# HARNESS VLA: STEERING FROZEN VLAS INTO RELIABLE MANIPULATION PRIMITIVES VIA MEMORY-GUIDED AGENTS

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

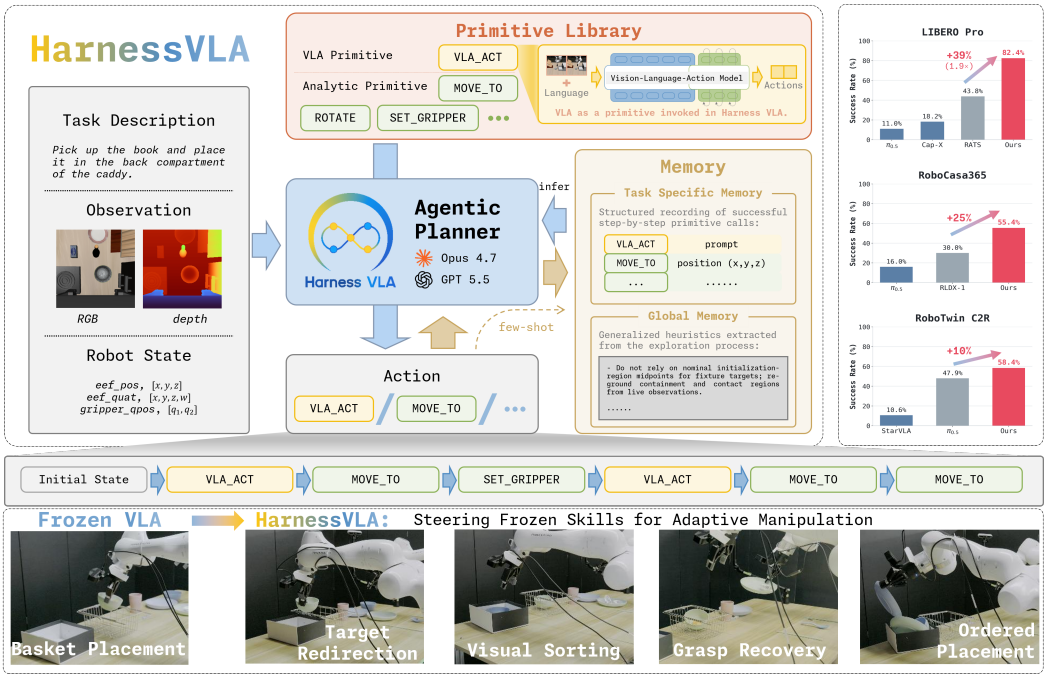



Figure 1: **Harness VLA system overview.** A memory-guided agentic planner composes frozen VLA contact skills with analytic primitives. The right panels summarize benchmark results; the command strip illustrates primitive composition. The bottom row contrasts frozen-VLA basket placement with Harness VLA's target redirection, visual sorting, grasp recovery, and ordered placement on dual-Franka robots, demonstrating adaptive manipulation with frozen skills.

## ABSTRACT

Language-conditioned manipulation requires both precise contact-rich control and robust reasoning over language, scenes, and long horizons. End-to-end Vision-Language-Action (VLA) models provide strong local visuomotor skills, but they are trained on in-distribution task trajectories and often degrade under deployment perturbations such as semantic retargeting, goal re-binding, spatial-layout shifts, and unstable local contacts. LLM coding agents provide complementary semantic and compositional reasoning, but purely analytic primitives struggle with irregular grasping, constrained placement, and articulated-object interaction. We present Harness VLA, a memory-augmented agentic framework that exposes a frozen VLA as a retryable contact-rich primitive and composes it with a small fixed library of analytic primitives for grounding, staging, transport, navigation, and release. Rather than expanding the skill library, the harness learns the operating range of these fixed primitives from task-specific execution traces, global success rules, and failure models. By lifting semantic re-grounding, non-contact execution, and VLA re-staging to the planner while reserving the frozen VLA for local



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.

contact-rich phases, Harness VLA extends pretrained VLAs beyond their original trajectory distribution without fine-tuning. Across perturbed tabletop, household kitchen, and clean-to-randomized bimanual manipulation, Harness VLA improves over the strongest relevant baselines by $38.6$ and $27.2$ percentage points on LIBERO-Pro and RoboCasa365, respectively, and reaches $58.4\%$ on RoboTwin C2R. Real-world demonstrations on dual-Franka robots further show target redirection, grasp recovery, and new task compositions with the same frozen VLA. Code is available at `https://anonymous.4open.science/r/HarnessVLA2027/README.md`.

## 1 INTRODUCTION

A long-standing goal of robotic manipulation is a system that reliably executes free-form natural-language instructions across changing objects, layouts, and embodiments. Two dominant paradigms approach this goal from opposite directions. End-to-end Vision-Language-Action (VLA) models learn contact-rich visuomotor control directly from robot trajectories, while LLM coding agents use language-model reasoning to compose explicit perception-and-control APIs. Each paradigm is powerful, but each assigns the wrong component too much responsibility: monolithic VLAs must absorb language grounding, long-horizon composition, and low-level control inside a single policy, whereas coding agents must realize physically delicate interactions through hand-designed or agent-generated APIs. Figure 2 visualizes our response: use analytic primitives to traverse deployment perturbations and invoke the VLA only inside local contact-rich regions where its training distribution is informative.

**End-to-end VLA models** have advanced rapidly, from generalist robot policies (Brohan et al., 2023; Kim et al., 2024) to flow-matching and action-reasoning architectures (Black et al., 2025; Physical Intelligence et al., 2025; Hung et al., 2025; Lee et al., 2025). Their strength is local, image-conditioned contact: grasping irregular objects, placing with tight tolerances, or actuating fixtures that are brittle for analytic controllers. Their weakness is deployment outside the trajectory distribution on which they were trained. A model trained on in-distribution task trajectories may know how to grasp a milk carton or turn a faucet, yet fail when semantic targets are redirected, goal predicates are re-bound, object layouts shift, or short skills must be composed into longer routines. Under such deployment perturbations, the policy may repeat a familiar training-time behavior even when the instruction or scene binding has changed (Kim et al., 2024; Physical Intelligence et al., 2025; Wang et al., 2025a); a single unstable contact failure can also derail the entire monolithic rollout.

**LLM coding agents and harnesses** provide complementary semantic and compositional reasoning. Systems such as Code as Policies and ProgPrompt synthesize executable programs over curated perception and control APIs (Liang et al., 2023; Singh et al., 2023), and recent multimodal or agentic variants extend this idea with richer perception, tool use, feedback, and persistent execution state (Gupta & Kembhavi, 2023; Shi et al., 2025; Fu et al., 2026; Zhang et al., 2026). More broadly, coding-agent harnesses wrap model outputs in a structured runtime with tool interfaces, memory, validators, execution loops, and feedback channels, allowing an agent to revise decisions, write successful traces or failure diagnoses back into memory, and orchestrate heterogeneous tools under a common control surface (Wang et al., 2024; Yang et al., 2024; Wang et al., 2025b; Ning et al., 2026). Yet in robot manipulation, scaling such systems often still means expanding the primitive or skill library, while purely analytic primitives–deterministic kinematic or model-based controllers such as IK transport, wrist rotation, base motion, gripper opening, and release–remain poorly suited to irregular grasping, constrained placement, and articulated-object manipulation.

