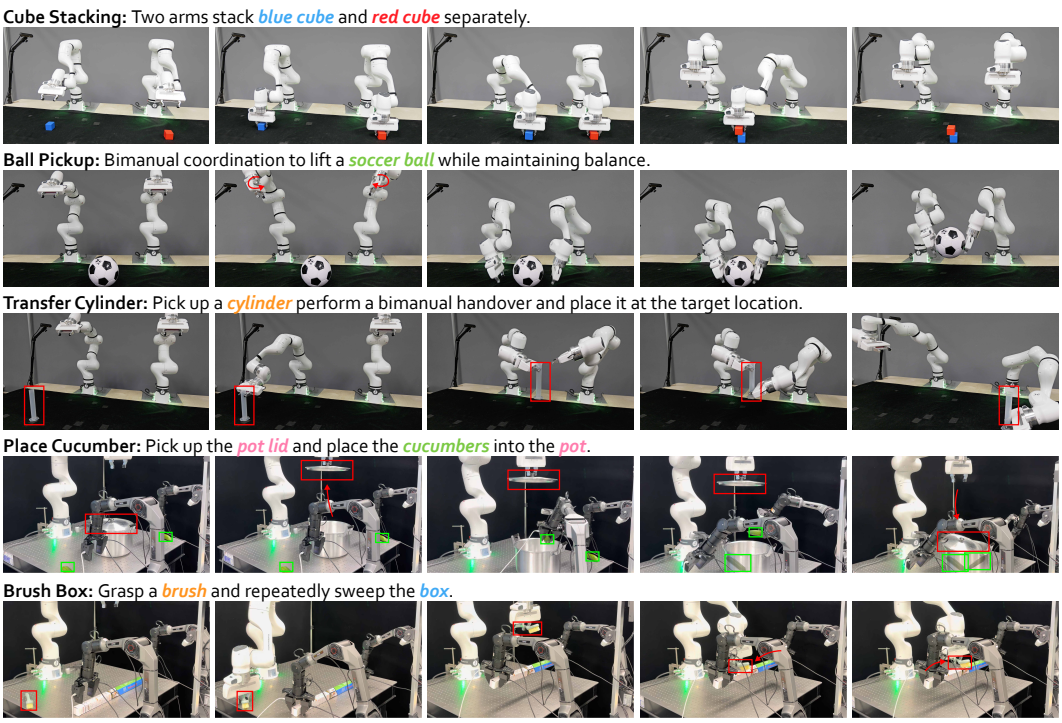

Figure 7: **Task suite.** Five multi-arm tasks with increasing coordination complexity. *Top three rows:* two-arm tasks (Franka $\times 2$): Cube Stacking, Ball Pickup, Transfer Cylinder. *Bottom two rows:* three-arm tasks (Franka + AgileX Piper): Place Cucumber, Brush Box. Each row shows keyframes from a successful real-world execution.

Table 5: **Subtask milestones and success criteria.** The last milestone of each task defines the overall success condition.

Task	Milestone	Criterion
Cube Stacking	S_1	Both cubes are successfully grasped and lifted.
	S_2	The cubes are stably stacked.
Ball Pickup	S_1	Both arms reach the ball with contact poses.
	S_2	The ball is successfully lifted off the surface.
Transfer Cylinder	S_1	The cylinder is grasped and lifted by the first arm.
	S_2	The second arm successfully receives the cylinder.
	S_3	The cylinder is placed at the target location.
Place Cucumber	S_1	Pot lid is opened and held stably.
	S_2	Two cucumbers are picked up.
	S_3	Both cucumbers are placed inside the pot.
Brush Box	S_1	The brush is successfully grasped.
	S_2	The box is successfully grasped.
	S_3	The brush contacts the box in a sweeping motion.

C.1 Digital Twin Construction

3D asset generation. Rather than relying on noisy photogrammetric meshes, we generate simulation-ready meshes from a few reference images per object using Meshy Model[1], scale-normalized to real dimensions, paired with low-polygon convex collision proxies, and assigned per-category physical attributes (mass, friction); authoring is one-shot per object class.

