PHYSICAL AGENCY: WHAT THE INTERFACE UNLOCKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Frontier models increasingly act as robot-use agents, calling a robot's motor skills as tools much as they interact with a browser or code editor. But exposing skills as tools leaves open how the model should use them: what it emits, what feedback it receives, and when it decides again. We study the interface through controlled comparisons with the robot, cameras, scenes, and motor-policy weights held fixed, and separately measure the benefit of adding a complementary frozen skill. We introduce `Pigey` (**P**hys**i**cal A**ge**nc**y**), an agent that (1) issues **language-level actions** one at a time over a persistent task state, (2) uses **verified execution**, checking post-action images and robot state and recovering when it detects failure, and (3) draws on **an open set of skills** exposed as tools it selects for each subgoal. No additional policy training or demonstrations are required. On 30 real-robot tasks designed to probe decisions over existing motor skills, the same frozen $\pi_{0.5}$ policy succeeds in 16.7% of trials when called directly and 84.0% through `Pigey`. Adding a frozen motion planner as a second skill raises success to 97.3%. Removing verification and recovery lowers success by 28 percentage points, the largest ablation drop. Gains also hold in simulation and across Claude, Gemini, and Qwen reasoners. Most of what the policy was missing was an interface, not a skill.[1]

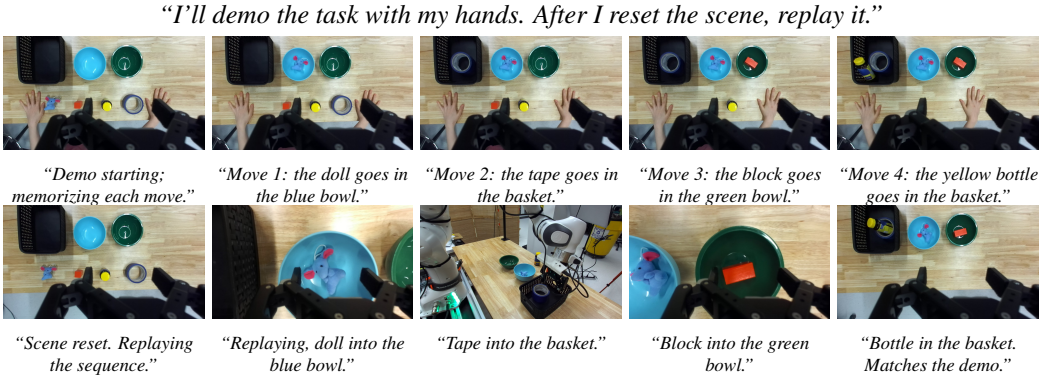


Figure 1: A single real-robot rollout of `Pigey`. A person moves four objects one at a time (top row), and the reasoner names each move as it happens. After the scene is reset (bottom row), the agent replays the sequence with the frozen skills, ending in the demonstrated arrangement. The motor policy is the same frozen $\pi_{0.5}$ that solves 16.7% of our 30 real-robot tasks on its own, 84.0% inside the interface, and 97.3% once a second frozen skill is added. The reasoner's decision at each step appears below the frame.

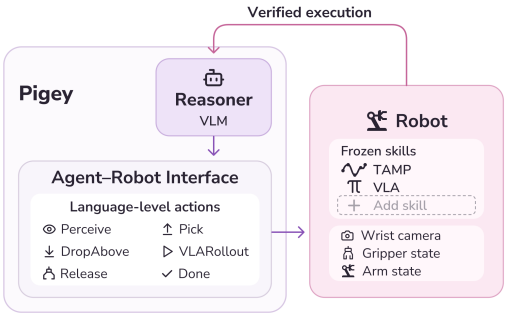
1 INTRODUCTION

Frontier models increasingly control robots by calling their existing motor skills as tools (Ahn et al., 2022; Liang et al., 2023; Shi et al., 2025a). This raises a question: once the reasoner and the motor skills are fixed, how should they be connected? In software, ReAct (Yao et al., 2023) and SWE-agent (Yang et al., 2024) demonstrated the importance of how models interact with their tools and environments. We study this question for robot manipulation: how much can task performance improve by changing the interface through which a model uses existing skills?

In robotics, a completed tool call does not guarantee its intended physical effect. A grasp may execute but miss the object. Moving one object may reveal another. A failed placement forces a

[1] Videos, rollouts, and code are in the supplementary material.

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.



Algorithm 1: The agent loop

$h \leftarrow$ instruction + initial observation;
for $t = 1, \ldots, T_{\max}$ **do**
  $a \leftarrow \phi(h, \mathcal{T})$; `// next tool call`
  if $a =$ DONE **then return** success iff predicate holds;
  if $a =$ PERCEIVE **then** append observation to h;
  if $a =$ PICK/DROPABOVE(ℓ) **then** run TAMP on ℓ;
  if $a =$ VLAROLLOUT(s) **then** run VLA on s;
  append wrist image, IS_GRASPED, labels, error to h;
return failure;

Figure 2: Left: `Pigey` and its interface. The reasoner (a frozen VLM) emits one language-level action per turn, and the interface runs the matching frozen skill and returns the wrist image, gripper state, and detected labels, which the reasoner checks before acting again. Adding a skill adds a tool description. **Right:** the loop.

different next action. The reasoner therefore needs more than motor skills. It needs observations to see what happened and chances to revise as the scene changes.

We examine three questions about the interface. The first is the *unit of decision*: should the model generate a program that coordinates multiple skill calls (Liang et al., 2023; Singh et al., 2023; Shi et al., 2025a; Fu et al., 2026), or select one action and decide again after observing its result? Generated programs can also use feedback, but the distinction is when task-level decisions return to the model. The second is the *execution-feedback loop*: what should the model observe, and how should it use those observations to verify outcomes and recover from failure? Feedback can include textual descriptions (Huang et al., 2022), images, and robot state. The third is the *skill set*: how much does adding a complementary frozen skill contribute beyond using an existing skill more effectively?

These questions can be difficult to assess across systems that use different robots, policies, and evaluation tasks. Therefore, we make controlled comparisons with the robot, cameras, scenes, and motor-policy weights held fixed. We first measure how action and feedback interfaces affect the use of existing skills, then separately evaluate the benefit of adding a complementary frozen skill.

We introduce `Pigey` (Figure 2, Algorithm 1), an agent that keeps task-level decisions with the reasoner and motor execution with frozen skills. The model issues one language-level action at a time, such as PICK(*cup*), using the interaction history as a persistent task state. After execution, it uses the wrist image and robot state to assess whether the action achieved its intended effect. When it detects a failure, it can re-perceive, retry, or choose another action. Skills are exposed as tools with descriptions that guide their selection, allowing an additional frozen skill to be incorporated into the same loop. Our experiments use a TAMP planner built on TiPToP (Shen et al., 2026) and a $\pi_{0.5}$ VLA (Physical Intelligence, 2025), without additional policy training or robot training data.

On 30 real-robot tasks designed to probe decisions over existing motor skills, the frozen $\pi_{0.5}$ policy succeeds in 16.7% of trials when called directly, 84.0% through `Pigey`, and 97.3% once a complementary TAMP skill is added. Removing verification and recovery is the largest ablation drop, at 28 points. The effect also holds in simulation. On LIBERO-PRO, `Pigey` reaches 55.0%, against 10.8% for the frozen policy and 14.6% for CaP-Agent0, and the gain carries across the Claude, Gemini, and open-weight Qwen reasoners, with matched comparisons favoring one action per turn over program generation (Section 5). These gains come from how existing motor skills are used, not from changing their weights. Most of what the policy was missing was an interface, not a skill.

This paper makes three contributions. First, **a controlled study of the agent–robot interface**: we evaluate the unit of decision, the execution-feedback loop, and the skill set, separating improvements in the use of existing frozen skills from the benefit of adding a complementary one. Second, **`Pigey`**, an agent that combines one language-level action at a time, persistent task state, and verified execution over an extensible set of frozen skills. Third, **evidence of generality**: the gain holds on 30 real-robot tasks, in simulation on LIBERO-PRO, and across frontier and open-weight VLMs.

2 Related Work

Pretrained robot policies and motor skills. Vision-language-action models extend imitation learning across tasks and embodiments (Black et al., 2025; Physical Intelligence, 2025; Kim et al., 2024;

Bjorck et al., 2025), supported by large demonstration datasets such as DROID (Khazatsky et al., 2024). Planning-based systems offer another source of motor skills: TiPToP (Shen et al., 2026) combines pretrained perception with classical task-and-motion planning. These approaches provide the capabilities that our agent uses. We keep the motor components frozen and study how their task performance depends on the interface through which a reasoner calls them.

Language models for robot planning. Language and vision-language models have been used to select, compose, and monitor robot skills. SayCan (Ahn et al., 2022) combines language-model scores with skill affordances, while Inner Monologue (Huang et al., 2022) supplies textual feedback to support planning. Code as Policies and ProgPrompt (Liang et al., 2023; Singh et al., 2023) generate programs that invoke robot skills, and VoxPoser (Huang et al., 2023) generates code that constructs 3D value maps for a motion planner. Maestro (Shi et al., 2025a) uses a VLM coding agent to compose robotics modules into closed-loop programmatic policies. Our work builds on this separation between reasoning and motor execution. We examine when control returns to the reasoner, what feedback it receives, and how verification and recovery shape its use of frozen skills.

Reasoning and hierarchical robot policies. Other approaches organize robot control through dual-system or task-decomposing architectures (Shi et al., 2025b; Bjorck et al., 2025; Google DeepMind, 2025; Chen et al., 2026a), or introduce memory, reasoning, and outcome-assessment modules (Pan et al., 2026; Zawalski et al., 2024; Kang et al., 2025; Liang et al., 2026; Ma et al., 2025). These works explore how additional structure and capabilities support task execution. We focus on the interaction between an existing VLM reasoner and frozen motor skills, without additional policy training. This lets us measure how action selection, feedback, and recovery change performance with the motor-policy weights fixed.

Agent interfaces. ReAct (Yao et al., 2023) interleaves reasoning with actions and observations, and SWE-agent (Yang et al., 2024) shows that designing the agent–computer interface can change task success without changing the model. The question carries over to robot agents, but the answer does not. In software, feedback is typically programmatic and explicit, such as exit codes, exceptions, and test results, so the interface's job is mainly to shape commands and feedback. A motor skill instead reports that it moved, not that the subgoal was achieved, and the physical state cannot simply be reset. The interface must therefore verify each outcome from sensors, recover from the failures it detects, and choose among skills that fail differently. We study how the interface supports these decisions: what observations are returned, and when task-level control returns to the model.