**Harness VLA** instantiates this coding-agent harness view for robot manipulation: keep the primitive library fixed and small, and let the agent learn how to orchestrate it. The planner composes analytic primitives for non-contact structure such as target grounding, free-space transport, posture adjustment, mobile staging, re-staging after failed attempts, and release. For contact-rich phases, it invokes a frozen VLA through a single learned primitive, VLA_ACT. This converts the VLA from a monolithic trajectory policy into a reusable contact specialist, extending it to the tasks outside its original trajectory distribution without fine-tuning or deployment-time primitive expansion. The key is not only exposing VLA_ACT, but learning when and how to use it. Harness VLA treats VLA execution as a retryable local attempt: the planner can stage the robot into a favorable local obser-

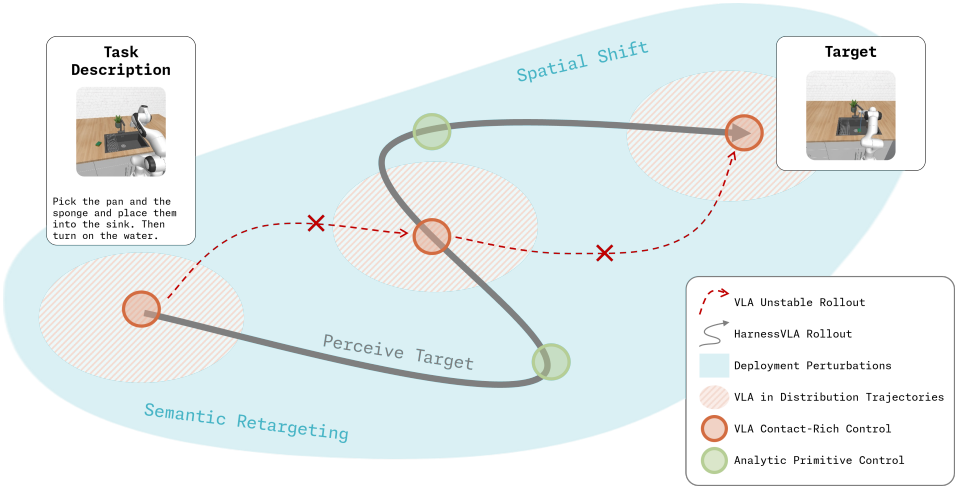



Figure 2: **Primitive composition extends a frozen VLA beyond its trajectory distribution.** Deployment perturbations expand the possible task configurations beyond the in-distribution trajectories covered by the frozen VLA. A direct VLA rollout may attempt to bridge the perturbed space and fail before reaching the target. Harness VLA instead decomposes the task into local contact-rich VLA invocations and analytic primitive control: analytic primitives perceive the current target, reground task bindings, and move the robot between VLA-compatible local regions, while VLA_ACT is invoked only for contact-rich phases inside those regions.

vation, invoke the VLA, inspect the contact outcome, and re-stage if needed. Two memory modules support this process inside the agentic harness (Wang et al., 2024; Yang et al., 2024; Wang et al., 2023): task-specific traces store successful primitive compositions for few-shot re-grounding, while global memory stores reusable success rules and failure models. Rather than adding more skills, the harness teaches the planner the operating range of each fixed primitive: which subproblems should be handled analytically, when VLA_ACT is appropriate, and how failed contact attempts should be re-staged. Our core contributions are:

- A memory-augmented agentic framework for using a frozen VLA as a primitive. Harness VLA composes VLA_ACT with fixed analytic primitives, extending a pretrained VLA from local contact-rich control to long-horizon, perturbed manipulation without fine-tuning the VLA or expanding the primitive vocabulary at deployment time.
- An empirical analysis showing why a small fixed primitive library is sufficient when the planner learns how to use it. Repeated planner-staged invocations can reframe brittle VLA attempts, while analytic primitives solve much of the non-contact structure around each contact-rich phase.
- Strong benchmark results across standard and perturbed tabletop manipulation, household kitchen manipulation, and clean-to-randomized transfer. Harness VLA preserves competitive standard LIBERO performance, improves over the strongest relevant baselines by 38.6 and 27.2 percentage points on LIBERO-Pro and RoboCasa365, respectively, and reaches 58.4% on RoboTwin C2R. We further conduct real-world experiments on dual-Franka robots to illustrate the mechanisms of Harness VLA.

## 2 The Harness VLA Framework

Our agentic framework for language-conditioned manipulation follows the system view in Figure 1. A task description, RGB-D observations, and robot state are passed to an agentic planner, which reasons over a fixed primitive library and retrieves context from Task Specific Memory and Global Memory. The agentic harness (Section 2.2) couples this planner to the simulator through a JSON-serialized primitive interface, drives the turn-based execution loop, and writes successful exploration traces into Task Specific Memory while committing generalized heuristics to Global Memory. The

primitive library defines the only operations the planner is allowed to invoke: a small set of analytic primitives together with a structurally special VLA primitive that encapsulates a pretrained visuomotor policy for contact-rich interactions. Section 2.1 first formalises the task and the iterative execution loop on which these components are built.

Appendix A details the Unified Primitive Interface, including the primitive vocabulary, JSON invocation schema, and VLA stopping criteria.

### 2.1 Problem Formulation and Agentic Execution Loop

**Task setup.** We consider language-conditioned robotic manipulation within an environment $\mathcal{E}$ driven by a rigid-body physics engine (e.g., MuJoCo via Robosuite). At each timestep $t$, the environment exposes a multimodal observation tuple $o_t = (I_t^{\text{rgb}}, I_t^{\text{d}}, q_t)$ comprising an agent-view RGB image $I_t^{\text{rgb}}$, a co-aligned metric depth map $I_t^{\text{d}}$, and a robot proprioceptive state $q_t$ (concatenating the end-effector pose and gripper state). A task is defined by a natural-language description $\ell$ alongside a binary completion predicate $\mathcal{G}$, exposed solely as a sparse success signal at episode termination.

**Agentic execution loop.** As illustrated by the rollout strip in Figure 1, a task rollout is formulated as an autoregressive, turn-based interaction between a high-level agentic planner $\Pi$ and the underlying physics engine. Instead of treating the visuomotor policy as a separate hierarchical tier, we unify all low-level control mechanisms—including the frozen pretrained VLA $f_\theta$ and all deterministic operational-space controllers—into a single, predefined primitive library $\mathcal{P}$.

At each execution turn $t$, the planner $\Pi$ processes the current multimodal observation $o_t$, the task description $\ell$, and the retrieved context from both the Task Specific Memory and Global Memory. Operating as the sole cognitive orchestrator, $\Pi$ emits a structured JSON invocation for a selected primitive $c_t \in \mathcal{P}$. The physics engine directly receives this invocation and executes the corresponding physical motions in the simulator until the primitive's internal post-condition is met. Upon primitive termination, the engine yields the subsequent observation $o_{t+1}$ and updated robot state $q_{t+1}$. This environment-planner loop iterates continuously until the goal predicate $\mathcal{G}$ is satisfied or a maximum step budget is exhausted.

### 2.2 The Harness VLA Architecture

Motivated by recent coding agents, which make model decisions executable through harnessed execution-feedback loops (Wang et al., 2024; Yang et al., 2024; Wang et al., 2025b), Harness VLA packages robot manipulation in the same REPL-style form. The harness is the runtime contract between the planner and the environment: it exposes primitive schemas, serializes decisions as JSON commands, executes primitives, refreshes RGB-D and proprioceptive observations, logs traces, retrieves Task Specific Memory and Global Memory, enforces reset and budget policies, and checks progress through the benchmark predicate (Ning et al., 2026). Appendix I details the REPL protocol, and Appendix M specifies the agent prompts.

Because this harness delegates all fine-grained execution to the primitive library, the agentic planner $\Pi$ is freed to focus entirely on compositional reasoning. To do so, it relies heavily on the multimodal observation channel: the RGB image supports qualitative scene reasoning (e.g., clutter, semantic identity), while the co-aligned depth map and proprioception supply metric spatial data for precise localization.

We structure the agent's lifecycle within this harness into two distinct phases: an exploratory bootstrapping phase and a rigorous deployment evaluation phase.

**Exploratory Bootstrapping Phase.** Operating on a single reference instantiation of a task, the agent autonomously interacts with the environment to discover a working solution. During this phase, the planner $\Pi$ is uniquely granted access to a RESET primitive and operates under a generous wall-clock budget. Because the primitive vocabulary is fixed, the exploration focuses entirely on iterative composition: discovering the optimized orchestration of the learned VLA primitive and the deterministic analytic primitives. The planner $\Pi$ repeatedly trials different staging orders, pre-contact poses, invocation timings for VLA_ACT, and early-return termination thresholds. It observes the physical effects of each primitive call and corrects course upon failure.

Upon successful task completion, the agent systematically abstracts its experience into the two memory modules shown in Figure 1. First, the verified sequence of primitive invocations is serialized into a JSONL format. This file explicitly records the successful step-by-step primitive calls, parameterizing them by replacing concrete spatial coordinates with symbolic perception queries to make the sequence reusable across different spatial layouts. This parameterized JSONL trace is stored in the **Task Specific Memory** to serve as a structural prior for subsequent generalization tests. Second, the agent extracts generalized heuristics from the exploration process and commits them to a persistent **Global Memory**. This shared repository explicitly aggregates *success rules*, such as optimal prompting strategies that utilize the full task instruction. It concurrently documents critical *failure models*, including the identification of empty-grasp executions and false success detections. This aggregation ensures the planner avoids repeating historical pitfalls across different tasks.