[1] https://www.meshy.ai/

Multi-view object localization. For each view $o_t^{(k)}$, Grounded SAM2 (Ren et al., 2024) detects and segments target objects, and GPT-5.6 Sol (OpenAI, 2026) disambiguates visually similar instances using spatial context (e.g., two same-shape cubes, or a brush partially occluded by a gripper). FoundationPose (Wen et al., 2024) then produces a 6-DoF pose $p^{(k)} = (\mathbf{t}^{(k)}, \mathbf{R}^{(k)})$ per view; we fuse poses across the K' detecting views $\mathcal{K}'$ as

$$\hat{\mathbf{t}} = \tfrac{1}{K'} \sum_{k \in \mathcal{K}'} \mathbf{t}^{(k)}, \qquad \hat{\mathbf{R}} = \mathcal{M}\big(\{\mathbf{R}^{(k)}\}_{k \in \mathcal{K}'}\big), \tag{2}$$

arithmetically for translations and via quaternion averaging (Markley et al., 2007) on $SO(3)$ for rotations, mitigating single-view occlusions and small extrinsic biases.

Iterative camera calibration. Exploiting the simulator's reset-ability, we refine camera extrinsics iteratively, rendering the current scene and comparing it to the real capture until reprojection error stabilizes at the millimeter range.

C.2 Hardware and Robot Control

Robot platforms. We use two robot platforms: the Franka Research 3 (7-DoF) and the AgileX Piper (6-DoF, dual-arm configuration). Both robots operate in joint position control mode. The Franka is controlled via Deoxys (Zhu et al., 2022), which provides a modular real-time control interface over the `libfranka` communication layer; we adopt its default interpolation scheme that generates smooth joint-space trajectories between waypoints. The AgileX Piper is controlled via the official Piper SDK (AgileX Robotics, 2024); since the SDK provides only raw joint position commands, we implement linear interpolation between consecutive waypoints to ensure smooth and safe motion execution.

Perception setup. We deploy 2 calibrated Intel RealSense D435i cameras for the two-arm setting (Franka $\times$ 2) and 3 cameras for the three-arm setting (Franka + AgileX Piper $\times$ 2). All cameras are mounted at fixed positions around the workspace and provide synchronized RGBD streams for multi-view scene reconstruction (Section 3.1).

C.3 Agentic Exploration in the Digital Twin

C.3.1 Shared Infrastructure.

Both agents share a common action and perception layer built atop ManiSkill (Tao et al., 2024). We define four delta-based action primitives, namely Move, Rotate, Grasp, and Release, all specified relative to the current end-effector pose to eliminate the need for global coordinate calibration. Each primitive accepts an arm index (`agent_id` in the prompts) that can be a single integer or a list for synchronized multi-arm execution. A state query API exposes the TCP pose, gripper aperture, and 6-DoF object poses, providing the numerical grounding for both agents. A unified controller dispatches primitives to robot-specific IK solvers for Franka Research 3 (7-DoF) and AgileX Piper (6-DoF). A virtual camera module renders 1920×1080 images from arbitrary viewpoints around the scene centroid.

C.3.2 CoEx w/o code agent: Step-wise VLM Baseline.

CoEx w/o code agent implements the closed-loop *execute* $\rightarrow$ *observe* $\rightarrow$ *verify* $\rightarrow$ *adapt* cycle using GPT-5.6 Sol (OpenAI, 2026) as the VLM. Execution proceeds in two phases.

Planning phase. The VLM performs multi-round visual analysis (up to $J_{\max}$=6 rounds), receiving a rendered scene image, structured state data, and a system prompt encoding shared manipulation knowledge (see Sec. F). It may request additional viewpoints via `CAMERA_ORBIT` to observe the scene from multiple perspectives, implementing the adaptive multi-view aggregation of Section 3.2. The phase concludes with `PLANNING_COMPLETE` and three structured outputs: *key observations* (spatial findings carried forward as persistent context), *checkpoints* (steps requiring visual verification with recommended camera angles), and an *execution plan* (action sequence with robot assignments).