Controlled studies of robot-agent design. CaP-X (Fu et al., 2026) evaluates action abstraction, interaction patterns, and feedback modalities under matched conditions, finding that textualized visual feedback performs best in its setting. In our real-robot setting, image and state feedback outperform text-only feedback (Section 5.2). We examine verification and recovery on a real robot through ablations and comparisons with retry baselines, text-only feedback, and a per-task skill-selection oracle. In simulation, we also compare action-by-action interaction with program generation.

Concurrent work. Concurrent studies explore complementary aspects of robot-agent design. A systematic study of hierarchical VLA agents (Hu et al., 2026) examines design questions including how control passes between planner and controller. Harness VLA (Zhang et al., 2026) combines a frozen VLA with analytic primitives through a memory-guided agent. Show-Harness (Chen et al., 2026b) exposes fine-grained semantic actions to VLMs for zero-shot control with frontier models or control with fine-tuned smaller models. AGM (Gao et al., 2026) trains a verification head that gates task-progress updates around a frozen VLA. Our study focuses on separating the benefit of using an existing motor skill more effectively from that of adding a complementary frozen skill, and on measuring the roles of action-by-action interaction, verification, and recovery.

3 The Agent–Robot Interface

Robot policies provide useful, though imperfect, motor skills, and vision-language models provide strong reasoning. Having both does not by itself yield reliable execution: the reasoner has to direct the skills, observe what they actually accomplished, and revise its decisions when execution falls short. Any system that connects a reasoner to a robot answers three questions. What does the reasoner emit, and how much execution happens before it decides again? What does it see afterwards, and what does it do when the outcome is not what it intended? And when more than one skill could

carry out a subgoal, how does it choose? We call the answers the agent–robot interface, and the rest of this section addresses each in turn: we state the alternatives, describe the one `Pigey` adopts, and name the comparison that tests it in Section 5. We begin with the components and the loop that connects them. Appendix A gives the formal setup.

3.1 Components and the interaction loop

`Pigey` has three components, none of which is trained or fine-tuned. A **perceiver**, an open-vocabulary detector (Gemini Robotics-ER or Qwen, depending on the configuration), converts camera images into object labels and bounding boxes. A **reasoner**, a general-purpose VLM, reads the current image, the detected labels, the robot state, and the interaction history, and selects the next tool call. The **motor skills** execute it: on the real robot, a task-and-motion planner built on TiP-ToP (Shen et al., 2026) and a $\pi_{0.5}$ VLA (Physical Intelligence, 2025), which reach the same goals by different routes, geometric planning for the first and learned visuomotor control for the second (the VLA inference loop is in Appendix C).

The loop is simple. At each turn the reasoner selects one tool call. The interface executes it, runs perception, and returns the result together with the new observation, and the reasoner decides again. Calls and observations accumulate in the history. The episode ends when the reasoner calls DONE or exhausts a budget of $T_{\max}$ turns (Figure 2, Algorithm 1). Six tools are exposed: PERCEIVE, which re-runs detection without touching the scene; PICK(ℓ) and DROPABOVE(ℓ), which invoke TAMP on a detected label ℓ; VLAROLLOUT(s), which runs the VLA on a short language subgoal s; RELEASE, which opens the gripper; and DONE. Tool schemas and the agent prompt are in Appendices H and J.

3.2 Language-level actions over a persistent task state

The first question is how much the reasoner commits to before it gets to look again. At one extreme it drives the arm directly, emitting end-effector targets and gripper commands, so the model must supply the geometry of every motion itself. At the other extreme it writes a program that strings several skill calls together, with its own branches and loops, committing to more of the task before it next revisits its decisions. In between, it can issue a single skill-level action and wait for the result before choosing the next.

`Pigey` takes the middle option: one language-level action per turn, such as PICK(*cup*). The reasoner names *what* to do to *which* object, and the skill supplies the motion. Two consequences follow. First, arguments must be labels the detector returned, so a reference like "the unsafe object" or "the smallest one" has to be resolved to a concrete detection before it can be acted on. This grounding step is explicit and visible in the trace. Second, because the model decides again after every action, the interaction history becomes a persistent task state: which subgoals were attempted and how they ended, what the gripper holds, which objects are already in place. A 10–20 step sorting or childproofing task depends on this state to avoid repeating or skipping items.

To justify this design, Section 5.4 compares it against both extremes in simulation. The program-generation condition keeps the same model, tools, perception, frozen policy, and action budget, and changes only how actions are emitted: the model writes a program that calls the same tools and runs to completion before the model sees the outcome. The direct-control condition keeps the model, perception, and budget and replaces the toolset with raw end-effector commands.

3.3 Verified execution and recovery

Returning control after every action is worth little if the reasoner cannot tell what happened. A motion that completes is not a subgoal that was achieved: the gripper can close on nothing, or on the wrong object, and a placement can miss. The second question is therefore what the reasoner sees after an action and what it does when the evidence disagrees with its intent.

After each action the interface returns the post-action wrist image, a binary IS_GRASPED flag derived from gripper width, the refreshed detections, and any typed execution error such as `no_grasp_found` or `motion_planning_failed` (Appendix D). The reasoner is instructed to check the outcome against the intended one before proceeding, using both the structured state

and the image. After PICK, for example, a closed gripper is not enough if the wrist image shows the wrong object. This verification is a judgment made by the VLM and can itself be wrong. When it does detect a failure, the reasoner re-perceives and chooses a recovery that fits what it now sees: retry from the updated scene, switch to the other skill, return a wrongly grasped object, uncover an occluded target, or clear an obstructed destination (Figures 4 and 3; guidance in Appendix E).

The tool boundaries are drawn to make this checking possible. TiPToP natively executes pick-and-place as one open-loop plan, so a grasp that slips still triggers the planned placement. We expose grasp and place as separate tools, PICK and DROPABOVE, so the reasoner inspects the grasp before it commits to the place. The interface thus closes a loop around a skill that is open-loop on its own.

We test this feedback loop on the real robot with three comparisons (Section 5.2): retrying the same skill on failure (three retries per subgoal, more than `Pigey`'s one retry and escalation) but no checking, an Inner-Monologue-style planner (Huang et al., 2022) that closes the loop with a text description of the outcome instead of the image and state, and `Pigey` itself with verification and recovery removed.

3.4 AN OPEN SET OF SKILLS

Verification can establish that an attempt failed; it cannot make the same skill succeed at something it cannot do. The third question is what the reasoner does when its skill is the wrong one for the subgoal. `Pigey` answers by exposing every skill as a tool whose description says when to use it. On the real robot the descriptions state the known strengths of each skill, geometric grasp planning for rigid objects and the VLA for deformable ones, and the reasoner picks per subgoal from the current scene and history, and can pick differently during recovery. Adding a skill means adding a tool and a description, and nothing is retrained.

Section 5.3 isolates this contribution in two ways. Running the loop with a single skill measures how much of the gain comes from using one skill better. Comparing against an oracle that selects the better raw skill for each task, without any loop, measures how much comes from having a second skill at all. Together they separate the interface's contribution from the skill set's.

4 EXPERIMENTAL SETUP

Section 3 named a comparison for each. This section describes where each is run and what the evaluation is built to support.

Evaluation settings. We evaluate `Pigey` on a real robot and in simulation. On the real robot, we test how verification and recovery improve task execution and how adding a complementary skill changes performance (Sections 5.2 and 5.3). We use the DROID tabletop setup (Khazatsky et al., 2024) with 30 tasks and five trials per task, for 150 trials per system. Simulation allows us to compare action interfaces and evaluate a wider range of reasoners (Section 5.4). We use LIBERO-PRO (Zhou et al., 2025), which introduces perturbations to the manipulation tasks in LIBERO (Liu et al., 2023) and provides comparisons with published baselines, including CaP-Agent0 (Fu et al., 2026). The agent uses the frozen $\pi_{0.5}$-LIBERO checkpoint and the same interaction loop, with tools adapted to simulation (Appendix H.1). Unless stated otherwise, we use Claude Opus 5 with extended thinking as the reasoner and Gemini Robotics-ER 2 as the perceiver.

Real-robot tasks. The 30 real-robot tasks test how the agent uses existing pick-and-place skills. Four tasks involve simple pick-and-place and provide a reference for basic execution. The remaining 26 require the agent to select objects, order actions, use information from earlier observations, or respond to changes and failures during execution. They cover world knowledge, conditional logic, multi-step and spatial reasoning, obstacle and safety reasoning, error recovery, and long-horizon memory (Appendix B). The tasks do not require new motor primitives, but the available skills can still fail. This allows us to evaluate both task-level decisions and recovery from execution errors.

Simulation benchmark. We focus on the *swap* and *task* perturbations in LIBERO-PRO, where the frozen policy reaches only 10.8% success. We follow the six evaluation cells in CaP-Agent0's published protocol and add the same two perturbations of the *Long* suite, giving eight cells in total. For completeness, we also evaluate the full benchmark: 20 cells covering four suites and five pertur-

bations. On the remaining twelve cells, the frozen policy reaches 81.5% and `Pigey` reaches 83.3%. Full results are reported in Appendix G (Table 16).

Baselines and ablations. All comparisons hold the frozen skills fixed and vary only the interface, each testing the alternatives named in Section 3: program generation and direct end-effector control for the action interface (Section 5.4), matched on model, perception, and budget; retry and text-only-feedback baselines and a verification-and-recovery ablation; and single-skill agents and a per-task skill-selection oracle for the added skill (both in Table 3). In simulation we additionally compare against published policies and CaP-Agent0.

Reasoner configurations. With the interface and frozen policy fixed, we vary the reasoner across the Claude, Gemini, and Qwen families, thirteen configurations in all (Appendix G). This tests whether the benefits extend across model families and sizes rather than depending on a single reasoner. These runs keep Gemini Robotics-ER as the grounder and change only the reasoner. A separate configuration replaces the grounder with Qwen3.8-27B as well, so that reasoner, perceiver, and policy all run on open weights.

Scoring. Every trial is scored binary success. On the real robot a human evaluator judges the final scene against the task specification, and calling DONE before the task is complete counts as a failure. In simulation the LIBERO predicate decides, following CaP-Agent0's protocol. We report task-clustered bootstrap confidence intervals for the key real-robot comparisons (Table 3). Hardware, software, and scoring details are in Appendices I and K.

5 RESULTS

The real-robot experiments separate two sources of improvement: using an existing motor skill more effectively and adding a complementary skill. The same frozen $\pi_{0.5}$ policy succeeds in 16.7% of trials when called directly and 84.0% through `Pigey`. Adding TAMP raises success to 97.3%. We first examine performance across the task categories, then evaluate verification and recovery and access to a second skill. Finally, simulation experiments compare action interfaces and test performance across VLM reasoners.