**Deployment Evaluation Phase.** During formal evaluation on unseen environment variations, including position swaps, instruction redirections, and testing across multiple initial state seeds, the harness imposes a strict execution regime. The RESET primitive is completely disabled, and the operational step budget is significantly shortened. To solve the perturbed tasks, the planner $\Pi$ retrieves the pre-computed JSONL trace from the **Task Specific Memory** and grounds it dynamically using the live RGB-D observation. By referencing the success rules and failure models accumulated in the **Global Memory**, the agent executes the trajectory deterministically. The performance achieved under this strict phase directly constitutes our reported benchmark results, validating the overall effectiveness of the Harness VLA framework.

## 3 EXPERIMENTS

We evaluate few-shot execution with bootstrapped harness memory and zero-shot execution without target-setting Task Specific Memory or Global Memory. Experimental settings and environment details are provided in Appendices B and K. Appendix P reports additional results with GPT-6 Astra as the planner.

### 3.1 OVERALL BENCHMARK PERFORMANCE

**Few-shot evaluation with Task Specific Memory.** We use Codex (GPT-5.5) and Claude Code (CC; Opus-4.7) as planners. For each benchmark, both use the same frozen VLA, with task memory bootstrapped on one reference seed per task and re-grounded on held-out evaluation seeds. Standard LIBERO results are reported in Appendix D (Table 6).

Table 1: Aggregate LIBERO-Pro success rates (%) across SPATIAL, OBJECT, GOAL, and LIBERO-10 under instruction-redirection (**T**) and position-swap (**S**) perturbations. Each cell aggregates 100 trials (10 tasks $\times$ 10 seeds); "/" indicates an unavailable or unreported cell. Cap-X and RATS report only the six non-LIBERO-10 cells, so their overall values average over reported cells only. $\pi_{\text{RLinf}}$ and Harness VLA are evaluated by us using the RLinf-released `pi05_libero130_fullshot` $\pi_{0.5}$-SFT checkpoint (Yu et al., 2025); Harness VLA exposes this frozen checkpoint through the VLA_ACT primitive. Bold marks the best reported method in each evaluation cell or overall column.

| Method | Spat-T | Spat-S | Obj-T | Obj-S | Goal-T | Goal-S | L10-T | L10-S | **Overall** |
|---|---|---|---|---|---|---|---|---|---|
| OpenVLA (Kim et al., 2024) | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| $\pi_0$ (Black et al., 2025) | 0.0 | 0.0 | 0.0 | 2.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.3 |
| $\pi_{0.5}$ (Physical Intelligence et al., 2025) | 1.0 | 20.0 | 1.0 | 17.0 | 2.0 | 38.0 | 1.0 | 8.0 | 11.0 |
| MolmoAct (Lee et al., 2025) | 0.0 | 0.0 | 0.0 | 6.0 | 0.0 | 0.0 | 6.0 | 0.0 | 1.5 |
| NORA (Hung et al., 2025) | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| X-VLA (Zheng et al., 2025) | 0.0 | 0.0 | 8.0 | 2.0 | 9.0 | 1.0 | 10.0 | 0.0 | 3.8 |
| AtomVLA (Sun et al., 2026) | 1.0 | 16.0 | 0.0 | 10.0 | 11.0 | 2.0 | 9.0 | 1.0 | 6.3 |
| Cap-X (Fu et al., 2026) | 14.0 | 12.0 | 18.0 | 22.0 | 17.0 | 26.0 | / | / | 18.2 |
| RATS (Zhang et al., 2026) | 31.0 | 29.0 | 63.0 | 61.0 | 36.0 | 43.0 | / | / | 43.8 |
| $\pi_{\text{RLinf}}$ | 42.0 | 59.0 | 71.0 | 78.0 | 45.0 | 42.0 | 49.0 | 14.0 | 50.0 |
| **Harness VLA (Codex)** | 81.0 | 69.0 | **94.0** | **91.0** | 75.0 | 66.0 | 52.0 | 49.0 | 72.1 |
| **Harness VLA (CC)** | **94.0** | **80.0** | 88.0 | 90.0 | **87.0** | **87.0** | **71.0** | **62.0** | **82.4** |

**LIBERO-Pro.** Table 1 reports 72.1% success with Codex and 82.4% with CC, compared with 50.0% for the same frozen $\pi_{\mathrm{RLinf}}$ policy executed directly. CC exceeds RATS's reported overall rate (43.8%) by 38.6 percentage points, although RATS covers only the six non-LIBERO-10 cells. Gains under both instruction redirection and position swaps show that planner-level adaptation extends the frozen backbone's robustness.

**RoboCasa365.** Harness VLA reaches 57.2% with Codex and 48.8% with CC, compared with 30.0% for the direct frozen RLDX-1 baseline (Table 2); Codex improves by 27.2 percentage points. These results extend the benefit of harness composition to mobile kitchen manipulation with articulated fixtures and longer composite routines.

Table 2: RoboCasa365 success rates (%). Baseline rows above the separator are results reported in prior work and the official leaderboard on ATOMIC-SEEN, COMPOSITE-SEEN, and COMPOSITE-UNSEEN, with RLDX-1 evaluated as the direct frozen-VLA baseline under our protocol. Harness VLA uses one reference seed only for bootstrapping; reported evaluation uses five held-out seeds $s_1$–$s_5$ per task in all three splits, totaling 250 rollouts. Bold marks the best method in each split.

| Method | Atomic-Seen | Composite-Seen | Composite-Unseen | **Overall** |
|---|---|---|---|---|
| RLDX-1 (Kim et al., 2026) | 60.0 | 21.3 | 5.0 | 30.0 |
| WorldDreamer (World Agents, 2026) | 66.3 | 26.7 | 9.0 | 35.3 |
| $\pi_{0.5}$ (Physical Intelligence et al., 2025) | 39.6 | 7.1 | 1.2 | 16.9 |
| $\pi_0$ (Black et al., 2025) | 34.6 | 6.1 | 1.1 | 14.8 |
| **Harness VLA (Codex)** | **92.2** | **61.3** | 13.8 | **57.2** |
| **Harness VLA (CC)** | 80.0 | 47.5 | **15.0** | 48.8 |

**Zero-shot evaluation without bootstrapped harness memory.** On LIBERO-PRO GOAL, zero-shot Harness VLA (CC) reaches 31.0% under position swaps and 79.0% under instruction redirection, compared with Cap-X's 25.6% and 16.8% (Table 3). Few-shot CC reaches 87.0% in both settings (Table 1). Without harness memory, the planner retains much of its instruction-following ability, but performance drops substantially under spatial perturbations. This difference indicates that the primitive organization discovered during exploration is particularly useful for transfer across object layouts.

Table 3: Per-task success rate (%) on LIBERO-PRO GOAL: zero-shot Harness VLA (CC) (no Task Specific Memory retrieval, 10 seeds/task) vs Cap-X (Fu et al., 2026). **Pos** = swap (S), **Task** = instruction-redirection (T). Bold marks the winner within each setting and task or average column.

| Setting | Method | Task 0 | Task 1 | Task 2 | Task 3 | Task 4 | Task 5 | Task 6 | Task 7 | Task 8 | Task 9 | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Pos (S)** | Cap-X | 0.0 | 4.0 | 0.0 | **36.0** | 22.0 | **60.0** | 4.0 | 2.0 | 62.0 | **66.0** | 25.6 |
| | Harness VLA (CC) | 0.0 | **10.0** | 0.0 | 20.0 | **90.0** | 0.0 | **10.0** | **80.0** | **100.0** | 0.0 | **31.0** |
| **Task (T)** | Cap-X | 0.0 | 0.0 | 10.0 | 38.0 | 12.0 | 4.0 | 34.0 | 12.0 | 40.0 | 18.0 | 16.8 |
| | Harness VLA (CC) | **10.0** | **100.0** | **90.0** | **100.0** | **20.0** | **80.0** | **90.0** | **100.0** | **100.0** | **100.0** | **79.0** |

RoboTwin C2R results and analyses are provided in Appendix E, including the benchmark comparison in Table 7 and the mechanism analyses in Figures 8 and 9.

