# AGENTIC RSR: REAL-TO-SIM-TO-REAL THROUGH SCENE RECONSTRUCTION AND EXECUTION-GROUNDED ROBOT POLICIES

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

A simulation of a real robot workspace must preserve task-relevant interactions, while policies developed in it must operate on observations available to the real robot. Yet scene reconstruction and policy development are often treated separately. We present **Agentic Real-to-Sim-to-Real (Agentic RSR)**, a framework that links scene reconstruction, policy development, and real-robot execution through the same manipulation task. Given a workspace video, a task description, and a known robot model, an agent recovers metric scale, iteratively refines the scene using visual feedback, and checks task-relevant interactions in MuJoCo. A coding agent then develops an executable policy, progressing from privileged object poses to visual observations and randomized simulation. The policy can interleave multiple observations and actions within one invocation, while the agent uses execution feedback to continue, retry, or revise its approach. A shared task-level interface carries the policy and accumulated experience to the real robot, where fresh observations and safety checks guide execution. Across 18 reconstructed scenes involving two robots, the mean four-view Depth MAE against reference depth estimates is 0.1057 m, the mean Lab $\Delta E_{76}$ is 11.04, and the mean grayscale SSIM is 0.6990. In real-robot experiments, the aggregate task success rate reaches 80% of the simulation task success rate, indicating substantial retention of simulated performance on hardware. Code and reconstructed scene data will be made publicly available.

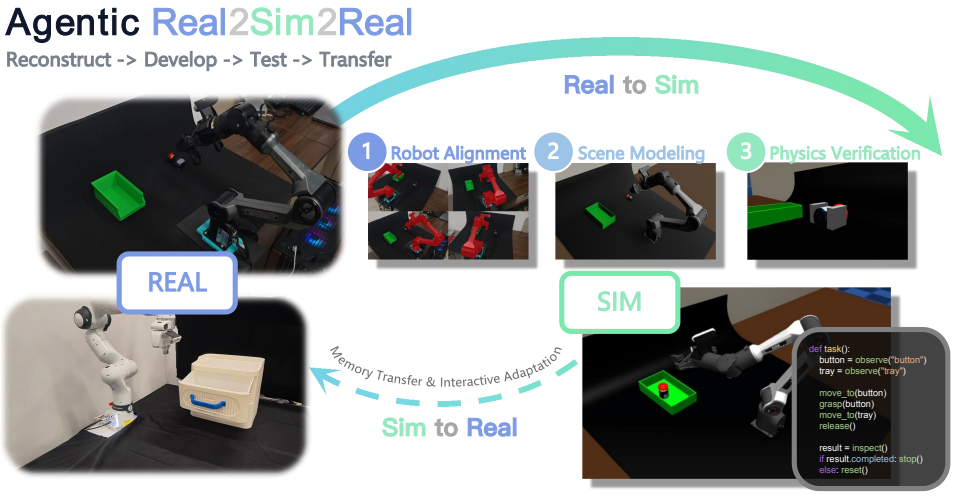



Figure 1: **Agentic RSR overview.** An agent uses real observations and known robot geometry to reconstruct a metrically aligned scene and check task-relevant interactions. A policy agent develops and tests an executable policy in simulation, then transfers it with experience memory to the real robot for feedback-driven adaptation.

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.

# 1 INTRODUCTION

Developing and evaluating robot policies through real-world trials is costly and difficult to scale. Simulation offers an alternative for policy evaluation and training (Li et al., 2025; Ranawaka et al., 2026). For a policy intended to return to a captured workspace, the simulated scene must support task-relevant interactions, and the policy must operate on observations available to the real robot.

Existing systems already bridge parts of this chain. Re$^3$Sim reconstructs interactive scenes and demonstrates sim-to-real policy transfer (Han et al., 2025); Agentic Real2Sim uses vision-language agents to build physics-based episodic scenes for downstream policy evaluation and fine-tuning (Chen et al., 2026a); and SimFoundry turns video into interactive scenes for simulation-based evaluation and sim-to-real policy training (Ranawaka et al., 2026). In programmatic control, Code as Policies generates executable programs over perception and control APIs (Liang et al., 2023), while CaP-X studies multi-turn execution feedback and transfers code-based policies using tools shared across simulation and reality (Fu et al., 2026). This motivates a more specific question: how can agent-refined reconstruction of an observed workspace be coupled with an execution-grounded programmatic policy and a consistent task-level interface across simulation and hardware?

We present **Agentic Real-to-Sim-to-Real (Agentic RSR)** to study this connection. Given a workspace video, a task, and a known robot, a Codex agent (OpenAI, 2026) reconstructs a metrically aligned, interactive scene and refines it against visual and task-relevant physical checks. The same Codex agent (OpenAI, 2026) then develops an executable programmatic policy, progressing from privileged object poses to visual observations and randomized simulation. The policy may be revised during development, but its version is locked during final evaluation. A shared task-level interface carries the policy and experience memory to hardware, where fresh observations and execution feedback support verification and recovery. Real-robot experiments show that the policies retain much of their aggregate task performance after transfer from simulation.

Our contributions are:

- **Agentic Real-to-Sim scene reconstruction for executable tasks.** We anchor the metric scale of a video-reconstructed workspace to known robot geometry, iteratively reconstruct the scene using reference views and depth and appearance feedback, and check task-relevant geometric and physical interactions so that the scene supports robot execution;
- **An execution-grounded programmatic policy developed by a Codex agent.** The Codex agent (OpenAI, 2026) generates a policy bundle containing a complete program and reusable steps. The bundle remains locked during an episode, while the Codex agent (OpenAI, 2026) uses execution results and fresh observations to decide whether to continue, switch steps, retry, or recover;
- **A unified Agentic Real-to-Sim-to-Real closed-loop framework.** A consistent task-level observation and action interface connects scene reconstruction, simulation-based policy testing, and hardware execution, where execution feedback supports verification and recovery.

# 2 RELATED WORK

**Real-to-Sim Scene Reconstruction for Robot Policies.** RL-GSBridge and GSWorld combine high-fidelity scene representations with robot simulation, while Re$^3$Sim and RoboSimGS couple rendering with interaction geometry for manipulation and sim-to-real learning (Wu et al., 2025; Jiang et al., 2026; Han et al., 2025; Zhao et al., 2026). RoboSnap builds simulation-ready scenes from a single RGB image (Zhang et al., 2026). Scalable Real2Sim recovers physics-aware assets through robot interaction, whereas TwinAligner addresses visual and dynamic alignment for policy transfer (Pfaff et al., 2025; Fan et al., 2025).

ACDC generates *digital cousins* that preserve task-relevant affordances, while SIMPLER studies how simulated evaluation relates to real-robot performance (Dai et al., 2025; Li et al., 2025). Holodeck generates prompt-conditioned environments; Gen2Sim produces 3D assets, tasks, and rewards for robot learning (Yang et al., 2024; Katara et al., 2024). SceneSmith, PhyScensis, SimWorld Studio, and Articraft respectively explore vision-language scene generation, physics-guided

arrangement, coding-agent environments, and programmatic articulated assets (Pfaff et al., 2026; Wang et al., 2026b; Kang et al., 2026; Zhou et al., 2026). Closer to video-based real-to-sim, Agentic Real2Sim uses agents and simulator-in-the-loop refinement to turn recorded robot–object interactions into simulatable episodic twins, while SimFoundry generates interactive scenes and *digital cousins* for policy evaluation and sim-to-real training (Chen et al., 2026a; Ranawaka et al., 2026).

**Programmatic Robot Policies and Execution-Grounded Agents.** Code as Policies and Instruct2Act compose perception and actions into executable programs (Liang et al., 2023; Huang et al., 2023a). SayCan pairs language planning with feasible pretrained skills, while VoxPoser uses 3D value maps for closed-loop trajectory planning (ichter et al., 2023; Huang et al., 2023b). Inner Monologue incorporates environmental feedback into planning; CaP-X studies multi-turn coding agents and code-policy transfer across simulation and hardware (Huang et al., 2023c; Fu et al., 2026). ASPIRE and ENPIRE explore feedback-driven skill development; Show-Harness offers a semantic action interface, and RoboClaw orchestrates learned primitives for long-horizon tasks (Lu et al., 2026; Xiao et al., 2026; Chen et al., 2026b; Li et al., 2026).

Visual and dynamics randomization support sim-to-real transfer (Tobin et al., 2017; Peng et al., 2018), while RT-2, OpenVLA, Octo, and $\pi_0$ develop data-driven visuomotor rather than executable robot policies (Zitkovich et al., 2023; Kim et al., 2025; Ghosh et al., 2024; Black et al., 2025). Our focus is to connect reconstruction of a particular captured workspace with a programmatic policy for the same task, evaluated in simulation and on real robots.

Table 1: **System-level capability comparison.** Reported capabilities in real-scene reconstruction, agentic scene revision, code-policy generation, execution-time recovery, and sim-to-real trials. Symbols are defined below the table.

| Capability | CaP | Agentic R2S | CaP-X | Re$^3$Sim | SimFoundry | Agentic RSR (Ours) |
|---|---|---|---|---|---|---|
| Real-scene reconstruction | × | ✓ | × | ✓ | ✓ | ✓ |
| Agentic scene revision | × | ✓ | × | × | × | ✓ |
| Code-policy generation | ✓ | × | ✓ | × | × | ✓ |
| Runtime agent recovery | – | × | ✓ | × | × | ✓ |
| Sim-to-real trials | – | – | ✓ | ✓ | ✓ | ✓ |

✓: experimentally demonstrated; –: not established in the paper; ×: not part of the implemented method.

# 3 METHOD

## 3.1 REAL-TO-SIM SCENE RECONSTRUCTION FOR EXECUTABLE TASKS

**Task and scene output.** Given a continuous RGB video $V$ of a workspace, a task description $q$, and a known robot asset $U$, we aim to reconstruct the particular captured workspace rather than generate a semantically similar scene variant. The resulting scene should reproduce task-relevant geometry and appearance from reference views while providing collision and physical representations that support robot–object interaction. The workflow connects robot-anchored metric registration, feedback-guided agentic scene reconstruction, and task-interaction checks in MuJoCo (Todorov et al., 2012). The robot asset $U$ supplies both a metric geometric reference and the kinematic model used in downstream simulation.