5.1 REAL-ROBOT PERFORMANCE ACROSS CAPABILITY PROBES

Table 1 shows that the largest gains occur beyond the simple pick-and-place controls. Across the other 26 tasks, which probe task-level decisions and recovery, the raw VLA averages 4.6% and the full `Pigey` system reaches 96.9%. On the four pick-and-place controls, the raw VLA reaches 95.0%, TiPToP 80.0%, and `Pigey` 100%. Thus, the full system improves performance on the demanding probes without sacrificing basic manipulation. This comparison includes both the interface and access to a second skill, and the comparisons below separate their contributions.

The failure analysis in Table 2 helps characterize the remaining errors. Grounding and reasoning/-planning account for 124 of the raw VLA's 125 failed trials and 70 of TiPToP's 77. The full system has four failed trials: two attributed to grasp execution and two to verifier false-success. Because these counts describe unsuccessful trials and exclude errors the loop recovered from, none of the full system's terminal failures trace to grounding or reasoning/planning.

Figure 3 illustrates how the agent uses observations and history in three representative rollouts. In the occlusion task (top), the reasoner considers that the doll may be hidden, removes the covering container, re-perceives, and then picks the doll. The sequence uses existing manipulation skills but requires an intermediate action to make the target accessible.

In the stacking task (middle), the instruction requires all four containers to be included without specifying their order. The reasoner orders them by detected size and nests them largest first. This allows each step to move one empty container rather than requiring a later step to lift a partial stack. It issues one PICK/DROPABOVE pair at a time and checks the result before proceeding.

In the memory task (bottom), the reasoner records the dolls' positions before its view is blocked and the scene is shuffled. It then uses the interaction history to restore the arrangement, postponing a placement when its destination is occupied. These rollouts illustrate the decisions enabled by the interface, and the controlled comparisons below assess their contribution to performance. Per-task

Category	Tasks	$\pi_{0.5}$-DROID	TiPToP	Pigey (ours)
Simple pick-place	4	95	80	100
World knowledge	4	0	90	100
Conditional logic	4	0	95	100
Multi-step reasoning	4	0	25	100
Spatial reasoning	4	20	75	100
Obstacle/Safety reasoning	4	0	0	90
Error recovery	4	10	0	90
Long-horizon memory	2	0	0	100
Overall	30	16.7	48.7	**97.3**

Table 1: Real-robot success rate (%) by capability probe. Each task has five trials. The Pigey column reports the full system with both skills. All motor skills are frozen. The VLA baseline and Pigey use identical $\pi_{0.5}$-DROID weights.

Failure mode	$\pi_{0.5}$	TiPToP	Pigey
Grounding	86	5	0
Reasoning/planning	38	65	0
Grasp	1	7	2
Verifier false-success	0	0	2
Total failures	125	77	4

Table 2: Failure modes in unsuccessful trials (counts, of 150 per system), by first unrecovered error. Errors recovered during execution are excluded.

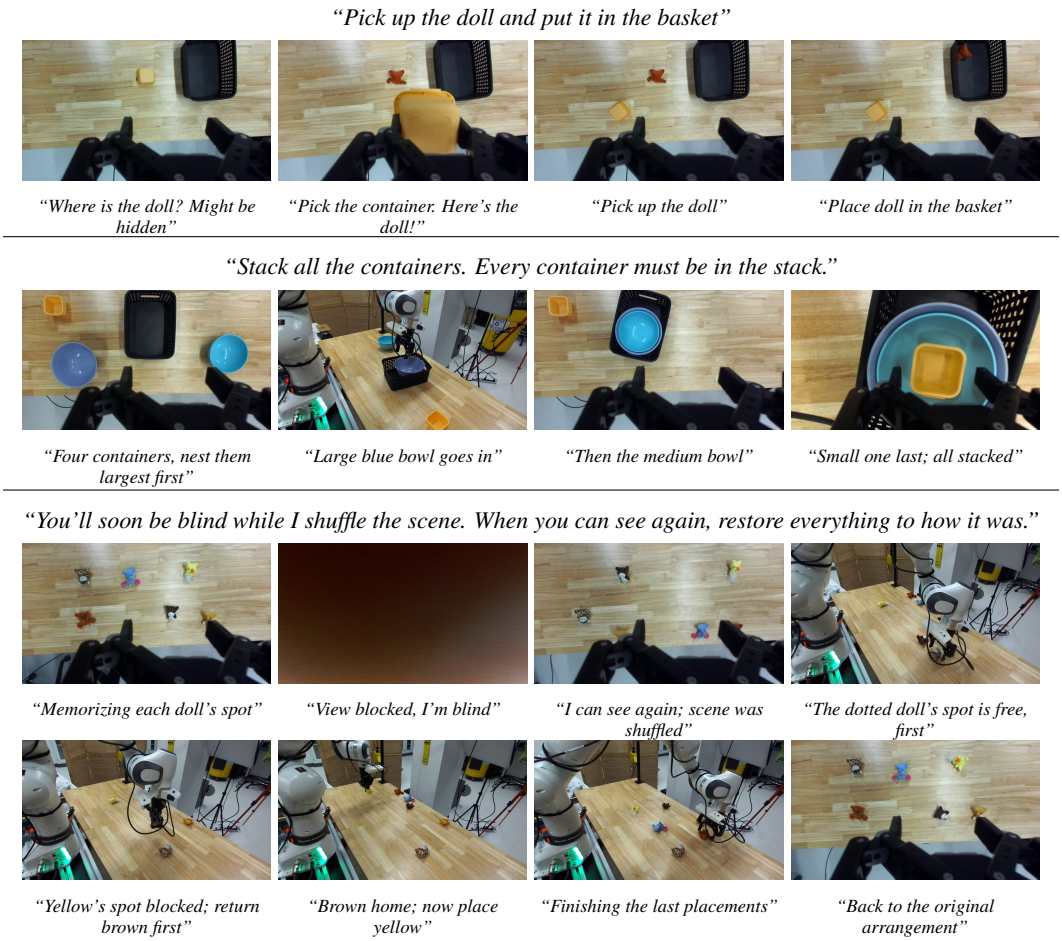


Figure 3: Real-robot rollouts, with Pigey's decisions below each frame: uncovering a hidden object before picking (top), multi-step stacking of every container in size order (middle), and long-horizon memory (bottom two rows: memorizing a scene, being blinded while it is shuffled, then restoring it one object at a time, ordering placements when a spot is blocked). Further rollouts involving world knowledge, safety, recovery, and conditional logic are provided in Appendix F.2.

results and additional failure analyses are in Appendix F. Further rollouts and videos are provided in Appendix F.2 and the supplementary material.

5.2 Using existing skills more effectively: verification and recovery

Table 3 examines whether the improvements can be explained by additional attempts, feedback alone, or the combined verification-and-recovery procedure. We first compare systems using the same single motor skill, then examine feedback baselines and an ablation of the full system.

Method	Tests	Success (%)
$\pi_{0.5}$ raw (frozen VLA)	–	16.7
TiPToP open-loop (frozen planner)	–	48.7
Oracle skill selection (per-task best raw skill, no loop)	skill selection alone	52.0
$\pi_{0.5}$ + retries (3/subgoal)	more attempts	26.7
TiPToP-CL (re-perceive, re-plan, retry)	closing the loop	63.3
IM-style planner (text-only feedback, same skills)	closing the loop, text feedback	51.3
`Pigey`−VR (verification/recovery removed)	verification and recovery	69.3
`Pigey`, planner skill only	interface, one skill	83.3
`Pigey`, VLA skill only	interface, one skill	84.0
`Pigey` (full)	second skill	**97.3**

Table 3: Real-robot baselines and ablations. Comparisons examine additional attempts, execution feedback, verification and recovery, and access to complementary skills. Retry baselines allow three retries per subgoal. `Pigey` allows one retry, with skill switching available in the full system. Reported task-clustered bootstrap 95% confidence intervals for key differences appear in the text.

The interface improves the use of either skill alone. With only the VLA skill, `Pigey` reaches 84.0%, compared with 16.7% for the raw VLA and 26.7% when the raw VLA receives three retries per subgoal. The single-skill agent cannot switch to TAMP, so its improvement does not depend on access to a second skill. Similarly, TAMP reaches 48.7% alone and 83.3% inside `Pigey`. These comparisons show that additional attempts alone do not account for the observed gains.

Feedback baselines recover part of the gap. TiPToP with re-perception, re-planning, and retries reaches 63.3%. An Inner-Monologue-style planner using the same VLM and skills with text-only feedback reaches 51.3%. These baselines remain below the full system. The verification-and-recovery ablation provides a more targeted comparison within `Pigey`.

Removing verification and recovery produces a substantial drop. Removing both from the full system lowers success from 97.3% to 69.3%, a drop of approximately 28 percentage points. Verification and recovery form a single loop: the agent verifies each outcome and acts on the failures it detects. Measured on the two-skill system, this drop also includes recovery through skill switching.

The reported task-clustered bootstrap estimates over the 30 tasks support these differences. The estimated gaps and 95% confidence intervals, in percentage points, are: full system versus raw VLA, 80.6 [68.0, 91.3]; full system versus −VR, 28.1 [18.0, 39.3]; and VLA-only `Pigey` versus VLA with retries, 57.3 [40.7, 74.0]. All three intervals exclude zero.

5.3 Adding a complementary skill

The single-skill agents establish how much can be gained without expanding the available motor capabilities. Adding TAMP to the VLA-only agent raises success from 84.0% to 97.3%. Conversely, adding the VLA to the TAMP-only agent raises success from 83.3% to 97.3%.

An oracle that selects the better raw skill for each task, without the agent loop, reaches 52.0%. Thus, selecting between the two raw skills at the task level does not reproduce the performance of either single-skill agent or the full system. The results support both contributions: more effective use of existing skills through the interface, and an additional benefit from access to a complementary skill.

Skill use is asymmetric. Across 150 real-robot trials, the full system issued 445 TAMP calls and 33 VLA rollouts. VLA rollouts therefore account for 6.9% of these motor-skill invocations, yet removing the VLA lowers success by 14 percentage points. The rollouts show the reasoner using TAMP for routine geometric manipulation and the VLA for deformable objects, clutter, and recovery. These counts describe routing frequency, not computation or execution time, since the two call types cover different amounts of motion. Costs and durations are in Appendix L.