Table 6: **Design comparison of the CoEx w/o code agent baseline and CoEx.**

Aspect	CoEx w/o code agent	CoEx (ours)
Foundation model	GPT-5.6 Sol (VLM)	Claude Code (code agent)
Decision granularity	One action per VLM call	Full program per iteration
Feedback signal	Real-time images + state	Post-hoc checkpoints + state
API calls / task	$\sim$30+ VLM calls	$\sim$1–5 code generations
Error correction	Adjust next action online	Rewrite code offline
Planning	Multi-round visual analysis	Structured system prompt
Verification	Checkpoint + visual + numerical	Checkpoint images + stdout logs
Advantage	Reactive, adaptive coordination	Precise, coherent trajectories

Execution phase. Given the plan $\mathcal{E} = \{e_1, \ldots, e_L\}$, each step follows an *observe–act–verify* loop: the VLM receives the current image and state, outputs structured reasoning followed by one or more actions. At checkpoint steps, the system enforces position, orientation, and visual verification before proceeding. Corrective actions are generated automatically when checks fail. To improve robustness, the system employs stuck pattern detection (injecting hints when repetitive low-magnitude actions are detected), post-action drift correction, and idle-arm stabilization for multi-arm tasks.

C.3.3 CoEx: Code-Driven Exploration.

CoEx implements the code generation and refinement loop of Section 3.2, using Claude Code (Anthropic, 2025) as the code agent. Rather than issuing one action at a time, the agent generates a *complete* Python program encoding the entire multi-arm execution logic.

System prompt and sandbox. The code agent receives a structured system prompt covering the task specification, API reference, manipulation knowledge base, collision avoidance rules, and a task-type-specific code template. Generated code runs in a sandboxed environment with a restricted namespace and timeout enforcement, ensuring reproducibility. A checkpoint function enables the code to capture multi-view images and state snapshots at critical execution points.

Iterative refinement. Each iteration m resets the simulation to the same initial state and executes the generated program. Post-execution, the code agent analyzes execution logs and checkpoint images to identify root causes of failure (e.g., unreachable targets, failed grasps), then refines the code accordingly. This process repeats for up to $M_{\max}$=10 iterations. Each iteration is *stateless*: the environment resets from scratch and the code must be self-contained.

C.3.4 Comparison of the Two Agents.

Table 6 summarizes the key design differences. In the real-robot results (Table 1), CoEx is ahead on all five tasks. What separates the two designs most clearly is cost and reusability (Section 4.5) and how success is retained across milestones (Section 4.2).

C.4 Per-Task Counts behind Fig. 5

Tables 7 and 8 give the per-task success counts (and, for the tool ablation, the subtask milestones) plotted in Fig. 5.

D Experience Reuse: Distillation Study Details

Fig. 6 in the main text illustrates how CoEx turns exploration into data: a program verified in the twin is executed on the robot, the scene is reset between episodes, and the recorded rollouts are stored for policy training. This section gives the protocol of the study in Section 4.5.

Table 7: **Comparison with prior VLM-planning frameworks** and our CoEx w/o code agent baseline on three representative tasks (easy two-arm, reactive two-arm handover, three-arm). SR (x/10); bold: best per task.

Task	Code-as-Monitor	SimPACT	CoEx w/o code agent	**CoEx**
Cube Stacking	5/10	8/10	6/10	**9/10**
Transfer Cylinder	1/10	0/10	4/10	**6/10**
Brush Box	0/10	4/10	7/10	**8/10**
Average	20%	40%	57%	**77%**

Table 8: **Tool ablation (CoEx w/o code agent):** removing adaptive camera control or checkpoint verification.