**Robot-anchored metric registration.** We sample 32 frames from $V$ and select four as fixed reference views for subsequent scene reconstruction. VGGT-$\Omega$ (Wang et al., 2026a) estimates camera intrinsics $K_v$, world-to-camera extrinsics $E_v$, and depth $d_v$ for sampled frame $v$, while SAM3 (Carion et al., 2026) supplies robot masks. Back-projecting masked depths yields a robot point cloud in the reconstruction frame. We align the known meter-scale URDF surface with this cloud using multi-scale Sim(3) registration, followed by local pose and scale refinement. Let $T_{B\to G}$ denote the similarity transform from robot-base frame $B$ to reconstruction frame $G$, with scale $s$ measured in reconstruction units per meter. Applying $T_{B\to G}^{-1}$ expresses the reconstructed scene and camera

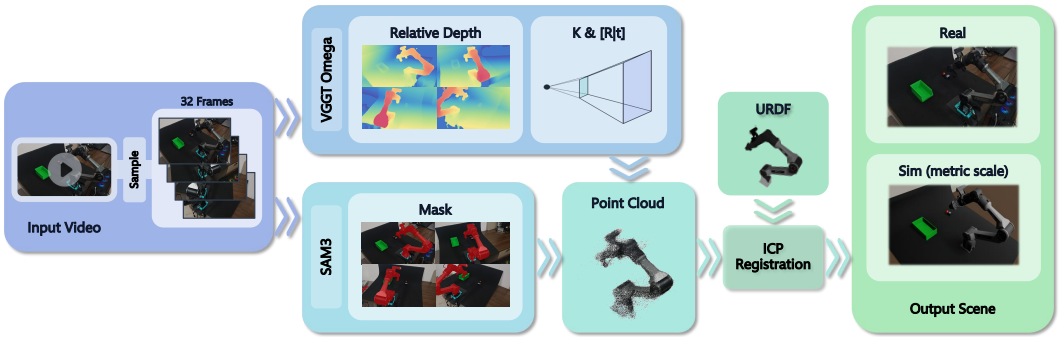



Figure 2: **Recovering metric scale from known robot geometry.** Video-estimated depth, camera parameters, and robot masks are used to construct an observed robot point cloud. Aligning this point cloud with the known URDF geometry yields metric depth and camera poses expressed in the robot-base frame.

poses in the metric robot-base frame; dividing the estimated depths by $s$ converts them to meters while preserving their relative geometry. Candidate registrations are ranked by projected-mask agreement, bidirectional point-cloud fit, coverage, and depth consistency. Appendix B specifies the objective and search settings.

**Feedback-guided agentic scene reconstruction.** We invoke Codex agents (OpenAI, 2026) through a Python SDK. A planning session first receives $q$ and four real reference RGB images and produces a plan for task objects, background elements, and spatial relationships. A separate refinement session receives this plan, the initial Blender (Blender Foundation, 2026) scene, the fixed reference-camera status, the real images, and RGB previews of the current scene. Across successive rounds, it edits object geometry, relative layout, materials, and lighting. The robot and four reference cameras are frozen by the execution program and cannot be changed by the agent. The agent can use Blender (Blender Foundation, 2026) Lab MCP tools for scene editing and preview when available. In each round, the refinement agent renders and inspects the candidate before submitting a Blender (Blender Foundation, 2026) Python edit script. An external runner applies the script, checks the frozen assets, saves a separate candidate version, and renders and evaluates it from the four reference views. The agent receives scalar feedback for overall depth and appearance error; acceptance is determined by the programmatic evaluator rather than the agent's self-assessment.

For reference view $v$, the evaluator records metric depth error $D_v$, Lab color difference $C_v$, and valid rendered-depth coverage $Q_v$. Equal-weight averages over the four views give $\overline{D}$ and $\overline{C}$. The joint visual check requires $\overline{D} < 0.15$ m and $\overline{C} < 15$, together with $D_v < 0.20$ m, $C_v < 30$, and $Q_v \geq 0.50$ for every view. Otherwise, refinement continues or records failure at the iteration limit. Reaching the mean-depth threshold alone does not establish full acceptance. Since the reference depths come from the preceding video reconstruction, $D_v$ measures consistency with that metric reference rather than geometric error against independently measured ground truth. Per-scene results appear in Appendix C.

**MuJoCo scene and task-interaction checks.** For a candidate that passes the visual check, a separate Codex session (OpenAI, 2026) receives the task description, scene plan, mesh inventory, and reference-view images. It classifies meshes as visual-only elements, static support or collision geometry, or movable or articulated task objects. These classifications are reviewable physical hypotheses, not measurements of real object properties. An exporter then combines the scene with the known robot model to generate a MuJoCo (Todorov et al., 2012) scene $\mathcal{S}_M$ with appropriate collision representations and simulated physical properties. A separate Codex physics-checking session (OpenAI, 2026) can use repository-provided inspection tools to examine task-object states and run settling, simplified grasp, or articulated-motion probes. Where justified, it tests a separate candidate with changes to collision geometry, object pose, or physical parameters. The runner records tool calls and reports and reruns the relevant probes on revised candidates; visible geometry or pose changes also trigger regression checks from the fixed reference views.

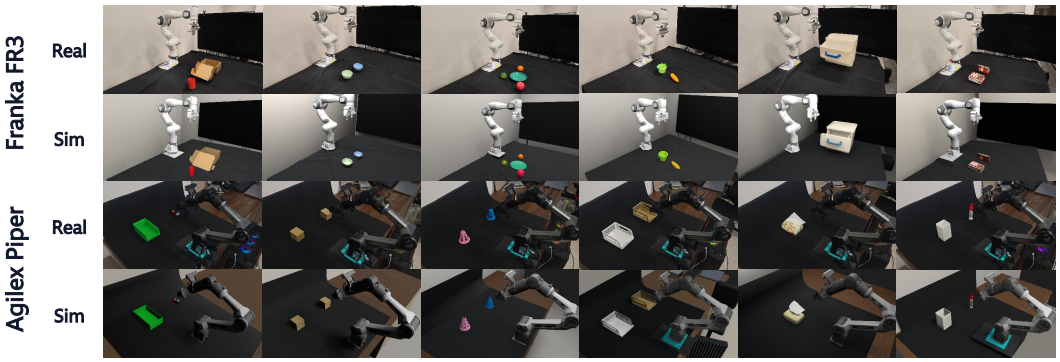



Figure 3: **Real and simulated views of a subset of task scenes.**

### 3.2 Simulation-based policy generation and execution

**Three-pass policy iteration.** Policy development proceeds through three successive passes. First, ground-truth object poses are temporarily available in the nominal scene, allowing the agent to establish task decomposition and action logic. Second, this privileged access is removed: the policy must estimate object state through the visual observation interface intended for subsequent real-robot use and complete the task in the nominal scene. Third, the same visual interface is retained while bounded variation is introduced in initial object positions, camera pose and calibration, and action execution. This pass tests whether the policy adapts to new observations and execution outcomes. The agent may revise its policy within each pass; when it judges the task complete from observations, it renders a final frame as evidence before passing the policy and experience notes to a fresh agent conversation for the next pass. These passes constitute iterative development; the submitted policy version is held fixed during final evaluation, while its actions can still respond to observations.

**Policy-facing interface.** The simulator loads a reconstructed MuJoCo (Todorov et al., 2012) scene and provides a consistent task-level interface for Piper and Franka. State and observation interfaces provide the end-effector state, estimated wrist-camera pose, gripper state, wrist and third-person images, and object-position estimates derived from wrist RGB-D and SAM3 (Carion et al., 2026). Action interfaces support end-effector pose commands, gripper control, and sparse trajectories; control-flow interfaces support scene reset, yielding control, and stopping execution. Apart from the temporary object-pose access in the first pass, the visual policy cannot read simulator truth or rely on a built-in pick-and-place action or task-success oracle.

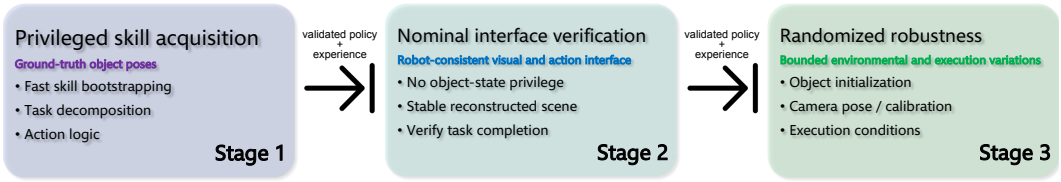



Figure 4: **Three-stage policy development.** The policy progresses from privileged skill acquisition to nominal interface verification and randomized robustness, passing validated policy and experience between stages.

**Execution-grounded policy loop.** In each development pass, the Codex agent (OpenAI, 2026) writes a programmatic policy using the task description and public interface. A single program execution may interleave repeated observations and actions. If execution fails or yields control, the runner returns action outcomes, errors, and recent observations to the same agent conversation. The agent can then continue from the current scene, reset and retry, or revise the policy before another execution. Yielding control preserves the scene state; resetting starts a new attempt and resamples perturbations in the randomized pass. The agent decides whether the task is complete

from its observations and provides a final rendered frame as evidence; it does not receive a built-in task-success oracle.

### 3.3 SIM-TO-REAL TRANSFER OF POLICY AND EXPERIENCE MEMORY

**Policy and experience memory.** The simulation phase transfers two complementary forms of information: the policy retains validated task steps and observation–action logic, while experience memory summarizes effective behaviors, failure causes, perception difficulties, and actions to try or avoid. This memory helps the agent adapt to the real environment; it does not provide ground-truth object poses or simulator internals. A new agent conversation begins with the policy and experience memory but must re-localize objects and assess task state from current real observations rather than reuse simulated coordinates or success claims.

**Robot-side safety constraints and online verification.** During transfer, a robot-side safety mechanism automatically checks coordinate frames and camera calibration, enforces action limits and a safe workspace specific to the robot and task, and gates execution accordingly. Actions that violate these constraints are not executed. In parallel, the real-robot policy instructions require the agent to observe more frequently within a single program invocation, particularly before and after consequential actions, so that it reconfirms object locations and robot state before continuing, adjusting, or stopping. Real execution outcomes may update the experience memory; the agent assesses task completion from current observations rather than relying on a built-in success signal.

## 4 EXPERIMENTS

We evaluate Agentic RSR on 18 real-world manipulation scenes and tasks across two robots, covering scene reconstruction, simulated and real-robot policy execution, and component ablations.

### 4.1 EXPERIMENTAL SETUP

Each of the 18 tasks is paired with a workspace RGB video. All scene-reconstruction methods receive the same source video and select frames according to their input requirements; method-specific additional capture is reported separately. The camera slowly circles the workspace, with each frame showing the robot, scene, and task objects. Unless noted, Agentic RSR uses Codex agents (OpenAI, 2026) powered by GPT-6 Astra with high reasoning effort; the model-swap ablation uses the stated alternative.

We measure scene quality with depth mean absolute error (Depth MAE), CIE 1976 Lab color difference (Lab $\Delta E_{76}$), and structural similarity index measure (SSIM). When object-level geometry is available, we also report symmetric Chamfer distance (CD), F1 at a 1 cm threshold (F1@1cm), and bounding-box center error.

For policy evaluation, task success rate (SR) is the fraction of completed tasks for Instruct2Act (Huang et al., 2023a) and Agentic RSR; Code as Policies (Liang et al., 2023) instead reports a task-equal average of recorded real-robot attempts. We report simulation success rate (Sim SR) and real-robot success rate (Real SR). For Code as Policies and Instruct2Act, conversion is the descriptive ratio $100 \times \mathrm{Real\ SR}/\mathrm{Sim\ SR}$ from unrounded rates; it is undefined when Sim SR is zero and is not paired task transfer. For Agentic RSR, conversion is the fraction of simulation-successful tasks that also succeed on the real robot. All aggregate results weight scenes or tasks equally.