### 3.2 EXPERIMENT ANALYSIS

Unless otherwise noted, analyses use Harness VLA (CC) with the same benchmark-specific frozen VLA as the main evaluation. Appendix F provides the detailed analysis and rollout examples.

**Key Finding 1: Planner-level semantic re-grounding restores task-conditioned behavior.** Figure 3 shows two failures of the frozen $\pi_{\mathrm{RLinf}}$ policy: it repeats the standard behavior when the instruction redirects the target, and moves toward the training-time region after an object-position swap. Harness VLA instead grounds the requested target and destination in the current observation, stages the robot, and delegates the local contact phase to the same frozen policy. The improvement comes from planner-level semantic and spatial binding; Appendix F gives the full case analysis.

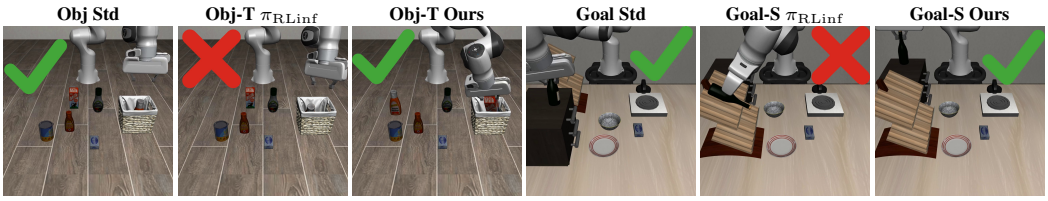
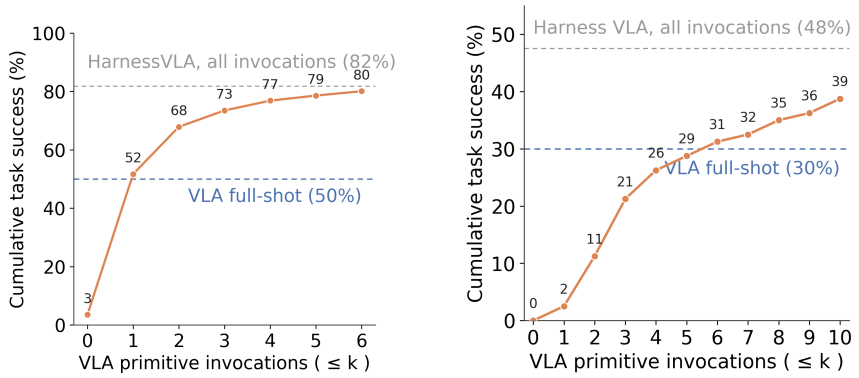



Figure 3: Terminal-state comparisons on OBJECT-PRO instruction redirection (left) and GOAL-PRO position swap (right). Each triplet shows $\pi_{\text{RLinf}}$ on the standard task, $\pi_{\text{RLinf}}$ on the perturbed task, and Harness VLA on the same perturbed task.

**Key Finding 2: Planner-staged VLA invocation improves frozen-policy reliability.** Figure 4 reports cumulative success as the maximum number of VLA calls increases. A few planner-selected invocations already exceed the frozen-policy baseline, while additional calls improve success on longer or more contact-heavy tasks. Staging and retry support this pattern: the planner restores a VLA-compatible local observation around the intended contact target, then inspects the outcome and re-stages the robot if contact is incomplete or unstable. A failed attempt can thus remain local to one subtask instead of terminating the rollout. The harness learns where the VLA should begin acting, rather than asking the frozen policy to absorb the full distribution shift. On PRESOAK-PAN, for example, the planner adjusts the mobile base and arm pose, retries until the pan is stably grasped, and later invokes the VLA again to turn the faucet. VLA calls remain sparse, with analytic primitives coordinating the surrounding motion. Appendix F details these mechanisms and provides representative re-staging and retry examples (Figure 10).




(a) LIBERO-PRO (b) ROBOCASA365

Figure 4: **Adaptive VLA invocation.** Cumulative task success versus the maximum number of VLA calls per episode on LIBERO-PRO and ROBOCASA365. Blue dashed lines mark direct frozen-policy baselines; gray dashed lines mark full Harness VLA performance.

**Key Finding 3: Analytic primitives isolate non-contact execution from contact-rich control.** Figure 5 attributes each successful rollout to the primitive that triggers the final benchmark completion predicate. Analytic primitives trigger 72–84% of completions across LIBERO Pro-family suites, typically through transport, release, or repositioning after stable contact. In RoboCasa365, 60–86% finish inside the VLA because completion requires fixture actuation or constrained placement. These outcomes support the division of labor: analytic primitives organize execution around the contact-rich phases handled by the frozen VLA. By handling the intervening motion, the planner reuses the same learned contact skills across longer routines without expanding the primitive library. Appendix N reports primitive usage, and Appendix F provides the expanded analysis and rollout examples (Figure 11).

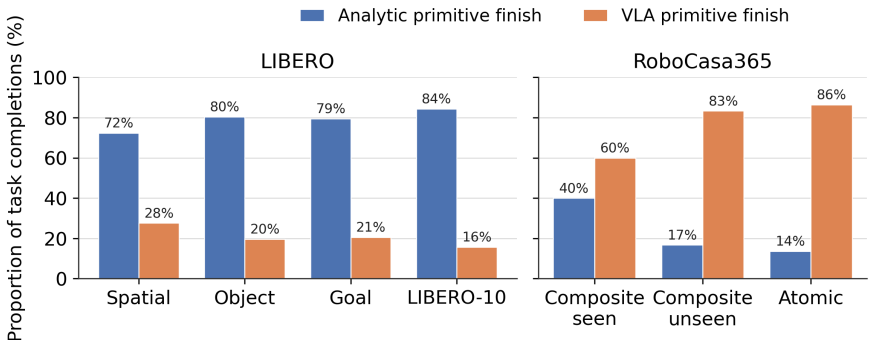



Figure 5: **Task completion attribution.** Fraction of successful rollouts whose final benchmark completion predicate fires after an analytic primitive (blue) or a VLA primitive (orange), grouped by LIBERO Pro-family suite and RoboCasa365 split.

## 4 Flash Mode and Real-World Deployment

Harness VLA extends the memory-guided primitive interface to two deployment settings: Flash Mode executes prepared procedures without online agent planning, while physical deployment uses the same separation between frozen contact skills and task-level coordination.

### 4.1 Flash Mode

Flash Mode executes a successful primitive sequence as a task card without online agent planning. It re-grounds object-relative waypoints in the current scene and targets simple tasks with a fixed subgoal order; new subgoals or unforeseen recovery require the online planner. Appendix O gives the execution, generation, and selection details.

**Success and execution time.** Table 4 reports $72.63\%$ success $(581/800)$ for Flash Mode (max) on LIBERO-Pro. The best card is selected and reported on the same seeds, so this is not a held-out estimate. Flash and Codex timings measure source-trajectory tool durations and planner durations, respectively, and do not establish an end-to-end speedup.

Table 4: Success rates (%) and execution time of Flash Mode and online planning on LIBERO-Pro. **T/S** denote instruction-redirection/position-swap perturbations; each suite contains 100 episodes. † Flash time averages source-trajectory tool durations; Codex time averages planner durations. Appendix O gives the timing protocol and same-seed max-card selection details. Missing cards count as failures.

| Method | Spat-T | Spat-S | Obj-T | Obj-S | Goal-T | Goal-S | L10-T | L10-S | Overall | Time (s) |
|---|---|---|---|---|---|---|---|---|---|---|
| Codex (reasoning high) | 88.0 | 88.0 | 95.0 | 98.0 | 77.0 | 56.0 | 68.0 | 58.0 | 78.50 | 529.23 |
| Codex (no reasoning) | 86.0 | 67.0 | 94.0 | 92.0 | 15.0 | 64.0 | 43.0 | 39.0 | 62.50 | 344.17 |
| Flash Mode (max) | 79.0 | 70.0 | 86.0 | 93.0 | 74.0 | 65.0 | 60.0 | 54.0 | 72.63 | 56.17† |

### 4.2 Real-World Deployment

On two dual-Franka installations, we compare fullshot execution of the frozen VLA with online Harness VLA (Figure 6) and Flash Mode (Figure 7). Online planning redirects destinations and action order and recovers from missed grasps; Flash Mode composes contact skills into pouring and plate wiping. These qualitative demonstrations show capability extension with frozen VLA weights. Appendix G provides deployment details, human-in-the-loop resets and success judgments, and case analysis.