	w/o Adaptive Camera				**w/o Checkpoint Verif.**			
Task	S_1	S_2	S_3	SR	S_1	S_2	S_3	SR
Cube Stacking	6/10	5/10	–	5/10	4/10	4/10	–	4/10
Ball Pickup	10/10	6/10	–	6/10	10/10	2/10	–	2/10
Transfer Cylinder	6/10	2/10	0/10	0/10	6/10	0/10	0/10	0/10
Place Cucumber	9/10	6/10	4/10	4/10	6/10	6/10	4/10	4/10
Brush Box	10/10	8/10	0/10	0/10	9/10	8/10	0/10	0/10
Avg.	82%	54%	13%	30%	70%	40%	13%	20%
Full tools	88%	60%	50%	**50%**	88%	60%	50%	**50%**

D.1 Demonstration Collection Protocol

For each of the two tasks (*Cube Stacking* and *Brush Box*) we collected 200 episodes on the real robot with CoEx. An episode enters the dataset only if the task is completed; the 50-rollout dataset is a subset of the 200-rollout one. In this study the initial configurations were chosen by hand and the scene was reset manually between episodes, to make sure every episode started from a valid, known configuration; the reset instruction of Fig. 6 then re-issues the task from that configuration. Each episode records three RGB camera streams at 640×480 and 30 Hz together with the joint positions of all arms; actions are joint-angle targets. The 20 reference demonstrations were collected by human teleoperation with GELLO (Wu et al., 2024), a low-cost leader-arm interface, and recorded in the same format.

D.2 Policy Fine-Tuning and Evaluation

We start from the publicly released pre-trained $\pi_{0.5}$ weights (Physical Intelligence et al., 2025) and fine-tune one policy per task and per dataset (20 teleoperated, 50 CoEx, 200 CoEx), for 30k steps with batch size 64 on 8 H100 GPUs; all other settings follow the released fine-tuning recipe, and no data augmentation is used. The policy observes the same three camera views and joint positions as recorded during collection, and outputs joint-angle targets for all arms. Every policy is evaluated for 10 real-robot trials on its task, with the same evaluation protocol as in Section 4.1.

D.3 Experience Storage and Reuse

The framework keeps the outcome of every verified episode. Rather than storing raw programs, the agent itself summarizes what it learned (strategies and grasp parameters that survived verification, and the causes of failed rounds) into short notes, which are stored and provided to CoEx through the *Operational Hints* field of its prompt (Section F.3) in later runs. We do not ablate this mechanism separately.

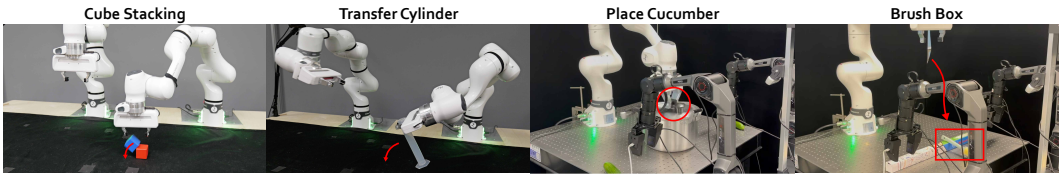

Figure 8: **Representative failure cases.** From left to right: (1) *Cube Stacking*: the stacked cube slips off due to sim-to-real positional offset at placement; (2) *Transfer Cylinder*: the receiving arm fails to align with the handing arm's gripper during handover, causing the cylinder to drop; (3) *Place Cucumber*: the pot lid is grasped at an unstable point (red circle), causing it to fall during the holding phase; (4) *Brush Box*: planning error causes the brush to be dropped (red rectangle) during moving phase.

E Failure Analysis

We observe three recurring failure modes in the real-robot experiments; Fig. 8 shows representative cases. We describe them qualitatively and note a mitigation for each.

E.1 Sim-to-real positional discrepancy

This mode affects all five tasks to varying degrees. The root cause is residual error in camera extrinsic calibration and object pose estimation, which introduces millimeter-level spatial offsets between the planned simulation trajectory and the actual real-world execution. In *Cube Stacking* (Fig. 8, left), the offset manifests at the final placement step: the arm positions the cube slightly off-center relative to the base cube, causing it to slide off after release. In *Ball Pickup*, both arms must converge on the ball simultaneously, and even small per-arm offsets ($\sim$2–3 mm) compound into a gap that prevents stable bimanual contact. In *Place Cucumber*, the narrow pot opening ($\sim$12 cm diameter) leaves minimal tolerance for insertion error, causing cucumbers to collide with the rim rather than entering cleanly. A lightweight residual correction at contact, or online re-calibration of camera extrinsics from fiducials between trials, would absorb these millimeter-level errors before they propagate to contact failures.