Table 2: Scene reconstruction results across 18 tasks. – indicates that the method does not provide the outputs needed to compute the corresponding metric.

| Method | Input | Depth MAE (m)↓ | $\Delta E_{76}$ ↓ | SSIM ↑ | CD (m)↓ | F1@1cm↑ | BBox (m)↓ |
|---|---|---|---|---|---|---|---|
| Re$^3$Sim (Han et al., 2025) | Multi-view | – | 20.60 | 0.5835 | – | – | – |
| RL-GSBridge (Wu et al., 2025) | Video | – | 22.87 | 0.5568 | – | – | – |
| RoboSimGS (Zhao et al., 2026) | Multi-view | – | 20.43 | 0.5939 | – | – | – |
| SimFoundry (Ranawaka et al., 2026) | Video | 0.3916 | 60.48 | 0.4495 | 0.1736 | 0.0084 | 0.2355 |
| Agentic RSR (Ours) | Video | **0.1057** | **11.04** | **0.6990** | **0.0303** | **0.3655** | **0.0869** |

### 4.2 Real-to-Sim Scene Reconstruction

We first test whether workspace videos can be converted into metric, visually consistent, task-relevant simulation scenes. Robot segmentation and registration use VGGT-Ω (Wang et al., 2026a), SAM3 (Carion et al., 2026), and the known robot URDF to recover scale and robot–camera alignment. Scene planning and refinement construct the task scene in Blender (Blender Foundation, 2026) while keeping the registered robot and reference cameras fixed; the selected scene is converted to MuJoCo (Todorov et al., 2012) for interaction.

Across the 18 scenes, refinement reduces mean four-view Depth MAE from 0.9036 m to 0.1057 m and mean Lab $\Delta E_{76}$ from 24.84 to 11.04, while achieving a mean full-frame grayscale SSIM of 0.6990. All scenes reach mean Depth MAE $< 0.15$ m by round 6; the mean first-hit and selected delivery rounds are 2.0 and 4.3. The depth threshold alone does not satisfy the complete acceptance criteria.

Across 42 measured objects, the scene-equal mean symmetric Chamfer distance is 0.0303 m, F1@1cm is 0.3655, and visible bounding-box center error is 0.0869 m. These results indicate that visual feedback improves appearance and depth consistency while recovering task-object arrangement. Per-scene results are provided in Appendix C.

We additionally evaluate Re$^3$Sim (Han et al., 2025), RL-GSBridge (Wu et al., 2025), RoboSimGS (Zhao et al., 2026), and SimFoundry (Ranawaka et al., 2026) on the 18 task videos. All methods share the source videos, but differ in whether they process video directly or selected multi-view frames. Missing entries in Table 2 reflect output incompatibility: some methods produce images or Gaussian representations without depth or instance-level geometry directly usable here, not an inherent inability of Gaussian representations to produce depth. Among comparable metrics, Agentic RSR has the lowest depth and color errors, highest SSIM, and best available object-geometry scores, indicating preservation of task-object geometry and spatial relationships for downstream interaction. Per-scene results appear in Appendix C.

### 4.3 Simulation and Real-Robot Policy Evaluation

We report simulation and real-robot results for Code as Policies (Liang et al., 2023), Instruct2Act (Huang et al., 2023a), and Agentic RSR across the 18 tasks, retaining each method's native policy format.

We measure success under randomized simulation conditions and recorded real-robot attempts. Table 3 reports aggregate results; task IDs are listed in Appendix A.

Code as Policies (Liang et al., 2023) reports Sim SR and task-equal real-robot success of 50.0% and 25.6%, respectively, for a descriptive conversion ratio of 51.1%. Instruct2Act (Huang et al., 2023a) succeeds on 7/18 simulation tasks and 5/18 real-robot tasks, yielding Sim SR, Real SR, and conversion of 38.9%, 27.8%, and 71.4%.

Agentic RSR completes 10/18 tasks in simulation, 8 of which also succeed on the real robot, yielding Sim SR, Real SR, and paired transfer of 55.6%, 44.4%, and 80.0%.

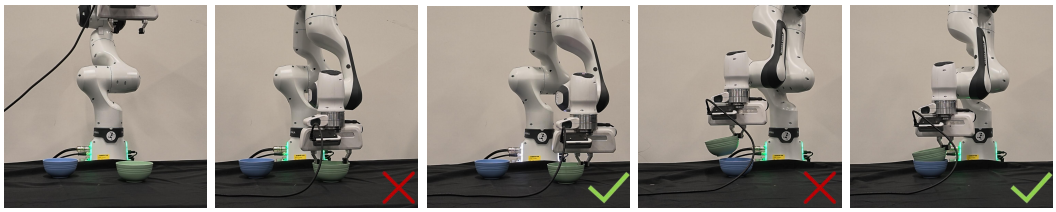



Figure 5: **Representative real-robot execution frames.** The initial scene is shown in (a); red crosses and green checks mark failed and successful placement outcomes in the subsequent trials.

Perception and manipulation technique both affect execution. A single wrist-camera view can be occluded or too narrow, and baseline records include cases where task objects were not recognized.

Agentic RSR instead iterates on task steps in simulation and uses execution feedback to reobserve, adjust, or retry, responding to viewpoint changes and post-manipulation object states.

For example, bowl stacking requires stable grasp and placement contacts, not just object localization. Repeated simulation attempts let the agent encode such techniques in policy steps and experience for real-robot execution. Table 3 therefore captures revision from execution outcomes, not only one-shot action generation. Because Code as Policies (Liang et al., 2023) hardware scenarios do not fully match the indexed tasks, real-robot comparisons remain descriptive rather than paired.

Table 3: Simulation and real-robot policy results across 18 tasks. For the Code as Policies and Instruct2Act baselines, conversion is the ratio of reported real-robot to simulation success rates, not paired task transfer. For Agentic RSR, conversion measures paired transfer among tasks that succeed in simulation.

| Method | Sim SR (%) ↑ | Real SR (%) ↑ | Conversion (%) ↑ |
|---|---|---|---|
| Code as Policies (Liang et al., 2023) | 50.0 | 25.6 | 51.1 |
| Instruct2Act (Huang et al., 2023a) | 38.9 | 27.8 | 71.4 |
| Agentic RSR (Ours) | **55.6** | **44.4** | **80.0** |

### 4.4 Ablations

We examine model choice, scalar metric feedback, and initial calibration on the same 18 scenes. The full system passes the visual check in all scenes; replacing the refinement agent with GPT-5.6 Sol at high reasoning effort reduces this to 10/18 and raises mean Depth MAE from 0.1057 m to 0.2119 m and Lab $\Delta E_{76}$ from 11.04 to 13.31. In this setting, GPT-6 Astra appears to use visual feedback more effectively, although this comparison does not establish a general advantage across spatial-reasoning tasks.

Without scalar metric feedback, eight scenes still reach the depth threshold at a mean first-hit round of 2.0, but only 6/18 pass the full visual check and mean Depth MAE rises to 0.3746 m. Images enable rapid adjustment, but explicit depth and appearance metrics better guide error prioritization and acceptance.

Without initial calibration, only 3/18 scenes pass. Of the four that reach the depth threshold, the mean first-hit round rises to 6.8. Lab $\Delta E_{76}$ remains close to the full system (11.99 versus 11.04), but Depth MAE increases to 0.2583 m. Thus, visually similar scenes can retain substantial position or scale errors; calibration provides a more reliable starting point. Table 4 summarizes these results, with per-scene data in Appendix C.2.

Table 4: Real-to-Sim scene-refinement ablations across 18 scenes. $R_{0.15}$ is the mean first-hit round among scenes reaching the depth threshold.

| Variant | Pass | $R_{0.15}$ ↓ | Depth MAE (m)↓ | Lab ↓ | SSIM ↑ |
|---|---|---|---|---|---|
| Full system | **18/18** | **2.0** | **0.1057** | **11.04** | **0.6990** |
| GPT-5.6 Sol (high) | 10/18 | 3.9 | 0.2119 | 13.31 | 0.6702 |
| w/o metric feedback | 6/18 | 2.0 | 0.3746 | 15.12 | 0.6490 |
| w/o initial calibration | 3/18 | 6.8 | 0.2583 | 11.99 | 0.6597 |

## 5 Limitations

In our exploratory use of GPT-6 Astra, we observed noticeable differences in agent behavior across devices and invocation interfaces, as well as between runs on the same device at different times. These observations have not been measured under a controlled, matched-input protocol, so their causes remain unclear. Such variability makes agent behavior difficult to reproduce when relying on a closed-source model service. We hope for stronger transparency and oversight of these services, including clear model versioning and stable invocation interfaces, so that researchers can access them consistently and reasonably reproduce experimental results.

Programmatic control and agent-driven revision present different trade-offs between execution speed and adaptability. Code as Policies (CaP) turns task instructions into executable programs that can themselves contain perception-feedback loops (Liang et al., 2023). Instruct2Act likewise generates programs that compose perception, planning, and robot actions from multimodal instructions (Huang et al., 2023a). When execution encounters a case outside a program's anticipated logic—such as perceptual confusion between similar-looking objects—a fixed program may have limited means to revise its subsequent strategy in light of the failure. Our approach allows a program to perform multiple observations and actions per invocation, while the agent uses execution feedback at checkpoints or after failures to decide how to proceed. This aims to balance execution efficiency and adaptability, but model-call latency and discrete checkpoints still limit our response to rapidly changing contacts compared with the high-frequency visuomotor control targeted by VLA policies such as $\pi_0$ (Black et al., 2025). Future systems could combine high-level programmatic reasoning with faster low-level closed-loop control. Tools such as Jev (Almeida, 2026), designed for fast structured decisions, offer another direction to investigate, although their benefit for robot tasks remains to be established.

## 6 Conclusion

We present Agentic RSR, a Real-to-Sim-to-Real loop connecting reconstruction of a captured workspace, execution-grounded programmatic policy development, and real-robot execution. Robot-anchored registration establishes metric scale; reference-view depth and appearance feedback guide scene reconstruction, while MuJoCo (Todorov et al., 2012) checks task-relevant physical interactions. Policy development progresses from privileged information to visual observations and then randomized simulation. A program can observe and act repeatedly within one invocation, and the agent uses execution outcomes to decide how to continue or recover. During transfer, the policy and experience memory carry task knowledge into real-robot execution; the agent adapts its subsequent behavior to real observations, while robot-side safety mechanisms constrain its actions. The central contribution is a task-level loop in which scene reconstruction, executable policies, and real-world execution feedback inform one another.

### AI use statement

Generative AI tools, including coding agents, assisted with research and evaluation planning, implementation of parts of the Real-to-Sim and simulation-policy workflows, organization of experimental records and references, and manuscript polishing and typesetting. The role of AI agents within Agentic RSR is described separately in Section 3. The authors retain responsibility for the study design, verification of reported results and citations, and the final manuscript.

## References

Diogo Almeida. Introducing system one models & jev. TypeSafe AI Blog, 2026. URL https://typesafe.ai/blog/introducing-system-one-models-and-jev.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. $\pi_0$: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems XXI*, 2025. doi: 10.15607/RSS.2025.XXI.010. URL https://www.roboticsproceedings.org/rss21/p010.html.

Blender Foundation. Blender: Free and open source 3d creation suite, 2026. URL https://www.blender.org/about/. Project website, accessed 2026-09-26.