Related work is reviewed in Appendix C; the conclusion, limitations, and future directions are provided in Appendix H.




Figure 6: **From the learned routine to online adaptation.** Rows read left to right. Fullshot VLA executes grasp–handoff–basket placement. Harness VLA redirects bowls to a box, orders green-then-blue plate transfers, sorts bowls by a visible dirty marker, and inspects and retries a missed grasp. The same frozen contact skills support all four adaptations.

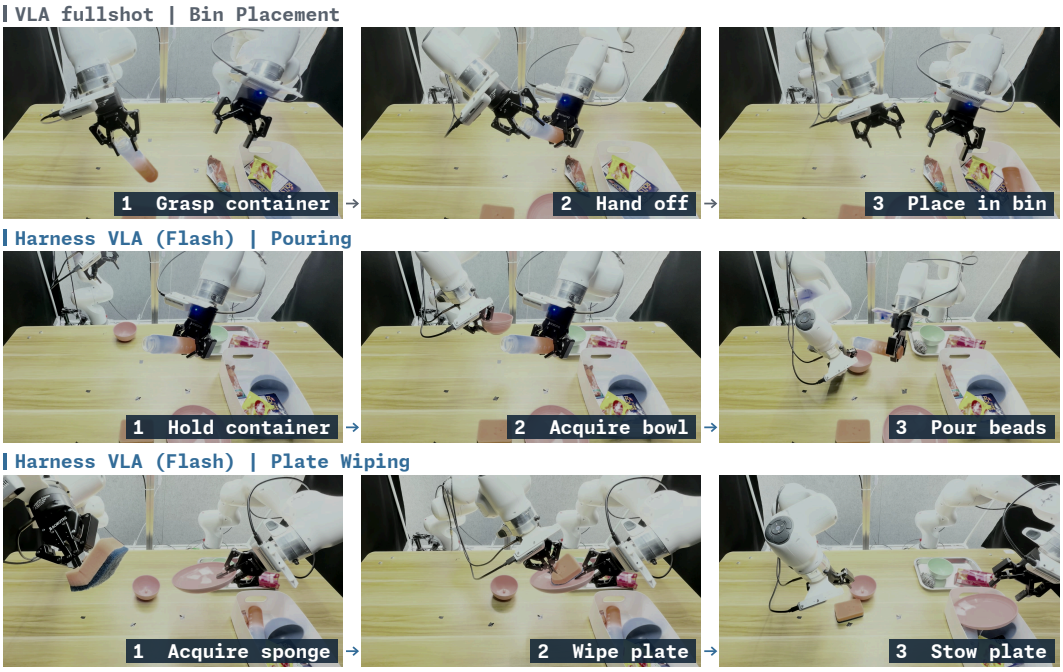



Figure 7: **From the learned routine to Flash composition.** Rows read left to right. Fullshot VLA executes grasp–handoff–bin placement. Flash Mode composes the frozen contact skills to pour steel beads into a receiving bowl and to acquire a sponge, wipe a held plate, and stow it. Both procedures execute without online agent planning.

## AI USE STATEMENT

Generative AI tools assisted with manuscript drafting and organization, language editing, LaTeX preparation, reference checking, and the explanation and interpretation of experimental results. Language models are also components of the evaluated system: they perform visual grounding, task planning, and memory-guided primitive composition through the interfaces described in Section 2 and Appendices A and M. Flash task-card generation and selection are described in Appendix O. The authors take responsibility for the final manuscript, including its claims, citations, experimental reporting, and AI-assisted content.

## REPRODUCIBILITY STATEMENT

The execution loop and memory workflow are described in Section 2. Appendices A and B specify the primitive interface and experimental configuration; Appendices I and M provide the REPL contract and agent prompts. Benchmark task splits, evaluation seeds, success criteria, and aggregation protocols are given in Appendix K, while the frozen VLA instantiations and RoboTwin post-training configuration appear in Appendix L. Appendix E reports the RoboTwin C2R results, and Appendix P reports the additional planner evaluation. Appendix O documents Flash card generation, selection on the same seeds used for reporting, and timing definitions. Appendix G specifies the human involvement in physical scene resets and success judgments; the real-world demonstrations are qualitative capability examples.

## REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Christopher Watkins, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, Andy Zeng, and Yi Zheng. Do as I can, not as I say: Grounding language in robotic affordances. In *Conference on Robot Learning (CoRL)*, 2022.

Anthropic. Introducing Claude Sonnet 4.5. *Online*, 2025.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. $\pi_0$: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems (RSS)*, 2025.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning (CoRL)*, 2023.

Boyuan Chen, Zhuo Xu, Sean Kirmani, Brian Ichter, Dorsa Sadigh, Leonidas Guibas, and Fei Xia. SpatialVLM: Endowing vision-language models with spatial reasoning capabilities. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024a.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations (ICLR)*, 2024b.

Yongchao Chen, Jacob Arkin, Charles Dawson, Yang Zhang, Nicholas Roy, and Chuchu Fan. AutoTAMP: Autoregressive task and motion planning with LLMs as translators and checkers. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024c.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Robotics: Science and Systems (RSS)*, 2023.

StarVLA Community. Starvla: A lego-like codebase for vision-language-action model developing. *arXiv preprint arXiv:2604.05014*, 2026.

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, et al. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PaLM-E: An embodied multimodal language model. In *International Conference on Machine Learning (ICML)*, 2023.

Max Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Fei-Fei Li, Guanya Shi, et al. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026.

Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Xinyu Geng, Peng Xia, Zhen Zhang, Xinyu Wang, Qiuchen Wang, Ruixue Ding, Chenxi Wang, Jialong Wu, Yida Zhao, Kuan Li, et al. Webwatcher: Breaking new frontier of vision-language deep research agent. *arXiv preprint arXiv:2508.05748*, 2025.

Andrew Goldberg, Kavish Kondap, Tianshuang Qiu, Zehan Ma, Letian Fu, Justin Kerr, Huang Huang, Kaiyuan Chen, Kuan Fang, and Ken Goldberg. Blox-net: Generative design-for-robot-assembly using VLM supervision, physics simulation, and a robot with reset. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

Jiangyong Huang, Silong Yong, Xiaojian Ma, Xiongkun Linghu, Puhao Li, Yan Wang, Qing Li, Song-Chun Zhu, Baoxiong Jia, and Siyuan Huang. An embodied generalist agent in 3d world. In *International Conference on Machine Learning (ICML)*, 2024.

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. *arXiv preprint arXiv:2305.11176*, 2023a.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023b.

Wenxuan Huang, Yu Zeng, Qiuchen Wang, Zhen Fang, Shaosheng Cao, Zheng Chu, Qingyu Yin, Shuang Chen, Zhenfei Yin, Lin Chen, et al. Vision-deepresearch: Incentivizing deepresearch capability in multimodal large language models. *arXiv preprint arXiv:2601.22060*, 2026.

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.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Siddharth Karamcheti, Suraj Nair, Ashwin Balakrishna, Percy Liang, Thomas Kollar, and Dorsa Sadigh. Prismatic vlms: Investigating the design space of visually-conditioned language models. *arXiv preprint arXiv:2402.07865*, 2024.

Dongyoung Kim, Huiwon Jang, Myungkyu Koo, Suhyeok Jang, Taeyoung Kim, Beomjun Kim, Byungjun Yoon, Changsung Jang, Daewon Choi, Dongsu Han, et al. Rldx-1 technical report. *arXiv preprint arXiv:2605.03269*, 2026.

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, Winson Han, Wilbert Pumacay, Angelica Wu, Rose Hendrix, Karen Farley, Eli VanderBilt, Ali Farhadi, Dieter Fox, and Ranjay Krishna. MolmoAct: Action reasoning models that can reason in space. *arXiv preprint arXiv:2508.07917*, 2025.

Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, et al. Websailor: Navigating super-human reasoning for web agent. *arXiv preprint arXiv:2507.02592*, 2025.

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. *arXiv preprint arXiv:2603.11558*, 2026.

Xinghang Li, Minghuan Liu, Hanbo Zhang, Cunjun Yu, Jie Xu, Hongtao Wu, Hanwei Dong, Haifeng Hu, Wanmeng Zhan, Hao Wu, Yifan Han, and Tao Kong. Vision-language foundation models as effective robot imitators. In *International Conference on Learning Representations (ICLR)*, 2024.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023a.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 36, 2023b.