E.2 Planning loop stagnation

In the step-wise agent, the VLM planner sometimes fixates on a specific grasp strategy that has already failed, repeatedly attempting minor pose adjustments without exploring fundamentally different approaches (e.g., switching from a top-down grasp to a side grasp). Our stuck pattern detection mechanism (Sec. C.3.2) mitigates but does not fully eliminate this issue. In code-driven exploration, the code agent occasionally produces only incremental parameter changes between iterations (e.g., shifting a target position by 1 mm) without addressing the underlying geometric constraint violation, exhausting the iteration budget $M_{\max}$ without convergence. Diversity-promoting prompts after repeated failures, or an explicit failure memory that steers the agent away from strategies that have already failed, are natural remedies.

E.3 Agent-specific limitations

In the step-wise agent, cumulative drift is the primary concern for long action sequences. Each VLM-issued action introduces a small tracking error from the PD controller, and over 20+ sequential actions, these errors accumulate to produce noticeable end-effector drift. This is especially problematic in *Brush Box* (Fig. 8, right), where the sweeping motion requires sustained contact between the brush and box over multiple strokes. By the third or fourth stroke, the brush may have drifted several millimeters away from the box surface. For CoEx, the demanding cases are those in which one arm's motion depends on where another arm actually ended up, as in the *Transfer Cylinder* handover (Fig. 8, second from left): the program must anticipate the hand-over pose instead of adapting to it online. Letting a program hand such reactive sub-steps to step-wise control, while keeping the rest at the program level, is a promising hybrid.

Table 9: **Shared domain knowledge** embedded in the system prompts of both agents. All entries are generic robot/physics specifications, not task-specific solutions.

Category	Content
Gripper specifications	Maximum opening width and finger length for each robot type (Franka, Piper).
Orientation semantics	Delta-based rotation convention; mapping from pitch/yaw values to physical gripper directions.
Grasp strategy guidelines	Shape-conditioned heuristics (top-down for flat objects, horizontal for upright cylinders, cooperative bimanual for oversized objects); edge-grasp safety margins; fingertip positioning rules (overshoot for horizontal grasps, low-z target for flat objects).
Workspace limits	Per-robot reachable workspace ranges.
Collision avoidance	Awareness that arm links sweep a volume beyond the TCP; minimum clearance thresholds; retreat-toward-base principle for clearing shared workspace.

F Prompt Design

This section presents the prompt designs for both agents. We first describe the shared domain knowledge embedded in both prompts (§F.1), then detail the prompt structures of the step-wise baseline (§F.2) and of code-driven exploration (§F.3).

F.1 Shared Domain Knowledge

Both agents embed a common manipulation knowledge base that provides the foundation model with the minimal physical specifications needed to operate the heterogeneous robot fleet. Table 9 summarizes the categories of domain knowledge and their roles. Importantly, this knowledge base contains only *generic robot specifications and physics constraints* (e.g., gripper dimensions, workspace limits, coordinate semantics); it does not include any task-specific solutions or pre-computed trajectories. The model must still reason about object geometry, plan grasp strategies, and coordinate multi-arm actions autonomously based on real-time observations.

F.2 CoEx w/o code agent Baseline: VLM Prompts

The step-wise baseline uses two distinct prompts sent to GPT-5.6 Sol with multi-modal (text + image) input: a *planning phase prompt* for multi-round scene analysis, and an *execution phase prompt* for closed-loop action generation. We present each prompt's overall structure (with short sections inlined), then expand the output format blocks separately.

F.2.1 Planning Phase Prompt.