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, Shoubhik Debnath, Ronghang Hu, Didac Suris Coll-Vinent, Chaitanya Ryali, Kalyan Vasudev Alwala, Haitham Khedr, Andrew Huang, Jie Lei, Tengyu Ma, Baishan Guo, Arpit Kalla, Markus Marks, Joseph Greer, Meng Wang, Peize Sun, Roman Rädle, Triantafyllos Afouras, Effrosyni Mavroudi, Katherine Xu, Tsung-Han Wu, Yu Zhou, Liliane Momeni, Rishi Hazra, Shuangrui Ding, Sagar Vaze, Francois

Porcher, Feng Li, Siyuan Li, Aishwarya Kamath, Ho Kei Cheng, Piotr Dollar, Nikhila Ravi, Kate Saenko, Pengchuan Zhang, and Christoph Feichtenhofer. Sam 3: Segment anything with concepts. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/e0982cbc81401df3430ee1ff780dc7a2-Abstract-Conference.html.

Guanxiong Chen, Qianjun Xia, Jiawei Peng, Heng Zhang, Pengyu Jing, Bole Ma, Justin Qian, Yixian Cheng, Ziyi Jiao, Bingyang Zhou, Yiduo Qu, Luoxin Ye, Kaifeng Zhang, Kunyi Wang, Weijia Zeng, Yunuo Chen, Pengzhi Yang, Ziqiu Zeng, Siyuan Luo, Huamin Wang, Chao Liu, Alan Yuille, Fan Shi, Changxi Zheng, Yunzhu Li, Chenfanfu Jiang, and Peter Yichen Chen. Agentic real2sim: Physics-based world modeling with vision-language agents, 2026a. URL https://arxiv.org/abs/2607.19190.

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, 2026b. URL https://arxiv.org/abs/2609.10522.

Tianyuan Dai, Josiah Wong, Yunfan Jiang, Chen Wang, Cem Gokmen, Ruohan Zhang, Jiajun Wu, and Fei-Fei Li. Automated creation of digital cousins for robust policy learning. In *Proceedings of The 8th Conference on Robot Learning*, volume 270, pp. 4912–4943, 2025. URL https://proceedings.mlr.press/v270/dai25a.html.

Hongwei Fan, Hang Dai, Jiyao Zhang, Jinzhou Li, Qiyang Yan, Yujie Zhao, Mingju Gao, Jinghang Wu, Hao Tang, and Hao Dong. Twinaligner: Visual-dynamic alignment empowers physics-aware real2sim2real for robotic manipulation, 2025. URL https://arxiv.org/abs/2512.19390.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Fei-Fei Li, Guanya Shi, Jiajun Wu, S. Shankar Sastry, Yuke Zhu, Ken Goldberg, and Jim Fan. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. In *Forty-third International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/66369.

Dibya Ghosh, Homer Rich Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Quan Vuong, Ted Xiao, Pannag R. Sanketi, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Robotics: Science and Systems XX*, 2024. doi: 10.15607/rss.2024.xx.090. URL https://doi.org/10.15607/rss.2024.xx.090.

Xiaoshen Han, Junqiu Yu, Minghuan Liu, Yilun Chen, Xiaoyang Lyu, Yang Tian, Bolun Wang, Weinan Zhang, and Jiangmiao Pang. Re$^3$Sim: Generating high-fidelity simulation data via 3d-photorealistic real-to-sim for robotic manipulation, 2025. URL https://arxiv.org/abs/2502.08645.

Siyuan Huang, Zhengkai Jiang, Hao Dong, Yu Qiao, Peng Gao, and Hongsheng Li. Instruct2act: Mapping multi-modality instructions to robotic actions with large language model, 2023a. URL https://arxiv.org/abs/2305.11176.

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 *Proceedings of The 7th Conference on Robot Learning*, volume 229, pp. 540–562, 2023b. URL https://proceedings.mlr.press/v229/huang23b.html.

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, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of The 6th Conference on Robot Learning*, volume 205, pp. 1769–1782, 2023c. URL https://proceedings.mlr.press/v205/huang23c.html.

brian ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine,

Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as i can, not as i say: Grounding language in robotic affordances. In *Proceedings of The 6th Conference on Robot Learning*, volume 205, pp. 287–318, 2023. URL https://proceedings.mlr.press/v205/ichter23a.html.

Guangqi Jiang, Haoran Chang, Ri-Zhao Qiu, Yutong Liang, Mazeyu Ji, Jiyue Zhu, Xueyan Zou, Zhao Dong, and Xiaolong Wang. GSWorld: Closed-loop photo-realistic simulation suite for robotic manipulation. In *2026 IEEE International Conference on Robotics and Automation (ICRA)*, 2026. URL https://ras.papercept.net/conferences/conferences/ICRA26/program/ICRA26_ContentListWeb_3.html#tui1i_48.

Haoqiang Kang, Xiaokang Ye, Yuhan Liu, Siddhant Hitesh Mantri, Lingjun Mao, James Fleming, Drishti Regmi, and Lianhui Qin. Simworld studio: Automatic environment generation with evolving coding agent for embodied agent learning, 2026. URL https://arxiv.org/abs/2605.09423.

Pushkal Katara, Zhou Xian, and Katerina Fragkiadaki. Gen2sim: Scaling up robot learning in simulation with generative models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024. doi: 10.1109/ICRA57147.2024.10610566. URL https://doi.org/10.1109/ICRA57147.2024.10610566.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. In *Proceedings of The 8th Conference on Robot Learning*, volume 270, pp. 2679–2713, 2025. URL https://proceedings.mlr.press/v270/kim25c.html.

Ruiying Li, Yunlang Zhou, Yuyao Zhu, Kylin Chen, Jingyuan Wang, Sukai Wang, Kongtao Hu, Minhui Yu, Bowen Jiang, Zhan Su, Jiayao Ma, Xin He, Yongjian Shen, Yang Yang, Guanghui Ren, Maoqing Yao, Wenhao Wang, and Yao Mu. Roboclaw: An agentic framework for scalable long-horizon robotic tasks, 2026. URL https://arxiv.org/abs/2603.11558.

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, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. In *Proceedings of The 8th Conference on Robot Learning*, volume 270, pp. 3705–3728, 2025. URL https://proceedings.mlr.press/v270/li25c.html.

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)*, 2023. doi: 10.1109/ICRA48891.2023.10160591. URL https://doi.org/10.1109/ICRA48891.2023.10160591.

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, 2026. URL https://arxiv.org/abs/2607.00272.

OpenAI. Openai codex python sdk, 2026. URL https://github.com/openai/codex/blob/main/sdk/python/README.md. Official SDK documentation, accessed 2026-09-26.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018. doi: 10.1109/ICRA.2018.8460528. URL https://doi.org/10.1109/ICRA.2018.8460528.

Nicholas Pfaff, Evelyn Fu, Jeremy Binagia, Phillip Isola, and Russ Tedrake. Scalable real2sim: Physics-aware asset generation via robotic pick-and-place setups. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2025. doi: 10.1109/IROS60139.2025.11246653. URL https://doi.org/10.1109/IROS60139.2025.11246653.

Nicholas Pfaff, Thomas Cohn, Sergey Zakharov, Rick Cory, and Russ Tedrake. Scenesmith: Agentic generation of simulation-ready indoor scenes. In *Forty-third International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/63465.

Nadun Ranawaka, Josiah Wong, Wei-Lin Pai, Wei-Teng Chu, Tianyuan Dai, Masoud Moghani, Hang Yin, Yunfan Jiang, Wesley Durbano, Brandon Huynh, Yu Fang, Danfei Xu, Ruohan Zhang, Li Fei-Fei, Linxi Fan, Bowen Wen, Ajay Mandlekar, and Yuke Zhu. Simfoundry: Modular and automated scene generation for policy learning and evaluation, 2026. URL https://arxiv.org/abs/2606.28276.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017. doi: 10.1109/IROS.2017.8202133. URL https://doi.org/10.1109/IROS.2017.8202133.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109. URL https://doi.org/10.1109/IROS.2012.6386109.

Jianyuan Wang, Minghao Chen, Shangzhan Zhang, Nikita Karaev, Johannes Schönberger, Patrick Labatut, Piotr Bojanowski, David Novotny, Andrea Vedaldi, and Christian Rupprecht. Vggt-$\omega$. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21486–21499, 2026a. URL https://openaccess.thecvf.com/content/CVPR2026/html/Wang_VGGT-ohm_CVPR_2026_paper.html.

Yian Wang, Han Yang, Minghao Guo, Xiaowen Qiu, Johnson (Tsun-Hsuan) Wang, Wojciech Matusik, Joshua B. Tenenbaum, and Chuang Gan. Physcensis: Physics-augmented llm agents for complex physical scene arrangement. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/8797d13e5998acfab387d4bf0a5b9b00-Abstract-Conference.html.

Yuxuan Wu, Lei Pan, Wenhua Wu, Guangming Wang, Yanzi Miao, Fan Xu, and Hesheng Wang. RL-GSBridge: 3d gaussian splatting based real2sim2real method for robotic manipulation learning. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, 2025. doi: 10.1109/ICRA55743.2025.11128103. URL https://doi.org/10.1109/ICRA55743.2025.11128103.

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, 2026. URL https://arxiv.org/abs/2606.19980.

Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, Chris Callison-Burch, Mark Yatskar, Aniruddha Kembhavi, and Christopher Clark. Holodeck: Language guided generation of 3d embodied ai environments. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. doi: 10.1109/CVPR52733.2024.01536. URL https://openaccess.thecvf.com/content/CVPR2024/html/Yang_Holodeck_Language_Guided_Generation_of_3D_Embodied_AI_Environments_CVPR_2024_paper.html.

Shujie Zhang, Jingkun Yi, Weipeng Zhong, Zirui Zhou, Yangkun Zhu, Hanqing Wang, Xudong Xu, Weinan Zhang, and Chunhua Shen. Robosnap: One-shot real-to-sim scene generation for generalizable robot learning and evaluation, 2026. URL https://arxiv.org/abs/2607.06699.

Haoyu Zhao, Cheng Zeng, Linghao Zhuang, Yaxi Zhao, Shengke Xue, Hao Wang, Xingyue Zhao, Zhongyu Li, Kehan Li, Siteng Huang, Mingxiu Chen, Xin Li, Deli Zhao, and Hua Zou. High-fidelity simulated data generation for real-world zero-shot robotic manipulation learning with gaussian splatting. *IEEE Robotics and Automation Letters*, 11(5):5310–5317, 2026. doi: 10.1109/LRA.2026.3671535. URL https://doi.org/10.1109/LRA.2026.3671535.

Matt Zhou, Ruining Li, Xiaoyang Lyu, Zhaomou Song, Zhening Huang, Chuanxia Zheng, Christian Rupprecht, Andrea Vedaldi, and Shangzhe Wu. Articraft: An agentic system for scalable articulated 3d asset generation, 2026. URL https://arxiv.org/abs/2605.15187.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of The 7th Conference on Robot Learning*, volume 229, pp. 2165–2183, 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

# A Task Catalog and ID Mapping

The task identifiers below are used throughout the scene and policy evaluations.