Runyu Lu, Yubo Wu, Ethan Kou, Letian Fu, Wenli Xiao, Ajay Mandlekar, Yinzhen Xu, Guanya Shi, Ken Goldberg, Ang Chen, et al. ASPIRE: Agentic/skills discovery for robotics. *arXiv preprint arXiv:2607.00272*, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

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

Meta. Llama 4 Herd. *Meta Blog*, 2025.

Yao Mu, Qinglong Zhang, Mengkang Hu, Wenhai Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng Dai, Yu Qiao, and Ping Luo. EmbodiedGPT: Vision-language pre-training via embodied chain of thought. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Yao Mu, Junting Chen, Qinglong Zhang, Shoufa Chen, Qiaojun Yu, Chongjian Ge, Runjian Chen, Zhixuan Liang, Mengkang Hu, Chaofan Tao, Peize Sun, Haibao Yu, Chao Yang, Wenqi Shao, Wenhai Wang, Jifeng Dai, Yu Qiao, Mingyu Ding, and Ping Luo. RoboCodeX: Multimodal code generation for robotic behavior synthesis. *arXiv preprint arXiv:2402.16117*, 2024.

Yao Mu, Tianxing Chen, Zanxin Chen, Shijia Peng, Zhiqian Lan, Zeyu Gao, Zhixuan Liang, Qiaojun Yu, Yude Zou, Mingkun Xu, et al. RoboTwin: Dual-arm robot benchmark with generative digital twins. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A large-scale simulation framework for training and benchmarking generalist robots. *arXiv preprint arXiv:2603.04356*, 2026.

Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, et al. Code as agent harness. *arXiv preprint arXiv:2605.18747*, 2026.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. *Online*, 2023.

OpenAI. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

OpenAI. Introducing deep research. *OpenAI Blog*, 2025.

OpenAI. Introducing GPT-5.2. *Online*, 2025.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration. In *International Conference on Robotics and Automation (ICRA)*, 2024.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Sundar Pichai, Demis Hassabis, and Koray Kavukcuoglu. A new era of intelligence with Gemini 3. *Google Blog*, 2025.

Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. SayPlan: Grounding large language models using 3D scene graphs for scalable task planning. In *Conference on Robot Learning (CoRL)*, 2023.

Junyao Shi, Rujia Yang, Kaitian Chao, Bingqing Selina Wan, Yifei Simon Shao, Jiahui Lei, Jianing Qian, Long Le, Pratik Chaudhari, Kostas Daniilidis, et al. Maestro: Orchestrating robotics modules with vision-language models for zero-shot generalist robots. In *NeurIPS 2025 Workshop on Space in Vision, Language, and Embodied AI*, 2025.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *Conference on Robot Learning (CoRL)*, 2022.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

Xiaoquan Sun, Zetian Xu, Chen Cao, Zonghe Liu, Yihan Sun, Jingrui Pang, Ruijian Zhang, Zhen Yang, Kang Pang, Dingxin He, et al. AtomVLA: Scalable post-training for robotic manipulation via predictive latent world models. *arXiv preprint arXiv:2603.08519*, 2026.

Dídac Surís, Sachit Menon, and Carl Vondrick. ViperGPT: Visual inference via Python execution for reasoning. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Gemini Robotics Team, Abbas Abdolmaleki, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Ashwin Balakrishna, Nathan Batchelor, Alex Bewley, Jeff Bingham, et al. Gemini robotics 1.5: Pushing the frontier of generalist robots with advanced embodied reasoning, thinking, and motion transfer. *arXiv preprint arXiv:2510.03342*, 2025.

Sai Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. ChatGPT for robotics: Design principles and model abilities. *arXiv preprint arXiv:2306.17582*, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Jie Wang, Matthew Leonard, Kostas Daniilidis, Dinesh Jayaraman, and Edward S. Hu. Evaluating $\pi_0$ in the wild: Strengths, problems, and the future of generalist robot policies. *Online*, 2025a.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *International Conference on Machine Learning (ICML)*, 2024.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations (ICLR)*, 2025b.

World Agents. WorldDreamer. RoboCasa365 official leaderboard submission, 2026. URL `https://github.com/robocasa-benchmark/leaderboard/blob/main/submissions_md/worlddreamer_2026_06_20.md`. Submitting team: World Agents. Evaluated June 20, 2026; accessed September 25, 2026.

Jinming Wu, Zihao Deng, Wei Li, Yiding Liu, Bo You, Bo Li, Zejun Ma, and Ziwei Liu. Mmsearch-r1: Incentivizing lmms to search. *arXiv preprint arXiv:2506.20670*, 2025.

Wei Wu, Fan Lu, Yunnan Wang, Shuai Yang, Shi Liu, Fangjing Wang, Qian Zhu, He Sun, Yong Wang, Shuailei Ma, et al. A pragmatic VLA foundation model. *arXiv preprint arXiv:2601.18692*, 2026.

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

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

Chao Yu, Yuanqing Wang, Zhen Guo, Hao Lin, Si Xu, Hongzhi Zang, Quanlu Zhang, Yongji Wu, Chunyang Zhu, Junhao Hu, et al. Rlinf: Flexible and efficient large-scale reinforcement learning via macro-to-micro flow transformation. *arXiv preprint arXiv:2509.15965*, 2025.

Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, and Pete Florence. Socratic models: Composing zero-shot multimodal reasoning with language. In *International Conference on Learning Representations (ICLR)*, 2023.

Junyi Zhang, Jiaxin Ge, Hanjun Yoo, Letian Fu, Zihan Yang, Yaowei Liu, Raj Saravanan, Shaofeng Yin, Justin Yu, Dantong Niu, et al. Playful agentic robot learning. *arXiv preprint arXiv:2606.19419*, 2026.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Robotics: Science and Systems (RSS)*, 2023.

Haoyu Zhen, Xiaowen Qiu, Peihao Chen, Jincheng Yang, Xin Yan, Yilun Du, Yining Hong, and Chuang Gan. 3d-vla: A 3d vision-language-action generative world model. *arXiv preprint arXiv:2403.09631*, 2024.

Jiandong Zheng, Junfeng Li, Ziyang Wang, Dawei Liu, Xiang Kang, Yejin Feng, et al. X-VLA: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. *arXiv preprint arXiv:2510.10274*, 2025.

Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. Libero-pro: Towards robust and fair evaluation of vision-language-action models beyond memorization. *arXiv preprint arXiv:2510.03827*, 2025.

# A UNIFIED PRIMITIVE INTERFACE

Table 5: Primitive vocabulary. The same primitive names are used throughout the paper; RoboCasa365 additionally uses mobile-base primitives for kitchen-scale staging.

| Primitive | Type | Role |
|---|---|---|
| MOVE_TO | Composite | Move an end-effector to a world-frame Cartesian target using the environment's embedded solver. |
| MOVE_POSE | Composite | Move the end-effector while co-varying pose variables such as pitch for reach-limited configurations. |
| ROTATE_WRIST | Atomic | Apply a wrist-yaw set-point while holding the current spatial position. |
| ROTATE_PITCH | Atomic | Apply a wrist-pitch set-point while holding the current spatial position. |
| SET_GRIPPER | Atomic | Drive the gripper to an open or closed set-point for a fixed number of steps. |
| RELEASE | Atomic | Open the gripper under a release post-condition. |
| VLA_ACT | VLA | Execute a frozen VLA in short bursts for local contact-rich interaction. |
| NAVIGATE_TO | Composite (RoboCasa365) | Drive the mobile base to a world-frame location for kitchen-scale staging. |
| MOVE_BASE | Atomic (RoboCasa365) | Apply an open-loop local base-velocity set-point for fine repositioning. |

The primitive library $\mathcal{P}$ is the only action interface exposed to the planner. Each primitive is invoked by a single JSON object, executes inside the environment until an internal post-condition is reached, and then returns control together with a refreshed observation. Example JSON invocations are shown at the end of this subsection. The planner therefore never emits low-level torques, joint targets, or action chunks directly; it selects a primitive and binds its arguments from language, RGB-D observations, proprioception, and memory.