5.4 Action interfaces and generality in simulation

Simulation serves two purposes: comparing action interfaces under controlled conditions and testing the agent across reasoners. On the eight LIBERO-PRO cells from CaP-Agent0's protocol (Section 4), `Pigey` with Opus 5 reaches 55.0%, against 14.6% for CaP-Agent0 and 10.8% for the frozen

Table 4: LIBERO-PRO success rate (%) on the eight evaluation cells (suite × {swap, task}). "+t" denotes extended thinking. *Prior VLAs* are OpenVLA, π_0, MolmoAct, NORA, and x-VLA (Kim et al., 2024; Black et al., 2025; Lee et al., 2025; Hung et al., 2025; Zheng et al., 2026). Full details are provided in Appendix K.

Method	Obj. swap	Obj. task	Sp. swap	Sp. task	Goal swap	Goal task	Long swap	Long task	Mean
Prior VLAs				all cells ≤ 10					0–3.8
$\pi_{0.5}$-LIBERO (Physical Intelligence, 2025)	17	1	20	1	38	0	8	1	10.8
CaP-Agent0 (Fu et al., 2026)	22	18	12	14	26	17	5.2	2.4	14.6
Direct frontier control (Opus 5 +t)	50	50	34	58	22	34	8	0	32.0
Program generation (Qwen3.8 27B, open grounder)	40	46	62	70	38	36	18	16	40.8
Program generation (Opus 5 +t)	40	36	70	78	50	44	22	30	46.2
`Pigey` (ours, Qwen3.5 9B)	40	42	66	68	42	34	14	16	40.3
`Pigey` (ours, Qwen3.5 27B)	42	50	60	78	52	36	18	30	45.8
`Pigey` (ours, Qwen3.8 27B)	50	38	74	90	52	58	10	24	49.5
`Pigey` (ours, Qwen3.8 27B, open grounder)	54	38	72	88	50	42	18	22	48.0
`Pigey` (ours, Opus 4.7 +t)	48	58	72	82	36	52	18	34	50.0
`Pigey` (ours, Opus 5 +t)	56	50	68	96	58	66	28	18	55.0

$\pi_{0.5}$ policy (Table 4). The direct-control and program-generation experiments below examine the action interface with the reasoner held fixed.

On the full 20-cell grid, `Pigey` reaches 72.1% against 53.0% for the frozen policy (Table 16). On the remaining twelve cells the two are close, 81.5% and 83.3%.

Skill-level actions versus direct control. We replace `Pigey`'s toolset with end-effector commands, keeping the model, perception, thinking, and budget fixed. Direct control reaches 32.0% against 55.0% through `Pigey`, isolating the value of skill-level actions over raw control.

One action per turn versus program generation. We compare one tool call per model turn with generating a program over the same tools, retaining the model, perception, frozen policy, and action budget. With Qwen3.8-27B and the open grounder, program generation reaches 40.8%, compared with 48.0% for one action per turn. With Opus 5, the corresponding results are 46.2% and 55.0%. Both comparisons favor returning control to the model after each call. Generated programs can use feedback within their own control flow, but the distinction is that the model revisits its decisions after the program returns rather than after each tool call.

Generality across VLMs. With the interface and motor policy fixed, the reasoners in Table 4 reach 40.3–55.0%, all above the 10.8% frozen-policy baseline: Qwen3.5-9B 40.3%, Qwen3.5-27B 45.8%, Qwen3.8-27B 49.5%, and Opus 4.7 and Opus 5 50.0% and 55.0%. With an open-weight grounder, Qwen3.8-27B still reaches 48.0%. The full sweep, including Gemini, is in Table 15 (Appendix G). The gains across families and sizes show the benefit is not tied to a single reasoner or grounder, though the reasoner still matters.

6 Discussion and Limitations

We studied how an agent–robot interface connects a VLM reasoner to existing motor skills. On our real-robot probes, the same frozen $\pi_{0.5}$ policy reaches 16.7% success when called directly and 84.0% through `Pigey`. Adding a complementary frozen TAMP skill raises success to 97.3%. These comparisons distinguish two sources of improvement: using an existing skill more effectively and expanding the available skill set. Removing verification and recovery produces the largest ablation drop, while simulation favors returning control to the reasoner after each call over generating a program. Substantial task-level gains are thus possible without updating motor-policy weights.

Our results suggest a sequencing rule. Task-level reasoning, memory, and verification can be supplied at inference time by a VLM, so before spending limited robot data to teach a policy to reason, it is worth first measuring how much an inference-time agent recovers from the frozen policy.

Limitations. `Pigey` depends on its skills' execution capabilities, which it composes but cannot guarantee. Extending it to other action classes (e.g., opening drawers, pouring) may need more tools and feedback. Verification is imperfect, especially for partial or deformable grasps, and wrong judgments can propagate. The loop also adds inference latency and cost. Finally, our real-robot evaluation is tabletop manipulation on a single embodiment, and the task categories are capability probes with few tasks each, not precise per-capability estimates.

AI USE STATEMENT

Generative AI was used as an assistive tool in this project. Frozen vision-language models are themselves the object of study (Section 3). Separately, we used language models for coding assistance and debugging while building the experimental and evaluation pipeline, and for editing the manuscript, revising text, improving clarity, and preparing figures and LaTeX. We also used them to surface potentially relevant related work, but the authors read and verified every reference cited in this paper. All research design, experiments, analysis, and claims are the authors' own. The authors reviewed all AI-assisted material and take responsibility for it, and AI-generated code was validated through the experiments described in the paper. Generative AI systems are not listed as authors.

ETHICS STATEMENT

This work studies agents that drive a physical robot by calling frozen motor skills. The experiments use a single tabletop robot arm and a public simulation benchmark, and involve no human subjects or personal data. Because the system acts in the physical world, its actions can have physical consequences: the verification-and-recovery loop reduces execution errors but is imperfect, and behavior may be unreliable outside the task types and settings evaluated here. Some tasks involve everyday safety reasoning, such as moving unsafe objects out of reach, evaluated only in a controlled laboratory. We therefore emphasize held-out evaluation, explicit assumptions about the deployment setting, and human oversight of any physical deployment. The approach may lower the effort of building capable robot agents from existing skills, and we see no application-specific dual-use concerns beyond those common to robotic manipulation.

REPRODUCIBILITY STATEMENT

We provide the definitions of the 30 real-robot tasks and the scoring rubric, the full agent prompt and tool schemas, the interaction loop, the hardware and software stack, the LIBERO-PRO protocol and its simulation-specific tools, the reasoner and perceiver models with their inference settings, and code, in the appendices and the anonymized supplementary material. All motor components are frozen and are not trained. The real-robot protocol uses 30 tasks with five trials each, and the simulation protocol follows CaP-Agent0's published cells with 100 episodes per cell (ten seeds). We describe the task-clustered bootstrap procedure used for the reported confidence intervals. Additional ablations report the effect of verification and recovery, single-skill agents, a per-task skill-selection oracle, one action per turn versus program generation, direct end-effector control, and a sweep over thirteen reasoner configurations spanning frontier and open-weight models.

REFERENCES

M. Ahn et al. Do as i can, not as i say: Grounding language in robotic affordances. In *CoRL*, 2022.

Johan Bjorck et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv:2503.14734*, 2025.

Kevin Black et al. $\pi 0$: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems (RSS)*, 2025.

W. Chen et al. Steerable vision-language-action policies for embodied reasoning and hierarchical control. *arXiv:2602.13193*, 2026a.

Yanzhe Chen, Zechen Bai, Zhijun Cao, Wenzheng Zeng, Kevin Qinghong Lin, Yiqi Lin, Guoqiang Liang, Kevin Yuchen Ma, Qiming Huang, and Mike Zheng Shou. Show-harness: Just a VLM agent can play robots. *arXiv preprint arXiv:2609.10522*, 2026b.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, Guanya Shi, Jiajun Wu, Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi Fan. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026.

Hongbo Gao, Zeyu Ni, Xin Wen, Siyu Xu, and Ruifeng Li. AGM: Achievement-grounded memory for closed-loop agents with frozen VLA policies. *arXiv preprint arXiv:2608.29537*, 2026.

Google DeepMind. Gemini robotics: Bringing ai to the physical world. Technical report, 2025.

Jiaheng Hu, Mohit Shridhar, Caden Lu, Dhruv Shah, Hao-Tien Lewis Chiang, Jie Tan, and Annie Xie. What matters in orchestrating robot policies: A systematic study of hierarchical VLA agents. *arXiv preprint arXiv:2606.10267*, 2026.

W. Huang et al. Inner monologue: Embodied reasoning through planning with language models. In *CoRL*, 2022.

W. Huang et al. VoxPoser: Composable 3d value maps for robotic manipulation with language models. In *CoRL*, 2023.

Chia-Yu Hung, Qi Sun, Pengfei Hong, Amir Zadeh, Chuan Li, U Tan, Navonil Majumder, Soujanya Poria, et al. NORA: A small open-sourced generalist vision language action model for embodied tasks. *arXiv preprint arXiv:2504.19854*, 2025.

J. Kang et al. RoboMonkey: Scaling test-time sampling and verification for vision-language-action models. *arXiv:2506.17811*, 2025.

A. Khazatsky et al. DROID: A large-scale in-the-wild robot manipulation dataset. In *Robotics: Science and Systems (RSS)*, 2024.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. OpenVLA: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Jason Lee, Jiafei Duan, Haoquan Fang, Yuquan Deng, Shuo Liu, Boyang Li, Bohan Fang, Jieyu Zhang, Yi Ru Wang, Sangho Lee, et al. MolmoAct: Action reasoning models that can reason in space. *arXiv preprint arXiv:2508.07917*, 2025.

Anthony Liang, Yigit Korkmaz, Jiahui Zhang, Minyoung Hwang, Abrar Anwar, Sidhant Kaushik, Aditya Shah, Alex S Huang, Luke Zettlemoyer, Dieter Fox, et al. Robometer: Scaling general-purpose robotic reward models via trajectory comparisons. *arXiv preprint arXiv:2603.02115*, 2026.

J. Liang et al. Code as policies: Language model programs for embodied control. In *ICRA*, 2023.

B. Liu et al. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *NeurIPS Datasets and Benchmarks*, 2023.

Yecheng Jason Ma, Joey Hejna, Chuyuan Fu, Dhruv Shah, Jacky Liang, Zhuo Xu, Sean Kirmani, Peng Xu, Danny Driess, Ted Xiao, et al. Vision language models are in-context value learners. In *International Conference on Learning Representations*, volume 2025, pp. 33984–34009, 2025.

J. Pan et al. MemER: Scaling up memory for robot control via episodic-keyframe selection. `https://jen-pan.github.io/memer/`, 2026.

Physical Intelligence. π0.5: A vision-language-action model with open-world generalization. *arXiv:2504.16054*, 2025.