Table 5: Task IDs and descriptions for the nine Franka FR3 and nine Agilex Piper tasks.

| ID | Franka task | ID | Piper task |
|---|---|---|---|
| F1 | Close the drawer | P1 | Close the cup lid |
| F2 | Put corn in the bucket | P2 | Put the brown block in the white-and-green storage box |
| F3 | Put the cup in the box and close the box | P3 | Put the emergency-stop button in the green storage box |
| F4 | Stack the snacks | P4 | Put the glue in the clear cup |
| F5 | Stack two bowls | P5 | Put the glue in the pen holder |
| F6 | Store fruit in the bamboo tray | P6 | Stack the brown blocks |
| F7 | Store fruit on the plate | P7 | Stack the cones |
| F8 | Store three tools | P8 | Stack the storage boxes |
| F9 | Store two beanbags | P9 | Take out the tissue |

# B Scene Reconstruction and Experiment Setup

## B.1 Registration objective and settings

We align the meter-scale robot surface to the reconstructed robot point cloud by estimating a similarity transform $T(\mathbf{x}) = sR\mathbf{x} + \mathbf{t}$, where $s$ converts meters to VGGT units (Wang et al., 2026a). Candidate transforms are scored using projected robot masks, bidirectional nearest-neighbor distances, point-cloud coverage, and depth consistency. Let $\overline{I}$ and $\overline{R}$ be mean mask IoU and recall, and let $r$ be the Euclidean distance between the componentwise 5th and 95th percentiles of the observed robot point cloud. For each direction, $\rho$ is the 60th-percentile nearest-neighbor distance divided by $r$; $O$ is the smaller fraction of points whose nearest-neighbor distance is below $0.05r$. Let $d_{50}^{\mathrm{VGGT}}$ be the median across frames of the median absolute projected-depth residual on valid robot-mask/depth overlap. The dimensionless ranking loss is

$$L = 0.50(1-\overline{I}) + 0.12(1-\overline{R}) + 0.14\min(5\rho_{m\to o}, 1) + 0.08\min(5\rho_{o\to m}, 1) + 0.08(1-O) + 0.08\min(d_{50}^{\mathrm{VGGT}}/r, 1).$$

When no valid depth overlap is available, the last term uses its maximum penalty. Table 6 summarizes the registration settings.

Table 6: Settings for robot-based metric-scale registration.

| Setting | Value |
|---|---|
| Sampled frames per capture | 32 |
| Rotation initialization | 128 candidates; refine the top 3 and select the lowest final loss |
| Initial alignment | Multi-scale Sim(3) ICP |
| Pose refinement | Two passes over translation, rotation, and scale, followed by coarse-to-fine rotation search |

## B.2 Registration summary

Table 7 summarizes registration fit across nine captures per robot. $L_f$ is the final ranking loss; IoU is averaged across frames. The task-median projected-depth residual $d_{50} = 10^3 d_{50}^{\mathrm{VGGT}}/s$ is in millimeters, where $s$ converts meters to VGGT units (Wang et al., 2026a). These fit statistics are distinct from reference-pose error and downstream scene-quality metrics.

Table 7: Robot-registration results across nine captures per robot.

| Robot | $n$ | Mean $L_f$ | Median $L_f$ | $L_f$ range | Mean IoU | Task-median $d_{50}$ (mm) |
|---|---|---|---|---|---|---|
| Franka | 9 | 0.11364 | 0.11221 | 0.10624–0.12115 | **0.8438** | 35.3 |
| Piper | 9 | **0.10464** | **0.10373** | 0.09997–0.11216 | 0.8405 | **21.3** |

# C Complete Quantitative Results

## C.1 Per-scene reconstruction comparison

Table 8 lists per-scene metrics for each method; task IDs are defined in Table 5. Lab $\Delta E_{76}$ is lower-is-better, grayscale SSIM is higher-is-better, and – denotes an unreported metric. CD is symmetric Chamfer distance in meters, F1 uses a 1 cm threshold, and BBox is visible bounding-box center error in meters.

Table 8: Per-scene reconstruction metrics for Franka and Piper. Bold indicates the best comparable result among reported methods.

| Method | Metric | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Franka** | | | | | | | | | | |
| Re$^3$Sim | Lab | 20.609 | 25.939 | 30.323 | 26.030 | 25.114 | 24.360 | 29.146 | 30.490 | 26.289 |
| | SSIM | .5612 | .5334 | .4875 | .5280 | .5229 | .5028 | .4980 | .4708 | .5151 |
| RL-GSBridge | Lab | 25.854 | 29.812 | 31.825 | 29.304 | 25.436 | 27.795 | 31.520 | 31.740 | 27.745 |
| | SSIM | .5507 | .5155 | .4297 | .4980 | .5465 | .5034 | .4644 | .4872 | .5296 |
| RoboSimGS | Lab | 20.818 | 24.948 | 28.042 | 24.814 | 23.658 | 23.916 | 26.386 | 28.163 | 24.992 |
| | SSIM | .5758 | .5717 | .5233 | .5525 | .5589 | .5236 | .5511 | .5300 | .5405 |
| SimFoundry | Lab | 55.781 | 55.634 | 51.913 | 53.194 | 55.386 | 56.286 | 51.924 | 51.914 | 29.455 |
| | SSIM | .4865 | .5303 | .5431 | .5370 | .5235 | .4887 | .5454 | .5515 | .6215 |
| | Depth | .4808 | .4476 | – | .8517 | .5711 | 1.5177 | – | – | .4680 |
| | CD | .4756 | .2134 | .1003 | .1404 | .1612 | .2444 | .1160 | .0805 | .2485 |
| | F1 | .0032 | .0000 | .0462 | .0027 | .0000 | .0000 | .0000 | .0578 | .0132 |
| | BBox | .6709 | .2571 | .1631 | .1954 | .1922 | .2704 | .1539 | .1163 | .3892 |
| Agentic RSR | Lab | **11.80** | **10.23** | **8.27** | **8.42** | **10.41** | **7.98** | **8.69** | **11.69** | **8.83** |
| | SSIM | **.7116** | **.7649** | **.7748** | **.7183** | **.7468** | **.7585** | **.7632** | **.7263** | **.7263** |
| | Depth | **.100** | **.104** | .100 | **.145** | **.080** | **.107** | .097 | .114 | **.100** |
| | CD | **.0222** | **.0435** | **.0149** | **.0133** | **.0167** | **.0417** | **.0145** | **.0153** | **.0319** |
| | F1 | **.2936** | **.0351** | **.4363** | **.4375** | **.3212** | **.1913** | **.3949** | **.3588** | **.1037** |
| | BBox | **.0556** | **.0894** | **.0711** | **.0589** | **.0471** | **.1003** | **.0550** | **.0297** | **.0991** |
| **Piper** | | | | | | | | | | |
| Re$^3$Sim | Lab | 15.487 | 14.695 | 15.855 | 15.386 | 14.669 | 13.772 | 14.391 | 14.995 | 13.304 |
| | SSIM | .6420 | **.6501** | .6538 | .6516 | .6565 | **.6715** | .6565 | **.6319** | **.6688** |
| RL-GSBridge | Lab | 17.767 | 17.605 | 16.069 | 17.019 | 18.058 | 14.977 | 17.094 | 16.651 | 15.459 |
| | SSIM | .5863 | .5993 | .6421 | .6168 | .5498 | .6440 | .6081 | .6116 | .6404 |
| RoboSimGS | Lab | 16.638 | 15.982 | 16.747 | 15.649 | 15.688 | 14.792 | 15.766 | 16.181 | 14.583 |
| | SSIM | .6227 | .6341 | .6434 | .6501 | .6422 | .6486 | .6432 | .6228 | .6561 |
| SimFoundry | Lab | 73.283 | 70.678 | 74.325 | 72.335 | 73.594 | 65.375 | 74.235 | 69.383 | 73.186 |
| | SSIM | .3446 | .3515 | .3344 | .3565 | .3387 | .3564 | .3326 | .3542 | .3296 |
| | Depth | .1569 | .1731 | .1201 | .1291 | .1421 | .2595 | .1951 | .2144 | .1460 |
| | CD | .1778 | .1665 | .0589 | .0545 | .1151 | .1069 | .3370 | .1835 | .1441 |
| | F1 | .0000 | .0000 | .0000 | .0245 | .0000 | .0035 | .0000 | .0000 | .0000 |
| | BBox | .1973 | .2058 | .2294 | .0821 | .1580 | .1455 | .3792 | **.2321** | **.2006** |
| Agentic RSR | Lab | **12.58** | **14.50** | **11.61** | **12.40** | **11.84** | **11.25** | **13.07** | **12.51** | **12.57** |
| | SSIM | **.6421** | .6358 | **.6805** | **.6583** | **.6697** | .6706 | **.6577** | .6156 | .6604 |
| | Depth | **.084** | **.091** | **.093** | **.087** | **.124** | **.092** | **.148** | **.109** | **.129** |
| | CD | **.0082** | **.0359** | **.0350** | **.0132** | **.0092** | **.0077** | **.0161** | **.0806** | **.1245** |
| | F1 | **.6817** | **.3730** | **.0809** | **.5571** | **.7296** | **.7146** | **.2658** | **.4905** | **.1131** |
| | BBox | **.0174** | **.0864** | **.0772** | **.0451** | **.0405** | **.0324** | **.0371** | .2856 | .3367 |

## C.2 Scene-refinement ablations

We evaluate three variants on the same 18 reconstructed scenes, each with at most 10 refinement rounds. The model-swap variant uses GPT-5.6 Sol at high reasoning effort and passes the visual check on 10 scenes. The other two retain GPT-6 Astra at high reasoning effort while removing scalar metric feedback or initial calibration. Table 9 reports the delivered-scene evaluation for accepted scenes, the last valid evaluation for unsuccessful scenes, and the round-0 baseline when no valid refinement-round evaluation was produced.