We organize $\mathcal{P}$ into two manipulation families. *Analytic primitives* are deterministic, model-based controllers specified from robot kinematics and require no training data. They split into *composite* primitives, which take a world-frame spatial goal and run an embedded solver to coordinate multiple degrees of freedom, and *atomic* primitives, which drive one intrinsic channel such as wrist orientation, gripper state, or base velocity to a parametric set-point. The *VLA primitive*, VLA_ACT, is a learned policy call that maps a prompt and live cameras to action chunks for local contact-rich behavior. The exploratory RESET utility is used only during bootstrapping and is not counted as a manipulation primitive.

Table 5 gives the primitive vocabulary used throughout the paper. The shared manipulation interface contains six analytic primitives and one VLA primitive; RoboCasa365 additionally uses two mobile-base primitives, NAVIGATE_TO and MOVE_BASE, for kitchen-scale staging. Details of RoboTwin bimanual execution are provided in Appendix J. Crucially, the primitive vocabulary is fixed before evaluation; the planner cannot invent new primitives at deployment time.

The compact JSON contract below illustrates the shared interface; Appendix J gives the benchmark-specific availability and implementation notes using the same primitive names.

```
{"action": "move_to",      "xyz": [<x>,<y>,<z>], ...}
{"action": "move_pose",    "xyz": [<x>,<y>,<z>], "pose": <orientation>,
 ...}
{"action": "rotate_wrist","target_yaw": <float>, ...}
{"action": "rotate_pitch","target_pitch": <float>, ...}
{"action": "set_gripper", "gripper": <open|close>, ...}
{"action": "release",      ...}
{"action": "navigate_to", "xy": [<x>,<y>], ...}
{"action": "move_base",    "forward": <float>, "lateral": <float>, "
turn": <float>, ...}
{"action": "vla_act",      "prompt": <str>, "max_chunks": <int>,
                           "stop": <predicate>}
```

**VLA-backed contact primitive.** VLA_ACT is the learned primitive for contact-rich interaction. Across benchmarks, VLA_ACT covers grasping, constrained placement, fixture actuation, button pressing, drawer or door manipulation, insertion, and embodiment-specific contact behaviors. The planner supplies a task-conditioned prompt and controls execution through an early-return predicate $\tau$ or a chunk budget. The frozen VLA $f_\theta$ then emits action chunks until $\tau$ is satisfied or the chunk budget is exhausted. This keeps the VLA as a local contact specialist while semantic grounding, spatial re-binding, navigation, re-staging, and long-horizon composition remain under planner control.

**Stopping criteria and feedback.** The agent can write code to check returned RGB-D observations and robot state between VLA calls, or select a bounded number of chunks, inspect the result, and decide whether to continue. In LIBERO, the illustrative `object_lifted` condition is a proprioceptive proxy: the end-effector must descend and then rise while the gripper opening lies within a specified range. The default descent and lift thresholds are $0.10$ and $0.05$ m, with gripper opening below $0.06$ m; these thresholds are configurable per call. This check does not query ground-truth object height. Non-grasp contact calls return on the chunk budget or official environment termination.

RoboCasa calls return on task success, the chunk budget, or a configurable settling criterion based on end-effector displacement and gripper-opening changes over consecutive chunks. The worker also reports grasp/contact diagnostics computed from simulator fingerpad–object contacts; these are separate from the stopping rule and are not RGB-D estimates. For online dual-Franka execution, phase boundaries use gripper transitions and end-effector motion or elapsed time, followed by visual inspection (Appendix G). Planner-side spatial grounding uses RGB-D rather than ground-truth object poses; this restriction is distinct from worker-side contact and benchmark-success checks.

## B Experimental Setup

We evaluate Harness VLA on four benchmark families. The tabletop suites are LIBERO (Liu et al., 2023b) and LIBERO-Pro (Zhou et al., 2025); the household and bimanual suites are RoboCasa365 (Nasiriany et al., 2026) and RoboTwin C2R (Mu et al., 2025), respectively. Benchmark-specific task splits and seed protocols are provided in Appendix K. Across all evaluations, the planner operates over the same frozen primitive vocabulary $\mathcal{P}$ (Appendix A) and is not allowed to introduce new primitives at deployment time. The VLA primitive is instantiated with benchmark-specific frozen policies while preserving a unified VLA_ACT interface: the RLinf-released `pi05_libero130_fullshot` $\pi_{0.5}$-SFT checkpoint, denoted $\pi_{\mathrm{RLinf}}$ (Yu et al., 2025), for LIBERO and LIBERO-Pro, the frozen RLDX-1 RoboCasa checkpoint (Kim et al., 2026) for RoboCasa365, and our post-trained LingBot-VLA checkpoint (Wu et al., 2026) for RoboTwin C2R. In the result tables, **Harness VLA (Codex)** and **Harness VLA (CC)** denote the same harness instantiated with Codex and Claude Code planners, respectively; **CC** abbreviates Claude Code. The $\pi_{\mathrm{RLinf}}$, RLDX-1, and LingBot-VLA rows serve as direct frozen-VLA baselines for their corresponding benchmarks. Appendix L details the VLA instantiations.

**Evaluation regimes.** In the few-shot regime, Harness VLA follows the memory-backed workflow in Section 2.2: the agent performs task-level bootstrapping on one reference seed, stores the successful primitive trace in Task Specific Memory, and re-grounds that trace under new seeds or perturbations. All spatial arguments are grounded from current observations. This setting tests whether the primitive organization discovered on the reference seed transfers to standard LIBERO, LIBERO-Pro instruction-redirection and position-swap perturbations, and RoboCasa365 kitchen tasks. In the zero-shot regime, the agent solves without retrieving Task Specific Memory or Global Memory for the target setting, testing online planner reasoning and the frozen primitive interface without task-specific harness memory.

## C Related Work

Harness VLA sits at the intersection of three lines of work: end-to-end robot foundation policies, multimodal LLM agents, and programmatic robot-control systems. We review these areas through the role each assigns to learned policies and explicit control. This perspective clarifies our position:

rather than fine-tuning a stronger VLA or expanding the primitive library, we study how a memory-guided agent can turn a frozen VLA into a controllable contact-rich primitive and compose it with fixed analytic controllers.

**VLA Models.** End-to-end Vision-Language-Action (VLA) models map natural-language instructions and visual observations directly to low-level robot actions by extending pretrained vision-language backbones with an action head. The generalist-policy line opened by RT-1 (Brohan et al., 2022) and Octo (Octo Model Team et al., 2023), scaled by RT-2 (Brohan et al., 2023) and the Open X-Embodiment release (O'Neill et al., 2024), established the training pattern of co-training a large VLM with cross-embodiment robot demonstrations. OpenVLA (Kim et al., 2024) brought this paradigm into the open by combining a Prismatic-style VLM (Karamcheti et al., 2024) with a Llama-2 action tokenizer, while the flow-matching $\pi_0$ (Black et al., 2025) and $\pi_{0.5}$ (Physical Intelligence et al., 2025) models reported substantial gains from co-training with heterogeneous data and out-of-distribution language. Recent large-scale systems such as GR00T (Bjorck et al., 2025) and Gemini Robotics (Gemini Robotics Team et al., 2025) continue to scale this pattern to humanoid and general-purpose embodiments, alongside related lines on 3D-aware VLAs (Zhen et al., 2024; Driess et al., 2023), VLM-based imitation (Li et al., 2024; Huang et al., 2024), and CLIP-conditioned controllers (Shridhar et al., 2022; Chi et al., 2023; Zhao et al., 2023). Empirically, however, these models exhibit a sharp asymmetry: they are strongest at contact-rich visuomotor phases—in particular for irregular grasping and fixture actuation that defeat analytic controllers—but degrade dramatically on instruction following, long-horizon composition, and out-of-distribution scenes (Kim et al., 2024; Physical Intelligence et al., 2025; Wang et al., 2025a). This asymmetry motivates a factorization in which the VLA is delegated planner-selected contact-rich operations while a higher-level controller assumes responsibility for language interpretation, target grounding, transport, posture adjustment, navigation, and release.