W. Shen et al. TiPToP: A modular open-vocabulary planning system for robotic manipulation. *arXiv:2603.09971*, 2026.

Junyao Shi, Rujia Yang, Kaitian Chao, Selina Bingqing Wan, Yifei Shao, Jiahui Lei, Jianing Qian, Long Le, Pratik Chaudhari, Kostas Daniilidis, Chuan Wen, and Dinesh Jayaraman. Maestro: Orchestrating robotics modules with vision-language models for zero-shot generalist robots. *arXiv preprint arXiv:2511.00917*, 2025a.

L. Shi et al. Hi robot: Teaching robots to listen and think harder. *arXiv:2502.19417*, 2025b.

I. Singh et al. ProgPrompt: Generating situated robot task plans using large language models. In *ICRA*, 2023.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, 2024.

S. Yao et al. ReAct: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

M. Zawalski et al. Robotic control via embodied chain-of-thought reasoning. In *CoRL*, 2024.

Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, Zhengru Fang, Yi Nie, Changxu Wei, Yu Wang, Wenbo Ding, and Chao Yu. Harness VLA: Steering frozen VLAs into reliable manipulation primitives via memory-guided agents. *arXiv preprint arXiv:2607.08448*, 2026.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, et al. X-VLA: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *International Conference on Learning Representations*, 2026.

X. Zhou et al. LIBERO-PRO: Towards comprehensive evaluation of robotic manipulation under perturbations. *arXiv:2510.03827*, 2025.

A Full Problem Setup

We study general-purpose tabletop manipulation from natural-language instructions. The agent observes the scene through cameras, receives a language instruction, and must execute physical actions until the instruction is satisfied. The instructions extend beyond direct pick-and-place: they require world knowledge, conditional logic, multi-step reasoning, spatial and obstacle reasoning, recovery from failed grasps, and long-horizon memory (Appendix B).

Let $\mathcal{I}$ be the space of natural-language instructions and $\mathcal{O}$ the space of observations available to the agent. Let $\mathcal{A}_{\text{motor}}$ denote the continuous motor-action space of the robot. A VLA policy is a function

$$\pi : \mathcal{O} \times \mathcal{S} \to \mathcal{A}_{\text{motor}},$$

where $\mathcal{S} \subset \mathcal{I}$ is the subspace of short, concrete, visually grounded subgoals on which the policy is reliable. A user instruction $I \in \mathcal{I}$ is solved if the final world state satisfies a task-specific success predicate $\mathbb{1}_I(\text{scene}) = 1$.

Direct VLA execution conditions the motor policy on the full instruction, $\pi(o, I)$, which can fail when I requires intermediate reasoning. `Pigey` instead has the reasoner emit a sequence of tool calls, each naming a subgoal $s \in \mathcal{S}$ or a TAMP target, and reads the post-action observation before choosing the next (Algorithm 1). The reasoner never sees the success predicate $\mathbb{1}_I$: its verification is made from sensors and remains fallible, and the Done branch of the algorithm denotes the evaluator's independent judgment of the final state (Appendix K).

B Task Suite

We evaluate across the capability probes in Table 5; per-task instructions, scene contents, and success predicates follow, each row showing a thumbnail of the task's initial state. Predicates use $\text{on}(x, r)$ (object x rests on receptacle r), $\text{in}(x, c)$ (object x inside container c), $\text{grasped}(x)$ (stable grasp achieved), $\text{color}(\cdot)$ and $\text{size}(\cdot)$ for attributes, and $\text{count}(\cdot)$ for cardinality.

Capability probe	# Tasks
Pick-and-place (control)	4
World knowledge	4
Conditional logic	4
Multi-step reasoning	4
Spatial reasoning	4
Obstacle reasoning	4
Error recovery	4
Long-horizon memory	2
Total	30

Table 5: Task suite composition.

B.1 Pick-and-place (control)

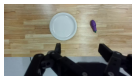
Simple single-step control condition: a named object to a named receptacle.

Table 6: Pick-and-place control tasks ($N = 4$).

ID	Init.	Instruction (verbatim)	Scene contents	Success predicate
PP1		*"Pick up the doll and put it in the bowl."*	doll, bowl	$\text{in}(\text{doll}, \text{bowl})$

continued on next page

Table 6: Pick-and-place control tasks (continued).

ID	Init.	Instruction (verbatim)	Scene contents	Success predicate
PP2		*"Pick up the eggplant and put it on the plate."*	eggplant, plate	$\mathrm{on}(\text{eggplant}, \text{plate})$
PP3		*"Pick up the cup and put it in the basket."*	cup, basket	$\mathrm{in}(\text{cup}, \text{basket})$
PP4		*"Pick up the mouse and put it on the plate."*	mouse, plate	$\mathrm{on}(\text{mouse}, \text{plate})$

B.2 World knowledge

Probes whether the policy resolves a referring expression using facts not stated in the scene. The four prompts share one scene; the target differs.

Table 7: World-knowledge tasks ($N = 4$).

ID	Init.	Instruction (verbatim)	Scene contents	Success predicate
WK1		*"Pick up something you would put in ratatouille and put it on the plate."*	doll, eggplant, glue, cup, mouse, plate	$\mathrm{on}(\text{eggplant}, \text{plate})$
WK2		*"Pick up something a child would sleep with and put it on the plate."*	doll, eggplant, glue, cup, mouse, plate	$\mathrm{on}(\text{doll}, \text{plate})$
WK3		*"Pick up something you could fix a broken mug with and put it on the plate."*	doll, eggplant, glue, cup, mouse, plate	$\mathrm{on}(\text{glue}, \text{plate})$
WK4		*"Pick up something that controls the cursor on a screen and put it on the plate."*	doll, eggplant, glue, cup, mouse, plate	$\mathrm{on}(\text{mouse}, \text{plate})$

B.3 Conditional logic

Probes selection of a target by an attribute or relational criterion rather than a name.

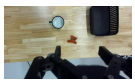
Table 8: Conditional-logic tasks ($N = 4$).

ID	Init.	Instruction (verbatim)	Scene contents	Success predicate
CL1		*"Pick up the smallest object on the table and put it on the plate."*	big bowl, medium bowl, tape, mouse, doll, plate	$\mathrm{on}(x, \text{plate})$ where $x = \arg\min \text{size}$
CL2		*"Pick up the biggest object on the table and put it on the plate."*	bowl, medium bowl, tape, mouse, doll, plate	$\mathrm{on}(x, \text{plate})$ where $x = \arg\max \text{size}$
CL3		*"Pick up the object that doesn't belong with the others and put it on the plate."*	dolls, mouse, plate	$\mathrm{on}(x, \text{plate})$ where x is the odd one out
CL4		*"Pick up a red or a blue object and put it on the plate."*	dolls of mixed colors incl. ≥ 1 blue, plate	$\mathrm{on}(x, \text{plate})$ with $\mathrm{color}(x) \in \{\text{red}, \text{blue}\}$

B.4 MULTI-STEP REASONING

Probes composition of subgoals, counting, and ordered execution.

Table 9: Multi-step-reasoning tasks ($N = 4$).

ID	Init.	Instruction (verbatim)	Scene contents	Success predicate
MS1		*"Put exactly 2 objects in the cup."*	≥ 3 small objects (dolls), glue, tape, mouse, cup	$\mathrm{count}(\{x : \mathrm{in}(x, \text{cup})\}) = 2$
MS2		*"If the cup is empty, put the doll in it; otherwise put it in the basket."*	doll, cup (empty or filled), basket	if cup initially empty then $\mathrm{in}(\text{doll}, \text{cup})$, else $\mathrm{in}(\text{doll}, \text{basket})$
MS3		*"Stack all the containers. Every container must be in the stack."*	multiple stackable containers	every container belongs to one connected stack

continued on next page

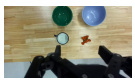
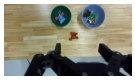
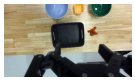
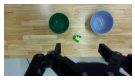
Table 9: Multi-step-reasoning tasks (continued).

ID	Init.	Instruction (verbatim)	Scene contents	Success predicate
MS4		*"First pick up the cup and put it on the plate. Then pick up the cup and put it in the basket."*	cup, plate, basket	ordered: $\mathrm{on}(\text{cup}, \text{plate})$ reached, then $\mathrm{in}(\text{cup}, \text{basket})$; final state $\mathrm{in}(\text{cup}, \text{basket})$

B.5 Spatial reasoning

Probes selection of a container by spatial / relational property.

Table 10: Spatial-reasoning tasks ($N = 4$).

ID	Init.	Instruction (verbatim)	Scene contents	Success predicate
SR1		*"Put the doll in the bowl next to the cup."*	doll, cup, two bowls at distinct positions	$\mathrm{in}(\text{doll}, b)$ where b is adjacent to the cup
SR2		*"Put the doll in the bowl that contains the most items."*	doll, bowls with differing item counts	$\mathrm{in}(\text{doll}, b^*)$, $b^* = \arg\max_b \mathrm{count}(\text{items in } b)$
SR3		*"Put the doll in the smallest container it still fits in."*	doll, containers of graded sizes	$\mathrm{in}(\text{doll}, b^*)$, smallest b with $\mathrm{fits}(\text{doll}, b)$
SR4		*"Put the doll in the bowl that matches its color."*	doll, bowls of varied colors incl. doll's color	$\mathrm{in}(\text{doll}, b^*)$, $\mathrm{color}(b^*) = \mathrm{color}(\text{doll})$

B.6 Obstacle reasoning

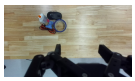
Probes handling of an obstacle that blocks the naive execution path. The italicized note in each scene is the obstacle.

Table 11: Obstacle-reasoning tasks ($N = 4$).

ID	Init.	Instruction (verbatim)	Scene contents (*obstacle*)	Success predicate
OB1		*"Pick up the doll and put it in the basket."*	doll, container, basket *(doll hidden under the container)*	$\mathrm{in}(\text{doll}, \text{basket}) \wedge \neg\,\mathrm{in}(\text{container}, \text{basket})$

continued on next page

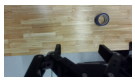
Table 11: Obstacle-reasoning tasks (continued).

ID	Init.	Instruction (verbatim)	Scene contents (*obstacle*)	Success predicate
OB2		*"A child is coming over, put the items they would want to play with in the basket."*	toys + basket *(basket contains dangerous items)*	child-appropriate items placed for the child; dangerous items not made accessible
OB3		*"Pick up the doll."*	doll *(surrounded by other objects)*	$\mathrm{grasped}(\mathrm{doll})$ without displacing neighbors
OB4		*"Put the doll in the cup."*	doll, cup *(cup already occupied)*	$\mathrm{in}(\mathrm{doll}, \mathrm{cup})$ after removing the occupant

B.7 Error recovery

Probes recovery from a perturbation introduced mid-rollout. The italicized note in each scene is the perturbation; thumbnails show the state before perturbation.