Table 9: Per-scene results for three scene-refinement ablations across 18 tasks.

| Variant | Task | Pass | Last round | $D$ (m) | Lab $\Delta E_{76}$ | SSIM |
|---|---|---|---|---|---|---|
| GPT-5.6 Sol | F1 | No | 10 | 0.2110 | 10.79 | 0.6883 |
| GPT-5.6 Sol | F2 | Yes | 2 | 0.1198 | 10.75 | 0.7141 |
| GPT-5.6 Sol | F3 | No | 10 | 0.4005 | 15.46 | 0.6718 |
| GPT-5.6 Sol | F4 | No | 10 | 0.6543 | 27.77 | 0.5333 |
| GPT-5.6 Sol | F5 | Yes | 9 | 0.1389 | 8.89 | 0.7757 |
| GPT-5.6 Sol | F6 | No | 10 | 0.2516 | 11.46 | 0.7084 |
| GPT-5.6 Sol | F7 | No | 10 | 0.2916 | 8.34 | 0.7622 |
| GPT-5.6 Sol | F8 | Yes | 4 | 0.1479 | 7.62 | 0.7666 |
| GPT-5.6 Sol | F9 | No | 10 | 0.3517 | 12.23 | 0.6830 |
| GPT-5.6 Sol | P1 | Yes | 2 | 0.1209 | 14.55 | 0.6353 |
| GPT-5.6 Sol | P2 | Yes | 10 | 0.1498 | 12.92 | 0.6552 |
| GPT-5.6 Sol | P3 | Yes | 1 | 0.1085 | 14.31 | 0.6512 |
| GPT-5.6 Sol | P4 | Yes | 2 | 0.1410 | 14.66 | 0.6266 |
| GPT-5.6 Sol | P5 | Yes | 6 | 0.1394 | 12.77 | 0.6602 |
| GPT-5.6 Sol | P6 | No | 10 | 0.1767 | 16.72 | 0.5963 |
| GPT-5.6 Sol | P7 | No | 10 | 0.1431 | 14.00 | 0.6515 |
| GPT-5.6 Sol | P8 | Yes | 8 | 0.1262 | 13.29 | 0.6260 |
| GPT-5.6 Sol | P9 | Yes | 3 | 0.1415 | 13.07 | 0.6586 |
| No metric feedback | F1 | No | 0 | 0.9390 | 29.92 | 0.4805 |
| No metric feedback | F2 | No | 10 | 0.1938 | 11.06 | 0.7450 |
| No metric feedback | F3 | No | 10 | 0.5095 | 26.04 | 0.5102 |
| No metric feedback | F4 | No | 10 | 0.2596 | 8.41 | 0.7656 |
| No metric feedback | F5 | No | 10 | 0.1616 | 7.04 | 0.7406 |
| No metric feedback | F6 | No | 10 | 0.2577 | 7.37 | 0.7313 |
| No metric feedback | F7 | No | 10 | 1.7704 | 14.70 | 0.7027 |
| No metric feedback | F8 | No | 0 | 0.9501 | 33.34 | 0.4854 |
| No metric feedback | F9 | No | 10 | 0.2661 | 10.52 | 0.7183 |
| No metric feedback | P1 | Yes | 2 | 0.1209 | 11.97 | 0.6708 |
| No metric feedback | P2 | Yes | 6 | 0.1493 | 12.62 | 0.6610 |
| No metric feedback | P3 | No | 10 | 0.2154 | 16.49 | 0.6338 |
| No metric feedback | P4 | Yes | 3 | 0.1348 | 11.60 | 0.6616 |
| No metric feedback | P5 | Yes | 2 | 0.1251 | 13.76 | 0.6334 |
| No metric feedback | P6 | No | 10 | 0.1488 | 16.54 | 0.6345 |
| No metric feedback | P7 | Yes | 5 | 0.1421 | 11.18 | 0.6650 |
| No metric feedback | P8 | No | 6 | 0.2630 | 17.30 | 0.5665 |
| No metric feedback | P9 | Yes | 1 | 0.1350 | 12.25 | 0.6762 |
| No initial calibration | F1 | Yes | 7 | 0.1388 | 9.54 | 0.5850 |
| No initial calibration | F2 | No | 10 | 0.2074 | 10.32 | 0.7309 |
| No initial calibration | F3 | No | 10 | 0.2830 | 10.71 | 0.7349 |
| No initial calibration | F4 | No | 10 | 0.3699 | 10.36 | 0.7349 |
| No initial calibration | F5 | Yes | 7 | 0.1477 | 9.01 | 0.7552 |
| No initial calibration | F6 | No | 10 | 0.1599 | 8.65 | 0.6328 |
| No initial calibration | F7 | No | 10 | 0.1483 | 8.54 | 0.7833 |
| No initial calibration | F8 | No | 10 | 0.2331 | 6.83 | 0.7956 |
| No initial calibration | F9 | No | 10 | 0.2354 | 8.35 | 0.7550 |
| No initial calibration | P1 | No | 10 | 0.2061 | 15.24 | 0.5556 |
| No initial calibration | P2 | Yes | 4 | 0.1467 | 13.41 | 0.6021 |
| No initial calibration | P3 | No | 10 | 0.1719 | 13.48 | 0.6318 |
| No initial calibration | P4 | No | 9 | 0.3461 | 15.16 | 0.6151 |
| No initial calibration | P5 | No | 10 | 0.2426 | 15.07 | 0.5786 |
| No initial calibration | P6 | No | 10 | 0.1725 | 13.65 | 0.6270 |
| No initial calibration | P7 | No | 0 | 0.9969 | 18.21 | 0.5913 |
| No initial calibration | P8 | No | 10 | 0.2573 | 15.05 | 0.5488 |
| No initial calibration | P9 | No | 10 | 0.1863 | 14.18 | 0.6165 |

# D AGENT PROMPT TEMPLATES

This section reproduces the task-independent prompt templates, rather than one task's fully instantiated conversation. Braced fields are runtime substitutions; local paths, private run locations, and identifying metadata have been removed. Images and public execution reports supplied alongside a prompt are described in the templates but are not reproduced here.

## D.1 REAL-TO-SIM

**Stage 1: robot mask query.** The only Stage 1 prompt reproduced here is the robot-mask query.

```
robotic arm with base and camera
```

**Stage 2: visual planning.** The planner receives four reference RGB views with the following template.

```
# Stage 2 plan agent

You are the planning-only agent for a real-to-sim reconstruction task.

## Task
{task_description}

You receive exactly four RGB reference images with this turn. Use them together
with the task description to infer the task-relevant scene structure. Do not read
or search for depth, camera JSON, masks, Stage 1 metadata, Blender files, or any
other artifact. Do not edit any file except the requested plan document.

Stage 1 uses the {robot_name} robot base as the complete scene world origin: `{robot_name}`
↪  is
at `[0, 0, 0]`. Treat this as a fixed coordinate convention for all later task
objects. Do not propose translating the whole scene or changing the robot origin.

The {robot_name} arm, its camera mount, gripper, attached parts, and every other robot
or robot-mounted accessory are explicitly out of scope. Do not describe them as
task-object capabilities. The later geometry agent will preserve them without
your help.

Write only this document:

```text
{plan_path}
```

The document must contain exactly these sections, with useful concise content:

## Object hierarchy
List task objects and their meaningful visual subparts only.

## Approximate dimensions
For every task object in the hierarchy, estimate its visible bounding-box size as
`length × width × height`, using metres or centimetres. Include a plausible
range when the scale is uncertain, state whether the estimate is for the whole
object or a visible subpart, and give a short basis for the estimate (for
example, image proportions or a known task convention). Do not invent false
precision: use rounded values and explicitly mark low-confidence estimates.
The {robot_name} robot may be used only as a visual scale reference; it is not a task
object and must not receive a capability requirement.

## Required capabilities
State only capabilities needed by the task. For example, a button that is
grasped needs a recognizable external shape and graspable dimensions; it does
not need an internal switch mechanism unless the task explicitly requires it.

## Relative layout
Describe the visible spatial relationships between task objects, support surfaces,
containers, and background geometry.

## Necessary assumptions
Record only assumptions required to build a visually and geometrically useful
scene. Mark uncertainty instead of inventing hidden structure.

Do not create a `.blend`, script, render, evaluation, or any other output.
```

**Stage 2: iterative scene refinement.** The refinement agent also receives the four current renders, the approved plan, and scalar feedback. The no-metric-feedback and uncalibrated ablations modify parts of this prompt and should not be confused with the default condition.

```
# Stage 2 RGB scene refinement agent

You are the single optimization agent for this real-to-sim task:

```text
{task_description}
```

## Fixed scene convention

Stage 1 already established the {robot_name} base as the complete scene world origin:
`{robot_name}` is at `[0, 0, 0]`. Keep this coordinate convention. Do not translate,
rebuild, or modify the {robot_name} robot, its gripper, its camera mount, or any camera.
```

````
Place every task object relative to this fixed origin.

The original Stage 1 Blend is:

```text
{initial_scene}
```

The current working copy for this round is:

```text
{current_scene}
```

Never overwrite either file. The orchestrator will execute your edit script on a
copy and save the result as a new round-specific `.blend`.

The working scene is a Stage 2-only copy named
`initial_with_default_light.blend`. It contains a replaceable neutral lighting
collection named `Stage2_Lighting` with lights such as
`Stage2_DefaultLight_Key` and `Stage2_DefaultLight_Fill`. You may tune, remove,
or replace these lights and the World when matching the reference appearance.
Never modify the Stage 1 source file, the {robot_name} robot, or the four cameras.

## Plan and camera status

The approved task plan is stored at:

```text
{plan_path}
```

Its contents are:

```markdown
{plan_text}
```

The four cameras are already installed and frozen. Their current status is:

```text
{camera_status}
```

Use these cameras as fixed observation views. Do not change their transforms,
lenses, clipping, resolution, scene assignment, or camera data.

## RGB inputs and final metrics

The four reference RGB images are:
{reference_rgb_paths}

The current working scene has been rendered from the same four cameras to:
{current_rgb_paths}

The only numeric feedback provided to you is the scalar final metric file:

```text
{metric_path}
```

It contains overall depth MAE and overall Lab E76, together with their
acceptable ranges. Do not request, read, or use depth images, depth arrays,
depth error maps, camera JSON, masks, or any other Stage 1 metadata. The
programmatic evaluator is authoritative; visual inspection and scalar metrics
must be used together.

## Required optimization order

Work through these stages in order during this turn and across later turns:

1. **Geometry blockout**: create or correct the task objects, support surfaces,
containers, relative positions, scale, silhouettes, occlusion, and stable
contact. Use simple externally visible geometry first. Do not invent hidden
mechanisms that the task does not require.
2. **RGB self-check**: before editing, render the current scene yourself into:

```text
{round_dir}/agent_self_render/
```

Use this command when available:
````

````
```bash
real2sim-stage2 render-rgb --scene CURRENT_SCENE --output {round_dir}/agent_self_render
```

If the command is unavailable, use the repository-local Python fallback:

```bash
PYTHONPATH={repo_root}/src python -m real2sim.stage2.agent_runner render-rgb --scene
↪  CURRENT_SCENE --output {round_dir}/agent_self_render
```

Open and compare all four reference RGB views and all four current RGB views.
Identify the three largest visible mismatches and fix only those first.
3. **Appearance refinement**: after geometry is plausible, add task-relevant
materials, colors, roughness, and neutral scene lighting so the rendered RGB
views match the references. Do not let appearance work hide incorrect geometry.
4. **Final self-check**: render RGB again, compare all four views, and use the
latest scalar metric to decide whether another geometry or appearance change is
needed. Continue iterating until the geometry and visual metrics pass or the
round limit is reached.

## Edit contract

Write the smallest evidence-based Blender Python edit script to:

```text
{edit_script}
```

The orchestrator will execute it, freeze-check cameras and {robot_name}, save a new
`candidate.blend`, render all four views, and calculate the final metrics. Every
round's Blend must remain in its round directory.

If a Blender Lab MCP tool is available, prefer it. Otherwise use the repository
Blender CLI/Python fallback; never claim an MCP call that did not happen.

This repository uses Blender 5.2.x. When creating Principled materials, locate
the node by `node.type == "BSDF_PRINCIPLED"` or create a
`ShaderNodeBsdfPrincipled`; do not assume that a node named `Principled BSDF`
already exists.

At the end, return JSON with exactly these fields:

```json
{{
  "summary": "what changed and why",
  "changed_objects": ["object names"],
  "changed_categories": ["geometry", "materials", "lighting", "relative_layout"],
  "next_hypothesis": "what the next four-view RGB comparison should test"
}}
```

Maximum rounds: {max_rounds}
````

**Stage 3: physical-role classification.**

```
# Stage 3 physical-entity classification

You are preparing a Stage 2 Blender scene for a MuJoCo physics simulation. Decide the
↪  physical role of **each listed non-robot mesh** before scene compilation. Use the task
↪  description, Stage 2 plan, inventory and four rendered views. This is a hypothesis
↪  about geometry and physical contact, not a measurement of mass or friction.

For every mesh, choose exactly one role:
- `visual_only`: room walls, curtains, backdrops, distant decoration and labels that cannot
↪  support or constrain task objects. They remain visible but **do not need a physical
↪  body or collision**. A wall that is part of a reachable container is not a backdrop.
- `static`: reachable tabletops/work surfaces, tray or container floors and walls, and any
↪  other surface needed to support or constrain manipulated objects. They MUST retain
↪  collision, even if visually similar to a non-colliding room backdrop.
- `dynamic`: a graspable independent rigid object with a free joint and box collision.
- `operable`: a task part with a real hinge/slide degree of freedom, only if the geometry
↪  and task make the joint plausible. Decorative pieces of the same object should be
↪  `visual_only`, not separate moving joints.
- `compound`: a **collidable fixed part attached to a moving entity** (basket walls/rims, a
↪  rigid gripper-relevant handle or structural box side). It has no independent joint; its
↪  box collider follows the owner's body. Do NOT use `visual_only` for a wall needed to
↪  contain items or support stacked containers merely because it is a child piece. Room
↪  walls and curtains remain `visual_only`.
```

```
Choose `collision` from `none`, `box`, `mesh`, `open_container`. Use `none` only for
↪  `visual_only`, use `box` for `compound`, prefer `box` for physical meshes with flat or
↪  thin geometry; never use `mesh` for a freely movable mesh. MuJoCo treats a mesh collider
↪  as a convex hull: for a task-relevant hollow cup, bowl, bucket, or pen holder, use
↪  `open_container` instead of a solid box or mesh, so a bottom and four wall proxies leave
↪  an accessible interior. This is only an approximation: note its limitations in the
↪  reason. Choose `joint_type` from `fixed`, `free`, `hinge`, `slide`: it must be `fixed`
↪  for `static`/`visual_only`/`compound`, `free` for `dynamic`, and `hinge` or `slide` for
↪  `operable`. Be conservative when uncertain: preserve appearance without inventing
↪  physics. Do not classify the robot or any of its cameras. Return **every source_name
↪  exactly once** and a short reason for each. No physical mass or friction measurement is
↪  available.

For visual pieces of a moving entity (e.g. labels, decorative cap), set `attached_to` to the
↪  **exact source_name** of that entity's `dynamic` or `operable` mesh. They will move with
↪  that entity without additional collision. For structural parts that need contact (for
↪  example open basket walls), choose `compound` and set `attached_to` to their owner, so
↪  they move with collision but no separate joint. Set `attached_to` to null for backdrops,
↪  room walls, standalone static support surfaces, and primary physical entities. Do not
↪  leave visual fragments of a movable object fixed in world space.

Task: {task_description}

Stage 2 plan:
{plan}

Meshes (dimensions and positions in metres, dimensions are approximations):
{inventory}
```

**Stage 3: image-only physical prior.** This turn is issued only when the compiled scene contains a free-joint object.

```
# Stage 3 visual physical prior

Before seeing the simulator's masses, friction or grasp trials, estimate a **rough prior**
↪  for ONE task-relevant movable object. Use only the attached third-person image, the
↪  task description and the Stage 2 visual plan below. Choose an exact body name from the
↪  given list. Ignore the robot and background. Consider apparent size, possible material,
↪  whether the item is hollow or heavy, and how a 1.0-friction, two-finger proxy might
↪  lift it. Record broad plausible ranges rather than pretending to know the real mass or
↪  the real gripping threshold. The same apparent object can have widely different mass
↪  and friction; explicitly describe that uncertainty. The per-finger actuator limit is 10
↪  N; estimates outside that capability may have a high endpoint above 10 N but may not be
↪  tested. Do not inspect the scene XML, mass values, physics reports or workspace files
↪  in this turn. This is not a real-world measurement or calibration.

Allowed body names: {body_names}

Task and visual plan:
{task_context}
```

**Stage 3: physical plausibility and repair.** This prompt is sent after scene compilation; a failed independent check may append a subsequent repair-feedback turn in the same conversation.

```
# Stage 3 physical plausibility agent

You are an independent Codex session assessing a Stage 3 MuJoCo tabletop scene **before**
↪  any downstream manipulation task. Your objective is to find and, only when justified,
↪  repair obviously implausible physics while preserving the original scene. A simulation
↪  sanity check cannot establish agreement with reality: distinguish prior guesses,
↪  measured real-world evidence, and simulation outputs.

Inputs below provide the absolute scene path, immutable prior-hypotheses JSON, and a
↪  third-person overview. The overview is an independent free camera, **not** one of the
↪  four reference cameras. Inspect the attached image and the observation JSON; query the
↪  scene yourself. Do not assume the visual appearance proves mass or friction. If a
↪  visual force prior path is given, read it before querying physical properties: it was
↪  estimated in a separate turn using only the image and Stage 2 plan. Treat it as a
↪  broad, uncertain hypothesis, never as a measured target.
```

```
Collision repair tools: `python -m real2sim.stage3.physics_probe patch --scene SCENE
↪  --patch-file OUTPUT/repair.json --output OUTPUT/candidate.xml` accepts named task
↪  bodies with `collision` (a dict with `geom`, `mode`: `none` for a non-supporting static
↪  backdrop only, or `box` plus `pos` and positive half-`size` in metres), `center_of_mass:
↪  [x,y,z]` in body-local coordinates, and `mass_kg`. Change only a named collision geom;
↪  never remove a tabletop/container support or any moving body's collider. A backdrop's
↪  visible mesh remains intact when its collision geom is removed. For a folded
↪  paper/pouch, a visual mesh can be left untouched while a flat thin contact box
↪  represents only the exposed contact area; place its mass center over actual support
↪  (not simply lower z). These are hypotheses: test one cause at a time (wrong backdrop,
↪  contact proxy, center of mass, mass) in separate candidate XMLs and static reports. A
↪  low center of mass alone cannot resolve collider penetration. Inspect initial contacts,
↪  verify that every candidate loads, then run the baseline and candidate static/grasp
↪  probes on matching settings. Grasp at modest forces including 0.2, 1, 3 N where
↪  plausible and do not assume higher always succeeds. Render the final candidate from all
↪  four reference cameras and check it against the original four views; keep the original
↪  visible geometry, camera poses, robot and accepted Stage 2 input unchanged. Save the
↪  working candidate only if it passes independent static and grasp tests; otherwise
↪  explain the failure, do not claim repair.

Workflow (actually execute these tools, do not merely describe them):
1. Read the hypotheses and observation JSON, and run `python -m
↪  real2sim.stage3.physics_probe objects --scene SCENE`. List task objects, their pose,
↪  mass, friction, mobility, and which prior is testable. Fixed bodies must not be
↪  reported as dynamically validated.
2. Run `python -m real2sim.stage3.physics_probe static --scene SCENE --seconds 3 --output
↪  OUTPUT/static`. Check every movable task object for spontaneous translation/rotation. A
↪  scene that lets an unsupported object fall or a tabletop object roll away is not ready.
↪  Inspect report.json even if the command exits 2.
3. Probe mode: {probe_mode}. If a **free-joint** object exists, test the object named in
↪  the visual prior with `python -m real2sim.stage3.physics_probe grasp --scene SCENE
↪  --body BODY --normal-forces 1 3 10 --output OUTPUT/grasp_BODY`. The two jaws are
↪  symmetrically coupled 0.025 kg pads with 1.0 sliding friction, 200-gain position servos
↪  and a hard **10 N actuator limit per jaw**; `--normal-forces` selects a per-jaw actuator
↪  force ceiling in (0, 10] N, not a measured contact force. Inspect actual contact
↪  forces, contact/hold/rise/slip; bracket the lowest successful tested setting using
↪  additional 2, 4, 6, etc. N trials where informative. Compare this bracket to the visual
↪  prior, including its confidence and uncertainty. Never request above 10 N or claim that
↪  this proxy proves a real Piper grasp. Failure at 1 N is normal for many objects, not a
↪  repair trigger; a higher-force success may support keeping the original parameters. Use
↪  distinct output directories for each trial and report the most informative result; a
↪  stronger trial failing does not invalidate an earlier success. If neither jaw contacts
↪  the object, or the jaws miss or slide off a container wall, diagnose geometry and try
↪  `--grasp-axis y` instead of the default `x`; if the pads shoot through thin walls, try
↪  `--jaw-damping 100` (default 2) before changing mass or friction. If no free-joint task
↪  object exists, DO NOT invent one or attempt `grasp`. Instead, test an existing
↪  hinge/slide task object: write OUTPUT/articulation_script.json with a time-limited
↪  `force` or `torque` action on the task body and an explicit displacement/rotation check,
↪  then run `python -m real2sim.stage3.physics_probe run --scene SCENE --script
↪  OUTPUT/articulation_script.json --output OUTPUT/articulation`. Choose a physically
↪  modest force, inspect actual response and stability, then adjust only with evidence.
↪  This is a simplified simulation probe, not a real robot actuator calibration.
```

```
4. Only if the original has a clear, well-supported mismatch with the broad prior or a
↪  static/geometry failure, consider a repair; do not tune to the visual prior as though
↪  it were ground truth. Grasp force alone cannot uniquely identify mass versus contact
↪  friction. For an initial interpenetration or unsupported object, inspect the contacts,
↪  the four Stage 2 reference RGB images listed in the task context, and the task
↪  geometry. If the object's initial pose is genuinely wrong, `patch` can change `position:
↪  [x,y,z]` and/or `quaternion: [w,x,y,z]` on its task body; the patch also synchronizes
↪  free-joint `stage2_initial` keyframe coordinates. Test the smallest visually defensible
↪  change as a separate candidate; render its four cameras with `mjpython -m
↪  real2sim.stage3.render_overview --scene OUTPUT/candidate.xml --output
↪  OUTPUT/candidate_views --reference-views`, compare against the original reference
↪  images, and retain only if it improves physical plausibility without losing visual
↪  alignment. Never move the robot or a correctly placed object just to make a collision
↪  test pass: if its Stage 2 pose is correct and the backdrop, tabletop or container
↪  collider is the cause, report that collider for geometry repair instead. Do not trade
↪  away four-view alignment for a simulator pass. Check mass against appearance, support
↪  and motion separately from friction/slip with bilateral contact; fix contact geometry
↪  before attributing failure to either. For a justified mass/friction change, test **one
↪  parameter at a time** using `python -m real2sim.stage3.physics_probe try` in separate
↪  directories (mass and friction are distinct candidates), compare with the unchanged
↪  original under matched conditions, then write only the chosen change to
↪  OUTPUT/candidate.xml via `patch`. Rerun `static` and `grasp` on both original and
↪  candidate at the SAME force settings, and report contact, hold, rise and slip
↪  before/after. Do not substitute direct upward force at the object's center for grasp
↪  testing; `try` is a diagnostic, not a substitute for matched grasps. `patch` also
↪  supports `joint_axis: [x,y,z]` for hinge/slide task joints when observed motion
↪  contradicts task geometry; rerun the applicable articulated `run` probe for a candidate.
↪  Never modify SCENE, priors, source code, or another workspace path. If the original is
↪  stable and its grasp bracket is plausible, **retain it unchanged**; do not generate a
↪  candidate just to claim optimization. Without matched real-world observations, neither a
↪  successful grasp nor a visual guess validates the true mass or friction.

Return the required JSON. `static_report` must be an absolute path ending in `report.json`.
↪  For free bodies, set `grasp_report` to a report path and `articulation_report` to null.
↪  For joint-only scenes, set `grasp_report` to null and `articulation_report` to the `run`
↪  report path. Do not append commentary to report paths; put findings in `findings`.
↪  `objects_checked` must contain body names only. Set `candidate_scene` to null if any
↪  candidate fails matched static or grasp rechecks; describe the failed experiment in
↪  `findings` instead and retain the original. In particular, do not promote a mass-only
↪  candidate while an initial collider penetration remains unresolved. Set
↪  real_world_verified=false: the current runner has no independent real-world measurement
↪  comparator, even if priors mention measurements. Set status=provisional when simulator
↪  sanity tests pass; use needs_repair for unresolved sanity failures. Never claim a
↪  successful manipulation or real-world alignment based solely on these probes.

Scene: {scene}
Prior hypotheses: {hypotheses}
Observation JSON: {observation}
Pre-physics visual estimate: {visual_prior}
Output directory: {output}
Task and Stage 2 scene plan (read before choosing objects to test):
{task_context}
```