The following box shows the full planning prompt skeleton. Sections referencing Table 9 provide the shared domain knowledge described above; the output format is given in Prompt 2.

```
You are a robotics expert planning a manipulation task.
Analyze the scene and create a HIGH-LEVEL strategy.

# Task
{task_name}: {task_description}
Success criteria: {success_criteria}
# Operational Considerations
- {hint_1}   (task-specific tips, e.g., reach limits)
- {hint_2}

# Current State
  Robot {name} (id={id}, type={type}):
    Position: (x, y, z)
    Euler (RPY): (r, p, y)
    Gripper: {open/closed}
```

```
  Object {name} ({attr_name}):
    Position: (x, y, z)
    Size: {sx} x {sy} x {sz} m

# Visual Observation Strategy
You MUST actively observe the scene from multiple angles
before declaring planning complete:
- Request at least 2 different camera angles
- Use top-down view (pitch=1.2+) for XY layout and
  object orientations
- Use side views (yaw=0, +/-1.57, +/-3.14) for heights
  and gripper clearances
- Cross-reference visual observations with numerical
  state data (positions, sizes)
- If visual contradicts numerical data, trust the visual

# Per-Object Scene Analysis
Before planning any grasp, analyze EACH object independently:
determine its shape, long-axis direction, and the graspable
width at the planned grasp point. Set gripper yaw so fingers
close along the narrowest dimension.

# Gripper & Grasp Reference
  [Domain knowledge: see Table 9]

# Multi-Arm Collision Avoidance
  [Domain knowledge: see Table 9]

# Output Format
  => See Prompt 2
```

Prompt 1: Planning Phase: Full Prompt Structure

```
<thinking>
Your analysis:
- What you see in the image

Per-Object Orientation Analysis:
| Object | Shape | Long Axis | Grasp Point Location | Graspable Dim at Grasp Point |
    Chosen Yaw | Reason |

For each object, explain:
- Why this yaw was chosen for THIS specific object
- What dimension the fingers will close along
- Grasp strategy for each object
- Pre-grasp verification strategy
- Potential collision risks and mitigations
</thinking>

<next_action>
Either:
1. Request a different view (RECOMMENDED):
   {"type": "CAMERA_ORBIT",
    "params": {"yaw": X.XX, "pitch": X.XX},
    "reason": "why this angle helps"}

2. Or declare planning complete:
   PLANNING_COMPLETE

   <key_observations>
   Critical findings for execution:
   - Object positions and orientations
   - Chosen grasp strategy
   - Key constraints (clearances, collision avoidance)
   </key_observations>

   <checkpoints>
   Steps requiring verification before proceeding:
```

```
   - CP1: what to verify
     Position: xyz within 0.02m?
     Orientation: pitch/yaw within 0.1 rad?
     Visual: camera confirms object between fingers?
     Recommended view: yaw=X.XX, pitch=X.XX
   </checkpoints>

   <execution_plan>
   Numbered steps with approximate targets:
   1. [Robot X] MOVE + ROTATE - reason
   2. [Robot X] GRASP - CHECKPOINT CP1
   3. [Robot X] MOVE (lift) + MOVE (transport) + RELEASE

   Multi-robot coordination:
   MERGE synchronized actions into ONE step
   SEPARATE for different actions or verification
   </execution_plan>
</next_action>
```

Prompt 2: Planning Output Format

F.2.2 Execution Phase Prompt.