Table 12: Error-recovery tasks ($N = 4$).

ID	Init.	Instruction (verbatim)	Scene contents (*perturbation*)	Success predicate
ER1		*"Pick up the tape."*	tape *(tape relocated mid-reach)*	$\mathrm{grasped}(\mathrm{tape})$ at its new pose
ER2		*"Pick up a container."*	several containers *(the one being grasped is removed on contact)*	$\mathrm{grasped}(c')$ for another available container
ER3		*"Put the doll on an empty plate. If there is no empty plate, put it in an empty bowl."*	doll, bowls, plate *(an item is dropped onto the empty plate during placement)*	re-evaluated against final state: $\mathrm{on}(\mathrm{doll}, \mathrm{plate})$ if the plate is still empty, else $\mathrm{in}(\mathrm{doll}, b)$ for an empty bowl b
ER4		*"Put the doll in the green bowl."*	doll, green bowl *(green bowl displaced after release; arm must correct)*	final $\mathrm{in}(\mathrm{doll}, \mathrm{green\ bowl})$

B.8 Long-horizon memory

Probes retention of earlier instructions or state across a long rollout. The italicized note in each scene is the off-camera event the policy must bridge.

Table 13: Long-horizon-memory tasks ($N = 2$).

ID	Init.	Instruction (verbatim)	Scene contents (*event*)	Success predicate
LM1		*"You'll soon be blind while I shuffle the scene. When you see the scene again, restore everything to how it was at the start."*	several objects in a known starting arrangement *(view occluded while the objects are shuffled; a hands-free frame cues action)*	every object returned to its initial pose: $\mathrm{pose}(x) \approx \mathrm{pose}_0(x)$ for all x
LM2		*"I'll demo the task with my hands, one object move at a time. Watch each move. After I reset the scene and my hands leave the frame, replay exactly what I demonstrated."*	objects on the table *(human demonstrates a sequence of single-object moves, then resets the scene; a hands-free frame cues replay)*	the executed move sequence reproduces the demonstrated one in order; final configuration matches the demonstration's end state

C $\pi_{0.5}$ INFERENCE LOOP

VLAROLLOUT runs the standard $\pi_{0.5}$ inference loop with the reasoner's subgoal s. The policy emits motion in chunks, not one action per call: on the real robot (`pi05_droid`) each call returns 15 actions and the policy is re-queried after H=8 of them; in simulation (`pi05_libero`) each call returns 10 actions and H=5. Images are resized to the policy's expected input size; the gripper bit is binarized and actions are clipped to the valid range before being sent to the controller. On the real robot we pace the loop at the robot's native control rate.

Algorithm 2: Server-side VLAROLLOUT(s, K, $H = 8$)

chunk $\leftarrow \emptyset$, $i \leftarrow 0$;
for $t = 1, \ldots, K$ **do**
  $\mathbf{I}^{ext}, \mathbf{I}^{wrist}, q, g \leftarrow$ `env.get_observation()`;
  resize $\mathbf{I}^{ext}$ and $\mathbf{I}^{wrist}$ to policy input size;
  if $i = 0$ ***or*** $i \geq H$ **then**
    chunk $\leftarrow \pi_{0.5}$`.infer(`$\mathbf{I}^{ext}, \mathbf{I}^{wrist}, q, g, s$`)` ; `// 15-action chunk`
    $i \leftarrow 0$;
  $a \leftarrow$ chunk$[i]$;
  binarize gripper bit and clip action;
  `env.step(`a`)`; record frame; $i \leftarrow i + 1$;
  pace to robot control rate on real hardware;
return *post-rollout observation*;

Unless otherwise specified, one rollout is capped at K=600 control steps on the real robot and K=500 in simulation.

D Observation Channel and Failure Annotations

After every tool call the reasoner receives

$$o_t = (I_t^{\text{wrist}},\ L_t,\ g_t,\ e_t),$$

where I_t^{wrist} is the gripper-mounted RGB view, L_t is the current set of detected object labels, g_t is the binary IS_GRASPED flag from the gripper-width sensor, and e_t is a typed error if the call failed (e.g. `no_grasp_found`, `motion_planning_failed`). An explicit PERCEIVE additionally returns the third-person view and the end-effector pose.

The flags and errors are not learned. They are deterministic predicates computed from controller state and the rollout log, and they expose execution failures in a form the reasoner can act on without inferring every low-level failure from pixels. None of them encodes task success; that judgment remains the reasoner's, made from I_t^{wrist}, L_t, and g_t (Appendix K).

E Skill Routing Guidance

The reasoner chooses between the TAMP and VLA skills using a small set of verify-and-escalate rules stated in the system prompt. They are guidance, not a hard-coded router: the reasoner applies them using the per-call feedback of Appendix D.

1. **Perceive before acting.** The reasoner always calls PERCEIVE first to obtain the detected object labels, then maps the instruction's target, a named object or an abstract, multi-object goal (e.g. "childproof the table"), onto those concrete labels. Every PICK/DROPABOVE argument must be an exact label from the most recent PERCEIVE.
2. **Rigid grasp via TAMP, then verify.** For a rigid target the reasoner calls PICK and verifies the grasp from the wrist view and the gripper-width sensor (IS_GRASPED).
3. **Retry once, then escalate.** An unverified PICK is retried once; the retry re-perceives and re-plans at the object's current location. A second failure escalates to VLAROLLOUT.
4. **Deformables go straight to the VLA.** Cable-, cloth-, or rope-like targets bypass TAMP entirely, since the grasp predictor is not trained for deformables.
5. **Placement.** A held object is placed with DROPABOVE on the cached target location from a prior PERCEIVE/PICK.
6. **Bidirectional fallback.** If a VLAROLLOUT makes no progress, the reasoner may fall back to a TAMP PICK on a freshly perceived scene, and vice versa.
7. **Stop condition.** The reasoner calls DONE only after verifying the task predicate; for clearing/sorting tasks it stops once all in-scope objects have been moved, rather than reaching for borderline items.

The detected object labels come from an open-vocabulary detector and are regenerated on each PERCEIVE; the reasoner therefore re-reads the label set after every observation rather than reusing labels remembered from earlier turns.

F Additional Results

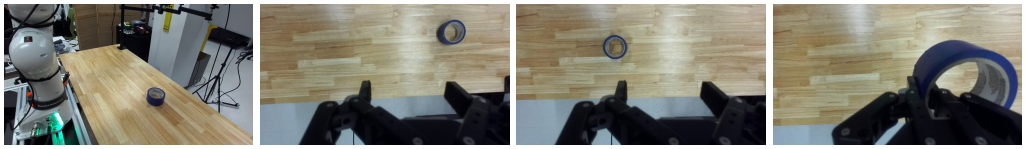
F.1 Per-task real-robot results

Table 14 reports per-task success on the real robot, mirroring the task suite of Appendix B. Each per-task cell is the number of successful rollouts out of five trials; category-mean and overall rows are success rates.

Table 14: Per-task real-robot success (per task: successes / 5 trials; summary rows: success rate). TT = TiPToP; TT-CL = closed-loop TiPToP (re-perceive, re-plan, up to three retries per subgoal); VLA+R = $\pi_{0.5}$ with three retries per subgoal and fresh perception per attempt; IM = same tools with text-only feedback; −VR = `Pigey` without verification and recovery; VLA-o. / TAMP-o. = the full `Pigey` loop with one backend. Best per row in bold.

ID	Task	VLA	TT	TT-CL	VLA+R	IM	−VR	VLA-o.	TAMP-o.	**`Pigey`**
Pick-and-place (control)										
PP1	doll → basket	**5/5**	4/5	5/5	5/5	5/5	4/5	5/5	5/5	**5/5**
PP2	eggplant → plate	4/5	4/5	5/5	5/5	5/5	4/5	5/5	5/5	**5/5**
PP3	cup → basket	**5/5**	4/5	5/5	5/5	5/5	4/5	5/5	5/5	**5/5**
PP4	mouse → plate	**5/5**	4/5	5/5	5/5	5/5	4/5	5/5	5/5	**5/5**
	category mean	95%	80%	**100%**	100%	100%	80%	100%	100%	100%
World knowledge										
WK1	ratatouille item (eggplant)	0/5	4/5	5/5	0/5	5/5	4/5	5/5	5/5	**5/5**
WK2	child sleeps with (doll)	0/5	**5/5**	5/5	0/5	5/5	5/5	5/5	5/5	**5/5**
WK3	fix a mug (glue)	0/5	4/5	5/5	0/5	5/5	4/5	5/5	5/5	**5/5**
WK4	controls cursor (mouse)	0/5	**5/5**	5/5	0/5	5/5	5/5	5/5	5/5	**5/5**
	category mean	0%	90%	**100%**	0%	100%	90%	100%	100%	100%
Conditional logic										
CL1	smallest object	0/5	4/5	5/5	0/5	1/5	4/5	5/5	5/5	**5/5**
CL2	biggest object	0/5	**5/5**	5/5	0/5	1/5	5/5	5/5	5/5	**5/5**
CL3	odd one out	0/5	**5/5**	5/5	0/5	5/5	5/5	5/5	5/5	**5/5**
CL4	red / blue object	0/5	**5/5**	5/5	0/5	0/5	5/5	1/5	5/5	**5/5**
	category mean	0%	95%	**100%**	0%	35%	95%	80%	100%	100%
Multi-step reasoning										
MS1	exactly 2 in cup	0/5	0/5	0/5	0/5	0/5	3/5	5/5	5/5	**5/5**
MS2	empty-cup conditional	0/5	5/5	5/5	0/5	2/5	4/5	5/5	5/5	**5/5**
MS3	stack all containers	0/5	0/5	0/5	0/5	1/5	2/5	1/5	5/5	**5/5**
MS4	cup → plate → basket	0/5	0/5	0/5	0/5	5/5	3/5	5/5	0/5	**5/5**
	category mean	0%	25%	25%	0%	40%	60%	80%	75%	**100%**
Spatial reasoning										
SR1	bowl next to cup	0/5	**5/5**	5/5	0/5	2/5	5/5	5/5	5/5	**5/5**
SR2	bowl with most items	2/5	**5/5**	5/5	2/5	2/5	4/5	5/5	5/5	**5/5**
SR3	smallest fitting container	0/5	0/5	0/5	0/5	1/5	5/5	1/5	5/5	**5/5**
SR4	color-matching bowl	2/5	**5/5**	5/5	2/5	5/5	5/5	5/5	5/5	**5/5**
	category mean	20%	75%	75%	20%	50%	95%	80%	**100%**	100%
Obstacle reasoning										
OB1	doll under container	0/5	0/5	0/5	0/5	0/5	4/5	5/5	5/5	**5/5**
OB2	child / dangerous basket	0/5	0/5	0/5	0/5	0/5	4/5	5/5	0/5	**5/5**
OB3	doll surrounded	0/5	0/5	0/5	0/5	0/5	3/5	0/5	0/5	**3/5**
OB4	occupied cup	0/5	0/5	0/5	0/5	0/5	4/5	5/5	0/5	**5/5**
	category mean	0%	0%	0%	0%	0%	75%	75%	25%	**90%**
Error recovery										
ER1	tape relocated mid-reach	2/5	0/5	5/5	5/5	5/5	0/5	5/5	5/5	**5/5**
ER2	container removed on contact	0/5	0/5	5/5	5/5	5/5	0/5	4/5	5/5	**4/5**
ER3	empty plate / fallback bowl	0/5	0/5	0/5	1/5	2/5	0/5	4/5	0/5	**4/5**
ER4	green bowl displaced	0/5	0/5	5/5	5/5	0/5	0/5	5/5	5/5	**5/5**
	category mean	10%	0%	75%	80%	60%	0%	**90%**	75%	90%
Long-horizon memory										
LM1	blind, then restore scene	0/5	0/5	0/5	0/5	0/5	2/5	0/5	5/5	**5/5**
LM2	replay demonstrated moves	0/5	0/5	0/5	0/5	0/5	3/5	5/5	5/5	**5/5**
	category mean	0%	0%	0%	0%	0%	50%	50%	**100%**	100%
Overall		16.7%	48.7%	63.3%	26.7%	51.3%	69.3%	84.0%	83.3%	**97.3%**