**Stage 3: independent-check feedback.** If the candidate fails verification, the same conversation receives this repair turn with the check results substituted at runtime.

```
                    feedback = (
                        f"Independent check for turn {attempt + 1}/{MAX_REPAIR_TURNS}: "
                        + json.dumps(verification, ensure_ascii=False, default=str)
                        + "\nYour previous conclusion: " +
                        ↪  json.dumps(record["agent_result"], ensure_ascii=False)
                        + "\nThe scene is NOT physically ready. Continue repairing in this
                        ↪  SAME session; "
                          "do not accept a failed candidate. Preserve the original and use
                          ↪  NEW candidate filenames "
                          "and unique probe directories. Inspect contact penetration and
                          ↪  support geometry: "
                          "a non-supporting backdrop may need its collider removed, but
                          ↪  removing it alone may "
                          "expose an oversized thin-sheet collision proxy. Repair the
                          ↪  contact proxy and "
                          "body-local center of mass over actual support; assess mass
                          ↪  independently. "
                          "Derive values from geometry and contacts, do not assume preset
                          ↪  numbers. "
                          "Re-run matched static and two-pad grasp tests and render four
                          ↪  reference views "
                          "for the final candidate. Return the full structured JSON again."
```

```
                    )
```

### D.2 SIM POLICY ITERATION

The three stages are `privileged_exact`, `visual_exact`, and `visual_randomized`, in that order. They use *one* authoring template with a stage-specific access note and optional preceding-stage policy and experience; there are not three independent fixed system prompts.

**Shared three-stage authoring template.**

```
def task_prompt(robot, task, context, names, practice_socket=None, *,
                stage="visual_exact", previous_policy=None, prior_experience=None):
    privileged = stage == "privileged_exact"
    stage_note = (
        "This stage exposes api.object_poses() and CLI object_poses()."
        if privileged else
        "Object ground truth is unavailable; use api.observe() only."
    )
    prior_policy = ("\nPRIOR STAGE POLICY\n"
                    + previous_policy
                    + "\nTreat this as a draft: preserve useful structure, but re-observe
                    ↪  and "
                      "remove hard-coded coordinates that are not justified in this stage."
                    if previous_policy else "")
    prior = ("\nPRIOR STAGE EXPERIENCE\n"
             + prior_experience + "\nUse this as development notes, not as simulator truth."
             if prior_experience else "")
    practice = f"""
Practice is available through one command per response. The harness returns JSON and both
RGB images. Use state, observe, move_pose, gripper, move_trajectory, reset, return_control,
stop{', object_poses' if privileged else ''}. Submit policy.py when ready.
""" if practice_socket is not None else ""
    return f"""You are authoring policy.py for the {robot} SIM task.

TASK
name: {task}
description:
{_prompt_context(context)}

QUERY NAMES
{json.dumps(names, ensure_ascii=False)}
These are SAM3 query identifiers, not known detections or positions.

STAGE
{stage}: {stage_note}
The order is privileged exact -> visual exact -> visual randomized. Randomization starts
↪  only
in the last stage and affects object poses, camera calibration and pose execution.
{prior_policy}{prior}

OUTPUT
Return JSON with mode, command and files. Use mode="practice", command=<one CLI command>,
files=[] while exploring. Use mode="policy", command="", files=[{{"path":"policy.py",
"source":"..."}}] when revising code. policy.py must define def run(api):.
After choosing the current policy, return mode="execute", command="run_policy", files=[];
SIM will not execute a policy until you explicitly request this.

RULES
No imports, file access, simulator internals, hidden truth, private attributes, classes,
lambda, with blocks, globals, pick/place helpers, grasp helpers, joint overrides or success
oracles. Runtime has only an overall time budget; retry and reset as needed.

API
- api.state() -> EE pose, wrist camera pose, gripper width/target width, moving. No object
↪  poses.
- api.observe(queries) -> SAM3+depth estimates, wrist RGB, and fixed third-person RGB. Query
  names with a list or name-to-English-prompt mapping; missing/occluded objects are absent.
- api.robot.move_pose(x,y,z,quaternion_wxyz): world pose, quaternion order (w,x,y,z).
- api.robot.gripper(width_m): width 0..0.08; closing does not guarantee a grasp.
- api.robot.move_trajectory(waypoints): sparse world waypoints with optional gripper widths.
- api.reset(): restore the current stage's initial scene and try again.
- api.return_control(reason): return a checkpoint without resetting.
- api.stop(): stop motion; it is not proof of success.

EXECUTION
Start with state(), then observe. Re-observe after meaningful actions. Use only fresh
↪  runtime
```

```
observations for coordinates. Return status="completed" only after post-action evidence
↪  supports
the task. You may return status="checkpoint" with a reason when the outer Agent should
↪  inspect
the result and decide whether to continue, reset, revise policy, or stop. A checkpoint is
↪  not a
task failure. Otherwise return status="failed" with a reason. Two initial images are
↪  attached:
wrist first, third-person second. Do not read image files from policy.py.
{practice}
"""
```

**Practice feedback.** During authoring, a single practice CLI command and its public result are returned to the same conversation before policy submission.

```
                except (ValueError, SyntaxError) as exc:
                    response = f"Your proposed policy.py is invalid: {type(exc).__name__}:
                    ↪  {exc}. Repair it."
                    images = ()
            elif payload.get("mode") == "practice":
                command = payload.get("command", "")
                if practice_socket is None:
                    response = "Practice CLI is unavailable in this visual stage. Submit
                    ↪  policy.py."
                    images = ()
                elif not isinstance(command, str) or len(command) > 4096:
                    response = "Practice command must be a string of at most 4096
                    ↪  characters."
                    images = ()
                else:
                    try:
                        result = request_command(practice_socket, command)
                    except Exception as exc:
                        result = {"status": "error", "error": f"{type(exc).__name__}:
                        ↪  {exc}"}
                    log.write(json.dumps({"command": command, "result": result},
                                         ensure_ascii=False) + "\n")
                    log.flush()
                    response = f"Practice CLI result for {command!r}:\n{json.dumps(result,
                    ↪  ensure_ascii=False)}"
                    images = tuple(Path(result["images"][view]["path"])
                                   for view in ("wrist", "third_person")
                                   if result.get("images", {}).get(view, {}).get("path"))
            else:
                response = "Return mode=practice with one CLI command, or mode=policy with a
                ↪  valid policy.py."
                images = ()
            if time.monotonic() >= deadline:
                raise TimeoutError("SIM authoring time budget exceeded")
            payload = session.run(response + "\nUse the same JSON format with mode,
            ↪  command, files.",
```

**Explicit execution request.** The agent must request execution after proposing a policy.

```
    prompt = """policy.py is ready. Decide whether to run it now.
Return mode="execute", command="run_policy", files=[] to execute this exact policy.
If it needs changes, return mode="policy" with a replacement policy.py instead.
Do not use hidden state or assume execution succeeded before receiving its result."""
```

**Within-stage continuation and revision.** Runtime results and the public action log are substituted into this template; the wrist and third-person images may accompany the turn.

```
def continuation_prompt(robot, task, program, actions, reset_before_run):
    return f"""Continue the same SIM task-agent conversation for {robot}, task {task}.

The previous generated policy segment ended without a verified completion.
Runtime result:
{json.dumps(program, ensure_ascii=False, indent=2)}

Public action log (estimated observations only; no simulator truth):
{json.dumps(actions[-16:], ensure_ascii=False, indent=2)}
```

```
{"The harness reset the scene before this segment. Start a fresh attempt." if
↪  reset_before_run else "The scene was not reset. Continue from the current checkpoint
↪  and re-observe before acting."}

Return mode="execute", command="run_policy", files=[] to run the same policy again, or
↪  return
mode="policy" with a replacement policy.py. Use only the same public Stage 3 interface. Do
not assume any object pose from the previous segment; call observe() again. If the current
attempt becomes unrecoverable, call api.reset() and retry. You may return
{{"status": "checkpoint", "reason": "..."}} for the outer Agent to decide the next step
without resetting. If visual evidence is insufficient but the scene is still usable, call
api.return_control(reason) so the outer Agent can inspect the next checkpoint. Only return
{{"status": "completed"}} after fresh post-action evidence; otherwise return a concise
↪  failed
status. The fixed third-person image attached to this turn is for global verification and
the wrist image is for SAM3/depth localization.
"""
```

**Experience handoff between stages.** After a completed stage, the agent summarizes public evidence for the next stage's fresh conversation.

```
def experience_prompt(robot, task, stage, report):
    return f"""Summarize this completed {stage} SIM development stage for the next fresh
    ↪  Agent.

Task: {task}
Robot: {robot}
The next stage will start a new conversation and read your summary document first.

Runtime result:
{json.dumps(report.get("program"), ensure_ascii=False, indent=2)}

Independent stage result:
{report.get("physical_verdict")}

Full public action log:
{json.dumps(report.get("actions", []), ensure_ascii=False, indent=2)}

Write concise Markdown covering:
- what the task requires;
- observations and coordinates that were useful, clearly marked as estimates;
- successful motion, grasp, release and verification patterns;
- failures, IK or perception problems, and what caused them;
- what the next Agent should try or avoid.

Do not include simulator internals, hidden object truth, private identifiers, or unsupported
certainty. The attached images are the visual evidence; refer to what is visibly supported.
Return JSON with one field, experience_markdown.
"""
```