The following box shows the full execution prompt skeleton. The output format is given in Prompt 4.

```
You are executing a manipulation task. Your job is to OBSERVE
the scene and CALCULATE the exact movements needed.

# Task
{task_name}: {task_description}

# Current State
  [Same format as planning, plus inter-agent distances]

# Strategy from Planning
{key_observations from planning phase}
NOTE: Delta values from planning may be STALE -- always
recompute from CURRENT state data.

# Execution Plan (Step {current}/{total})
  [check] Step 1: [Franka] MOVE to lid (done)
  [->]    Step 2: [Franka] GRASP - CHECKPOINT (current)
  [o]     Step 3: [Franka] MOVE lift lid (pending)
  Current Goal: {current_step_description}

# Actions
Camera:
- CAMERA_ORBIT(yaw, pitch)
Robot (always include agent_id):
- MOVE(delta_x, delta_y, delta_z, agent_id)
- ROTATE(delta_yaw, agent_id, delta_pitch=0, delta_roll=0)
- GRASP(target_width, agent_id): -0.6 Franka, -0.8 Piper
- RELEASE(agent_id)
  agent_id: single int OR list for synchronized actions

# Gripper Geometry & Workspace Limits
  [Domain knowledge: see Table 9]

# Pre-GRASP Verification
Before GRASP, verify numerically AND visually:
1. Position: error > 0.02m -> MOVE to correct first
2. Orientation: error > 0.1 rad -> ROTATE first
3. Visual: CAMERA_ORBIT to confirm object between fingers

# Multiple Actions in One Output
Combine when safe (e.g., ROTATE -> MOVE -> GRASP).
Do NOT combine when visual verification is needed.
```

```
# Output Format
  => See Prompt 4
```

Prompt 3: Execution Phase: Full Prompt Structure

```
<observation>
What you see:
- Where are the target object and gripper?
- Cross-check: does the image match numerical state data?
- Any collision risks?
</observation>

<reasoning>
1. Goal of current step
2. Read orientation from state data: quote the EXACT current
   yaw and pitch. Compare with target.
   If already within 0.1 rad -> do NOT rotate.
3. Position: current vs target -> required delta_x/y/z
4. Orientation: if drift > 0.1 rad -> corrective ROTATE
5. If about to GRASP: position < 0.02m? orientation < 0.1
   rad? visually verified? If any fails -> fix first
6. Decision and action
</reasoning>

<action>
Single action:
{"type": "ACTION_TYPE", "params": {...}, "step_done": false}

OR multiple sequential actions:
[
  {"type": "ROTATE", "params": {...}, "step_done": false},
  {"type": "MOVE", "params": {...}, "step_done": false},
  {"type": "GRASP", "params": {...}, "step_done": true}
]
</action>
```

Prompt 4: Execution Output Format

F.3 CoEx: Code Agent Prompt

Code-driven exploration uses a single comprehensive system prompt. The following box shows the full prompt skeleton; sections marked with "⇒ *Prompt X*" are expanded in the corresponding prompts below.

```
You are a robot manipulation expert. Your job is to write and
iteratively refine Python code that controls multi-arm robots
in a physics simulation.

# TASK INFORMATION
Task: {task_name}
Description: {task_description}
Success Criteria: {success_criteria}
Robots:
  ID {id}: {type} ({n}-DOF), Name: "{name}", Role: {role}
Objects:
  {attr_name}: "{name}", Size: {x}x{y}x{z}m, {description}
Operational Hints:
  - {hint_1}   (task-specific tips from prior experience)

# API REFERENCE
  => See Prompt 6

# Coordinate System
X: +right/-left | Y: +away/-toward | Z: +up/-down
Table surface: z ~ 0
```

```
# Manipulation Knowledge
  [Domain knowledge: see Table 9]

# Robot Workspace & Reach Limits
- Franka: reach ~0.855m. At XY > 0.5m, Z severely limited.
- Piper: reach ~0.3m. Cannot raise above initial height.
- CRITICAL: MOVE to unreachable target silently stops at
  closest reachable point. MUST print diagnostics to detect.

# Diagnostic Printing
After critical MOVE -- print target vs actual position:
  state = robot.get_robot_state(0)
  print(f"TARGET: {t}, ACTUAL: {state['position']},
         GAP: {np.linalg.norm(...):.4f}")
  Large gap = arm hit joint limits.
After GRASP + LIFT -- print object z before/after:
  No z change = grasp missed.

# CODE RULES
1. ALWAYS read current state before computing deltas.
2. Call checkpoint() at critical moments.
3. Code must work for ANY initial state -- no hardcoding.
4. Use numpy (np) for vector math.
5. Each round resets from scratch -- code is self-contained.

# CODE TEMPLATE
  => See Prompt 7

# WORKFLOW (6 steps)
  => See Prompt 8
```