F.2 ADDITIONAL QUALITATIVE ROLLOUTS



"There's the tape" | *"Pick up the tape"* | *"Gripper's empty, the tape moved. Where is it now?"* | *"Found it. Pick up the tape"*

Task: "Pick up the tape." (tape is moved mid-execution)

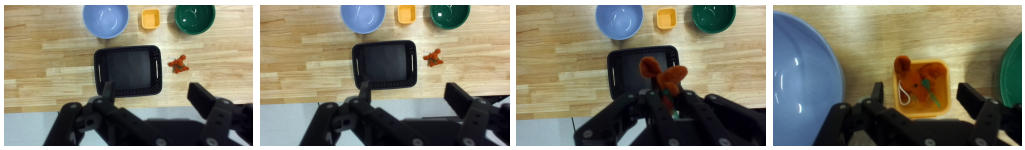
Figure 4: Key frames from a single execution where the target is displaced mid-task; per-frame reasoning appears below the images. The agent detects the empty gripper, re-perceives, and re-plans against the tape's new pose, a recovery an open-loop plan cannot make.



"The cup's occupied, there's a mouse on it" | *"I'll clear it first."* | *"Mouse set aside. The cup's empty now."* | *"Now I can put the doll in the cup."*

Task: "Put the doll in the cup." (cup is occupied with a mouse)

Figure 5: Key frames from a single execution of an obstacle task where the goal container is already occupied; per-frame reasoning appears below the images. The agent recognizes the cup is taken, clears the occupying object, and only then places the doll, a detour a policy that maps the instruction straight to a place action cannot make.



"Four containers, basket, large bowl, medium bowl, and small one." | *"The doll's small; the smallest one should still hold it. Let me try that."* | *"Picking up the doll."* | *"It fits, the small container was the right choice."*

Task: "Put the doll in the smallest container it still fits in."

Figure 6: Key frames from a single execution of a spatial-reasoning task where the doll must go in the smallest container it still fits in; per-frame reasoning appears below the images. The agent compares the available containers by size, reasons that the smallest one should still hold the doll, and confirms the fit, a relational size judgment beyond a policy that grounds only on object names.



"Only the toys are kid-safe." | *"Glue's on the plate, unsafe; clear it."* | *"Place the bunny on the plate."* | *"Place the elephant on the plate."*

Task: "A child is coming over, put the items they would want to play with on the plate."

Figure 7: Key frames from a single safety and world-knowledge execution. The agent selects the kid-safe toys, clears an unsafe item (glue) off the plate before placing, and puts the appropriate toys on the plate, decisions the motor policy is never asked to make.



"The plate's empty, that's where the doll goes." | *"Doll in hand, heading for the plate."* | *"Wait, the plate's not empty anymore."* | *"No empty plate, so the doll goes in the empty bowl."*

Task: "Put the doll on an empty plate. If there is no empty plate, put it in an empty bowl." (an item is dropped onto the plate mid-placement)

Figure 8: Key frames from a single execution of a conditional error-recovery task: the doll should go on an empty plate, with an empty bowl as the fallback. The plate starts empty, but an item is placed on it mid-execution; the agent detects that the plate is no longer empty, re-evaluates the condition, and falls back to the empty bowl, a re-plan an open-loop policy cannot make.



"Only banana and eggplant are vegetarian." | *"Put the eggplant in the bowl."* | *"Pick up the banana next."* | *"Both vegetarian items placed; animals left out."*

Task: "A vegetarian guest is coming for dinner. Set the bowl for them."

Figure 9: Key frames from a single world-knowledge execution. The agent selects only the plant-based food items (banana, eggplant) for a vegetarian guest and leaves the animal plush toys on the table, a categorization the motor policy is never asked to make.

Recorded sessions of the agent on representative tasks are provided as videos in the supplementary material (open `site/index.html`); the agent code and system prompts are in the `real/` and `sim/` folders.

F.3 First-failure mode distribution

For every failed episode we record the *first* error the policy could not recover from, and bucket it into one of four modes. Transient errors caught and retried by the closed loop are not counted as failures. Table 2 (Section 5.1) reports the distribution over all 150 trials for the raw policy, the raw planner, and `Pigey`.

G Additional Simulation Results

Full reasoner sweep (LIBERO-PRO). Table 15 gives the full sweep: thirteen reasoner configurations spanning the Claude, Gemini, and Qwen families, with and without extended thinking, each on the eight collapse cells at 100 episodes per cell (ten seeds). We sweep on the collapse cells because that is where the frozen policy fails and reasoner differences are measurable; on the remaining twelve cells the frozen policy already performs well (81.5%, against 83.3% for `Pigey`), reported in Table 16. Three things hold. First, every configuration clears both baselines, raw $\pi_{0.5}$ (10.8%) and CaP-Agent0 (14.6%), by a wide margin, so the gain is a property of the interface rather than of any single model. Second, the floor is high: open-weight Qwen models reach 40–46%, above the strongest closed models driven directly without the interface (32.0–33.5%), and replacing Gemini Robotics-ER with Qwen3.8-27B as grounder costs only 1.5 points (49.5 to 48.0), so the entire stack can run on open weights. Third, the model matters more than the thinking budget: within Opus 4.7, extended thinking adds 1.8 points, while swapping to Opus 5 adds 5.0 (50.0 to 55.0). The mapping is not strictly monotonic, however: Fable 5.1 trails Opus 5 (53.2 vs. 55.0) despite being stronger on text benchmarks. The reasoner sets the magnitude of the gain, not its sign.

Table 15: LIBERO-PRO success rate (%) on the eight collapse cells (suite × {swap, task}), 100 episodes per cell. Top rows are baselines without the interface; remaining rows use the same $\pi_{0.5}$-LIBERO motor policy inside `Pigey`. CaP-Agent0's six protocol cells are as published; its *Long* cells are from our run of the released system under the same protocol, and its mean is over all eight cells (published six-cell mean: 18.2). "+t" marks extended thinking.

Method / Reasoner	Obj. swap	Obj. task	Sp. swap	Sp. task	Goal swap	Goal task	Long swap	Long task	Mean
π_0 (Black et al., 2025)	0	0	0	0	0	0	0	0	0
$\pi_{0.5}$ (Physical Intelligence, 2025)	17	1	20	1	38	0	8	1	10.8
CaP-Agent0 (Fu et al., 2026)	22	18	12	14	26	17	5.2	2.4	14.6
`Pigey` (Ours)									
Qwen3.5 9B (open-weight)	40	42	66	68	42	34	14	16	40.3
Qwen3.5 27B (open-weight)	42	50	60	78	52	36	18	30	45.8
Qwen3.8 27B (open-weight)	50	38	74	90	52	58	10	24	49.5
Qwen3.8 27B, open grounder	54	38	72	88	50	42	18	22	48.0
Gemini Robotics-ER 2	52	48	74	84	44	44	20	16	47.8
Claude Opus 4.7	52	50	66	80	52	44	20	22	48.2
Gemini 3.5 Flash	40	50	72	92	44	48	28	26	50.0
Gemini 3.8 Flash	52	46	70	86	56	56	20	22	51.0
Claude Sonnet 5 +t	40	32	72	82	52	34	12	20	43.0
Claude Fable 5 +t	56	44	76	86	56	50	14	16	49.8
Claude Opus 4.7 +t	48	58	72	82	36	52	18	34	50.0
Claude Fable 5.1 +t	52	58	70	84	56	48	38	20	53.2
Claude Opus 5 +t	56	50	68	96	58	66	28	18	55.0

Table 16: Full LIBERO-PRO coverage (`Pigey`, 8,000-step horizon, 100 episodes/cell, 2,000 episodes per model). Success (%) per suite × perturbation; the six CaP-Agent0-protocol cells average 57.3 (Opus 4.7) and 65.6 (Opus 5 +t) vs. CaP-Agent0's published 18.2 and $\pi_{0.5}$'s 12.8.

Suite	Task	Pos	Env	Lang	Obj	Mean
`Pigey` (Opus 4.7)						
Long (LIBERO-10)	22	20	66	90	60	52
Goal	44	52	58	90	76	64
Object	50	52	74	100	90	73
Spatial	80	66	78	90	94	82
Mean	49	48	69	92	80	**67.6**
`Pigey` (Opus 5 +t)						
Long (LIBERO-10)	18	28	66	86	62	52
Goal	66	58	66	92	80	72.4
Object	50	56	80	100	98	76.8
Spatial	96	68	82	94	96	87.2
Mean	58	53	74	93	84	**72.1**

H TOOL SCHEMAS

The JSON schemas exposed to the VLM reasoner are reproduced below.

```
[
  {
    "name": "Perceive", "description": "Capture the wrist-camera image
        and robot state (end-effector pose, gripper aperture, is_grasped)
         and return the set of detected object labels.", "input_schema":
        {"type": "object", "properties": {}, "required": []}
  }, {
    "name": "Pick", "description": "Grasp a detected object using the
        TAMP grasping backend (detect -> segment -> depth -> M2T2 grasp
        -> cuRobo plan -> execute -> close gripper).", "input_schema": {
      "type": "object", "properties": {
        "label": {"type": "string", "description": "An exact object
            label from the most recent Perceive."}
      }, "required": ["label"]
    }
```

```
  }, {
    "name": "DropAbove", "description": "Place the currently held object
        . Either target_label OR all three of (abs_x, abs_y, abs_z) MUST
       be provided. RELATIVE mode: pass target_label (+ optional dx_m,
       dy_m) and the drop point is reference_centroid + offset. ABSOLUTE
        mode: pass abs_x, abs_y, abs_z and the drop point is exactly
       that world coordinate (target_label is ignored).", "input_schema"
       : {
      "type": "object", "properties": {
        "target_label": {"type": "string", "description": "Required in
            RELATIVE mode."}, "dx_m": {"type": "number"}, "dy_m": {"type"
            : "number"}, "abs_x": {"type": "number", "description": "
            Required (with abs_y, abs_z) in ABSOLUTE mode."}, "abs_y": {"
            type": "number"}, "abs_z": {"type": "number"}
      }, "required": []
    }
  }, {
    "name": "VLARollout", "description": "Execute a physical subgoal
        using the pi0.5 visual policy.", "input_schema": {
      "type": "object", "properties": {
        "subgoal": {"type": "string", "description": "A short concrete
            manipulation command, e.g. put the red cup on the plate."}
      }, "required": ["subgoal"]
    }
  }, {
    "name": "Release", "description": "Open the gripper unconditionally
        (recovery if the wrong object was grasped).", "input_schema": {"
       type": "object", "properties": {}, "required": []}
  }, {
    "name": "Done", "description": "Signal that the task is complete.
       The agent must call Perceive immediately beforehand to verify the
        predicate against the current scene.", "input_schema": {"type":
       "object", "properties": {}, "required": []}
  }
]
```

The schemas above are the real-robot tool set: PICK and DROPABOVE are backed by the TAMP engine, and VLAROLLOUT is backed by the frozen $\pi_{0.5}$ policy. In all evaluated memory-task trials, the wrist camera was physically occluded by the experimenter during the perturbation; these executions are reported in Appendix F.2. The LIBERO-PRO experiments use the seven simulator tools described in Appendix H.1.

H.1 LIBERO-PRO TOOL INTERFACE

The interface is the same in simulation and on the robot: the reasoner is given the same skills as tools, the same argument schemas, the same per-call observations, and the same routing guidance. Only the implementation of the deterministic skill differs: GRASP/PLACE is a top-down grasp at the detected position rather than the real robot's grasp-synthesis and motion-planning pipeline. Every simulation configuration uses it, so it does not enter the differences we measure. VLAROLLOUT runs the frozen $\pi_{0.5}$-LIBERO policy.

I HARDWARE AND SOFTWARE SPECIFICS

Real robot. We use a Franka Research 3 arm with a Robotiq 2F-85 parallel gripper, two side-mounted Zed-2i RGB cameras, and one Zed-Mini wrist camera. The robot is controlled through a DROID/polymetis-style stack at the robot's native control rate. The agent process runs on a developer workstation and communicates with the robot and VLA policy server over a JSON-line protocol. Each VLAROLLOUT is capped at $K{=}600$ control steps (Appendix C).

Simulation. We evaluate in LIBERO-PRO using $\pi_{0.5}$-LIBERO as the motor substrate. Standard per-suite horizons are used for spatial, object, and goal suites. Each VLAROLLOUT runs the simulator faster than realtime.

Reasoner integration. VLM calls use each provider's standard API. Tool calls are returned as structured JSON.

Compute and software stack. The real-robot stack runs on two machines. A control NUC drives the robot through polymetis (zerorpc on port 4242, gRPC on 50 051) under the `PREEMPT_RT` kernel `6.8.0-rt8`, which polymetis requires for deterministic real-time control. A workstation (AMD Ryzen 7 9800X3D, 32 GB RAM, NVIDIA GeForce RTX 5090 32 GB, Ubuntu 24.04 LTS) hosts the perception and policy servers: the TiPToP grounding and grasp pipeline (open-vocabulary detection with Gemini Robotics-ER, segmentation with SAM 2, stereo depth with FoundationStereo, grasp synthesis with M2T2, and motion planning with cuRobo/cuTAMP), the $\pi_{0.5}$-DROID VLA policy server, and the `Pigey` agent process. The two GPU-resident components dominate VRAM ($\pi_{0.5}$-DROID inference $\approx$ 16 GB and FoundationStereo $\approx$ 8 GB) and fit on the single RTX 5090. Simulation (LIBERO-PRO with $\pi_{0.5}$-LIBERO) runs on a multi-GPU node equipped with NVIDIA B300 SXM6 GPUs (288 GB HBM each); one $\pi_{0.5}$-LIBERO server per GPU, with the LIBERO-PRO cells running in parallel.

J PROMPT STRUCTURE

The system prompt used in our real-robot experiments is structured around the following blocks; we summarize each below. The prompt and routing rules are frozen across all evaluations reported in this paper; the full prompt and code are in the supplementary material.

Tool definitions. The prompt lists each tool, its input schema, and its return shape, distinguishing deterministic sensor-derived fields (e.g. `is_grasped` on PICK) from judgments that require visual verification. A grasp flag alone does not establish that the intended object or task outcome is correct.

Skill routing. The routing guidance of Appendix E: perceive before acting, prefer the planner for rigid pick-and-place, escalate to VLAROLLOUT for deformables or when the planner fails, and call DONE only after a fresh PERCEIVE. The interface enforces the last rule; whether the task is complete is the reasoner's own judgment.

$\pi_{0.5}$ language conventions. VLAROLLOUT subgoals must stay close to the policy's training distribution: short, concrete commands such as *"pick up the red cup"* or *"put the cup in the basket"*; descriptors or hedges degrade the policy.

Task-agnostic content. The prompt gives general guidance on reading an instruction (quantifiers, comparatives, objects not yet detected) and names no task, object, scene, or coordinate.

Operating discipline. Every output ends with a tool call, and object arguments must be exact labels from the most recent PERCEIVE. The prompt does not train or specify motor control; it constrains *when* and *which* skill to call and what to verify before DONE.

K SCORING PROTOCOL

Each trial is scored binary success/failure. For real-robot tasks, success is determined by a human evaluator using the task specification and final scene state. For simulation tasks, success is determined by the LIBERO predicate. A trial that exhausts its budget is a failure. On the real robot, a trial in which DONE is emitted before the success condition is satisfied is also a failure. Each real-robot task is evaluated with 5 trials under matched initial scenes.

LIBERO-PRO protocol. We follow CaP-Agent0's published evaluation protocol: the *Spatial*, *Object*, and *Goal* suites under the *Pos* (the benchmark's `use_swap` flag) and *Task* perturbations, with an 8,000-step episode horizon matching CaP-Agent0's. For completeness we also evaluate `Pigey`

on the complete grid (Table 16): all four suites (adding *Long*) under all five perturbation types (adding *Env*, *Sem*, *Obj*), 100 episodes per cell (10 tasks $\times$ 10 seeds). We do not rerun CaP-Agent0 on the twelve additional cells: the frozen $\pi_{0.5}$ policy is already near ceiling there (81.5%), so they carry little discriminatory signal for agent orchestration, while a rerun would be costly at CaP-Agent0's documented $\sim$90 frontier-model calls per episode (about 108 000 calls).

Program-generation setup. The program-generation condition holds everything fixed except how the reasoner acts: the same tools are exposed as Python functions, and each turn the reasoner writes a program that is executed in a restricted interpreter (the tool functions, safe built-ins, and a small set of standard-library modules; no file, network, or process access). Tool return values and any exception are returned to the reasoner as the program's output, together with the post-execution observation, before the next turn. Turn and action budgets equal those of the one-action-per-turn condition, and a program that calls tools in a loop spends the action budget accordingly.

Direct frontier control setup. The direct condition changes only the toolset: same reasoner and thinking configuration, same object detections, camera views, and horizon, and a 30-call budget. The model controls the robot through a single tool, MOVE_TO$(x, y, z, \text{gripper})$: an absolute end-effector target with fixed top-down orientation, executed by a position controller (2 cm tolerance, at most 200 simulation steps), followed by the gripper change. Detected objects are numbered on the image with world coordinates but no labels, so the model must ground the instruction visually. One asymmetry favors this condition: its DONE is rejected while the task is incomplete, an oracle signal the full agent does not receive.

L Cost Accounting

Each real-robot `Pigey` trial uses 3–15 reasoner calls depending on task length and number of failures. Approximate API cost ranges from \$0.02 to \$0.50 per trial depending on model tier. Each call takes roughly 30 seconds, and wall-clock time is approximately 2–6 minutes per real-robot trial. In LIBERO the reasoner is capped at 30 calls per episode; solved episodes average 3.7 calls.

M Operational profile comparison

Table 17 profiles related systems along the capabilities the interface of Section 3 provides: whether the system keeps a history across steps, verifies the outcome of each action from observation, recovers when an action fails, chooses at run time among heterogeneous motor skills, exposes its decisions, and requires added training. ✓ denotes integral support; *Partial* denotes restricted support (e.g. scoring candidate actions before execution rather than checking the outcome after it); – denotes absence. Ratings follow the cited papers' descriptions of their released systems.

Table 17: Operational profile across modular manipulation systems.

System	History	Verify	Recover	Skill choice	Auditability	Added training
$\pi_{0.5}$ alone (Physical Intelligence, 2025)	–	–	–	–	Black-box	VLA
Hi Robot (Shi et al., 2025b)	✓	Partial	Partial	–	Partial	High-level policy + VLA
GR00T N1 (Bjorck et al., 2025)	–	–	–	–	Black-box	Joint VLM+VLA
Gemini Robotics (Google DeepMind, 2025)	Partial	–	–	–	Black-box	End-to-end VLA
TiPToP (Shen et al., 2026)	Partial	Partial	–	–	Explicit	None (planner)
CaP-Agent0 (Fu et al., 2026)	✓	✓	Partial	Partial	Explicit	None (code policy)
Inner Monologue (Huang et al., 2022)	✓	✓	✓	Partial	Explicit	Success detector
ECoT (Zawalski et al., 2024)	–	–	–	–	Explicit	CoT into VLA
MemER (Pan et al., 2026)	✓	–	–	–	Partial	Policy fine-tuning
RoboMonkey (Kang et al., 2025)	–	Partial	–	–	Partial	Action verifier
RoboMeter (Liang et al., 2026)	–	Partial	–	–	Partial	Reward model
`Pigey` (Ours)	✓	✓	✓	✓	Explicit	None