Prompt 5: Code Agent: Full Prompt Structure

```
## State Queries
robot.get_robot_state(robot_id: int) -> dict
  Returns: {"robot_id": int, "position": [x,y,z],
            "orientation": [qw,qx,qy,qz],
            "euler_rpy": [roll,pitch,yaw], "gripper": float}
  position: TCP fingertip in world coordinates (meters)
  gripper: 1.0 = open, -1.0 = fully closed
robot.get_all_robot_states() -> Dict[int, dict]
robot.get_object_state(attr_name: str) -> dict
  Returns: {"name", "position", "orientation"}
robot.get_all_object_states() -> Dict[str, dict]

## task_config Fields
task_config.robots  -- list of RobotConfig:
  .robot_id, .robot_type ("franka"/"piper"), .name, .role
task_config.objects -- list of ObjectConfig:
  .attr_name, .name, .size_x, .size_y, .size_z, .description

## Action Execution
robot.execute_action(action: dict) -> bool
  MOVE:    {"type":"MOVE", "params":{"delta_x":f, "delta_y":f,
            "delta_z":f, "agent_id":int|list}}
  ROTATE:  {"type":"ROTATE", "params":{"delta_yaw":f,
            "delta_pitch":f, "delta_roll":f,
            "agent_id":int|list}}
  GRASP:   {"type":"GRASP", "params":{"target_width":f,
            "agent_id":int|list}} // -0.6 Franka, -0.8 Piper
  RELEASE: {"type":"RELEASE", "params":{"agent_id":int|list}}
  Multi-robot: agent_id as list for synchronized actions.
  Params can be scalars (broadcast) or lists (per-robot).

## Checkpoint
checkpoint(name, checkpoint_type="generic", notes="",
           suggested_views=None)
  Types: "grasp", "lift", "handover", "place", "generic"
```

```
  Auto-generated: final_state, EXECUTION_ERROR, TIMEOUT_ERROR
```

Prompt 6: API Reference

```
# Available: robot, task_config, checkpoint, np, time, print

# 1. Read current state
r0 = robot.get_robot_state(0)  # Piper-0
r1 = robot.get_robot_state(1)  # Piper-1
r2 = robot.get_robot_state(2)  # Franka
objects = robot.get_all_object_states()

# 2. Open all grippers
robot.execute_action(
  {"type": "RELEASE", "params": {"agent_id": 0}})
robot.execute_action(
  {"type": "RELEASE", "params": {"agent_id": 1}})
robot.execute_action(
  {"type": "RELEASE", "params": {"agent_id": 2}})

# 3. Move to target (compute deltas from current position)
# For Piper: GRASP target_width = -0.8
# For Franka: GRASP target_width = -0.6

# 4. Grasp, lift, checkpoint ...
# checkpoint("step_name", checkpoint_type="generic",
#            notes="Describe what to verify here")
```

Prompt 7: Code Template (2 Piper + 1 Franka)

```
Step 1: Understand the API
  The API REFERENCE above is the complete guide.

Step 2: Make a Plan
  Save to {workspace}/plan.md:
  - Grasp strategy per object (approach, orientation, yaw)
  - Action sequence and multi-arm coordination
  - Collision risks
  Optional: run exploration scripts (don't count as rounds).

Step 3: Write Code and Run
  Write to {code_file}. Code runs to completion in one shot
  -- NO real-time visual feedback. checkpoint() captures
  images for post-run review only.

Step 4: Analyse Results
  Output directory contains:
  - summary.json: error status, final states, checkpoints
  - execution_stdout.txt: all print() output
  - initial_state/: front.png, top.png, side.png
  - checkpoints/<name>/: images + states.json

  Analysis checklist (EVERY round):
  1. Read stdout -> target vs actual position gaps
  2. Read summary.json -> did objects actually move?
  3. View key checkpoint images (pre_grasp, post_lift)
  4. Identify ROOT CAUSE, not symptoms

Step 5: Iterate
  Fix code, run next round. Environment resets each time.

Step 6: Final Report -> {workspace}/final_report.json
```

Prompt 8: Iterative Workflow