**LLM-driven Multimodal Agent.** Frontier multimodal large language models have rapidly closed the gap on dense perception, spatial reasoning, and long-horizon tool use. Recently, releases including GPT-5.2 (OpenAI, 2025), Gemini 3 (Pichai et al., 2025), Qwen3-VL (Bai et al., 2025), the Claude 4 family (Sonnet 4.5 and Opus 4.7 (Anthropic, 2025)), and Llama 4 (Meta, 2025) have demonstrated qualitatively stronger physical-scene grounding than the GPT-4o (OpenAI, 2024) / Gemini 1.5 (Gemini Team et al., 2023) generation, while open-weight models such as Molmo (Deitke et al., 2025) and Qwen3-VL (Bai et al., 2025) make these capabilities broadly available. Targeted spatial benchmarks (Chen et al., 2024a) and tool-augmented browsing agents (OpenAI, 2025; Gemini Team et al., 2023; Wu et al., 2025; Geng et al., 2025; Huang et al., 2026; Jin et al., 2025; Li et al., 2025) further show that, given a closed-loop interface, these backbones can sustain multi-hop perception and decision making over rich, partially observed environments. These advances make it increasingly viable to delegate semantic grounding and deterministic manipulation phases—language parsing, target localization, transport planning, posture adjustment, navigation, and release timing—to a frontier VLM at the top of the agent stack (Gemini Robotics Team et al., 2025; Team et al., 2025; Wang et al., 2023). We build on this premise but, rather than driving the robot end-to-end with the VLM, place it inside an agentic harness that emits structured primitive calls, observes execution feedback, and iterates—reserving direct action prediction for planner-selected contact-rich phases.

**Programmatic and Tool-Using Robot Agents.** Code-as-policies systems recast robot control as program synthesis: the model writes an executable program that coordinates perception and motion APIs, leveraging the LLM's compositional generalization while keeping low-level control deterministic. Beginning with Code-as-Policies (Liang et al., 2023), ProgPrompt (Singh et al., 2023), Instruct2Act (Huang et al., 2023a), and ChatGPT-for-Robotics (Vemprala et al., 2023), this paradigm has been extended with multimodal program synthesis (RoboCodeX (Mu et al., 2024), ViperGPT (Surís et al., 2023), VisProg (Gupta & Kembhavi, 2023)), 3D value-map generation (Huang et al., 2023b), VLM-supervised assembly (Goldberg et al., 2025), and long-horizon agentic frameworks (Li et al., 2026; Shi et al., 2025). Harness VLA shares this literature's goal of making language-model reasoning executable through explicit perception and control interfaces, but differs in the action representation: our agentic planner does not synthesize executable code or new control programs. It emits structured JSON primitive invocations inside a closed-loop harness, observes the execution outcome after each primitive, and re-binds the next primitive arguments from

current RGB-D evidence and memory. Recent work also studies how agents can grow their own reusable skill libraries: ASPIRE (Lu et al., 2026) uses fine-grained execution traces to diagnose failures, synthesize validated repairs, and admit the resulting localization, navigation, motion, grasping, and debugging patterns into a continually expanding skill library. This direction is complementary to ours: ASPIRE expands the agent's reusable skills, whereas Harness VLA deliberately keeps the primitive vocabulary fixed and studies how memory-guided composition can extend a frozen VLA without deployment-time primitive expansion. A second strand draws on software-engineering agents: executable code is empirically a strong action representation for LLM agents (Wang et al., 2024), and systems such as SWE-agent (Yang et al., 2024) and OpenHands (Wang et al., 2025b) formalize harnesses for iterative editing, execution, and feedback. We borrow the harness principle—structured interfaces, persistent state, execution feedback, and memory—rather than the requirement that actions be represented as code. Reliability is further improved by self-correction (Shinn et al., 2023; Madaan et al., 2023; Chen et al., 2024b) and persistent symbolic state across steps (Yoneda et al., 2024), while LLM-driven planners (Ahn et al., 2022; Huang et al., 2022; Mu et al., 2023; Zeng et al., 2023; Liu et al., 2023a; Chen et al., 2024c; Rana et al., 2023; Mandi et al., 2024) demonstrate that LLMs can sequence pretrained primitives or modules, ground 3D scenes, and recover from failures. Two limitations recur across this literature, however. First, task-specific execution traces are rarely represented as reusable, parameterized memory that can be grounded again under new spatial layouts. Second, failure knowledge is seldom distilled into a Global Memory that prevents the planner from repeating known empty grasps, false successes, or unstable staging choices. Voyager (Wang et al., 2023) showed that persistent memory can improve embodied agents in a digital sandbox, but this memory-centric design has not been combined with a VLA-backed contact-rich primitive for physical manipulation. Our framework couples the two: a frozen VLA serves as a contact-rich specialist invoked through a single primitive interface, successful primitive sequences are stored in **Task Specific Memory**, and reusable success rules and failure models are distilled into **Global Memory**. Together, these two design choices—VLA delegation for contact-rich operations, and memory-augmented primitive composition for everything else—let a single memory-guided agentic planner cover the full task distribution exposed by a given environment, including paraphrased and re-targeted natural-language instructions that defeat monolithic VLAs whose language channel is largely vestigial (Kim et al., 2024; Physical Intelligence et al., 2025).

## D STANDARD LIBERO RESULTS

**Standard LIBERO.** Table 6 reports results on the four standard LIBERO suites. Harness VLA (CC) achieves an aggregate success rate of $\mathbf{96.0}\%$ $(384/400)$, including $100.0\%$ on OBJECT and $93.0\%$ on LIBERO-10. Compared with the frozen $\pi_{\mathrm{RLinf}}$ checkpoint used inside VLA_ACT, which obtains $95.3\%$ overall, Harness VLA preserves competitive standard-suite performance while exposing the same policy through a controllable primitive interface for the perturbed evaluations in Section 3.1.

Table 6: Success rate (%) on standard LIBERO. $\pi_{\mathrm{RLinf}}$ (Yu et al., 2025) and Harness VLA (CC) are evaluated by us on 100 trials per suite (10 tasks $\times$ 10 seeds). Harness VLA uses the same RLinf-released `pi05_libero130_fullshot` $\pi_{0.5}$-SFT checkpoint inside VLA_ACT. Bold marks the best method in each suite or overall column.

| Method | Spatial | Object | Goal | LIBERO-10 | **Overall** |
|---|---|---|---|---|---|
| OpenVLA (Kim et al., 2024) | 84.7 | 88.4 | 79.2 | 53.7 | 76.5 |
| NORA (Hung et al., 2025) | 85.6 | 89.4 | 80.0 | 63.0 | 79.5 |
| $\pi_0$ (Black et al., 2025) | 96.8 | 98.8 | 95.8 | 85.2 | 94.2 |
| $\pi_{\mathrm{RLinf}}$ | **99.0** | 96.0 | 97.0 | 89.0 | 95.3 |
| AtomVLA (Sun et al., 2026) | 96.4 | 99.6 | **97.6** | 94.4 | **97.0** |
| **Harness VLA (CC)** | 97.0 | **100.0** | 94.0 | 93.0 | 96.0 |

## E ROBOTWIN C2R RESULTS AND ANALYSIS

**RoboTwin C2R.** RoboTwin C2R evaluates zero-shot clean-to-randomized transfer: the agent obtains a Task Specific Memory trace from the clean setting and transfers it directly to randomized task instances, with no randomized-setting bootstrapping, additional task-level exploration, or VLA

fine-tuning. The VLA_ACT backend here is LingBot-VLA, our RoboTwin-specialized VLA checkpoint; after post-training, it is frozen for both the direct VLA baseline and Harness VLA evaluation. Table 7 compares C2R success rates against representative VLA baselines. Direct LingBot-VLA reaches $50.4\%$, while Harness VLA raises the same frozen backend to $58.0\%$ with Codex and $58.4\%$ with CC. The table also reports external VLA baselines for context.

Table 7: RoboTwin C2R success rates (%). LingBot-VLA is our post-trained RoboTwin VLA checkpoint evaluated directly without agent-level decomposition, and is also the frozen VLA_ACT backend used by Harness VLA. Other VLA rows are representative external baselines. Harness VLA is evaluated on 50 tasks with 5 randomized seeds per task. Bold marks the best method.

| Method | Success (%) |
|---|---|
| GR00T-N1.7 (Bjorck et al., 2025) | 20.7 |
| $\pi_{0.5}$ (Physical Intelligence et al., 2025) | 47.9 |
| StarVLA (Community, 2026) | 10.6 |
| LingBot-VLA (Wu et al., 2026) | 50.4 |
| **Harness VLA (Codex)** | 58.0 |
| **Harness VLA (CC)** | **58.4** |

**Planner-staged invocation and task completion.** Figure 8 extends the invocation analysis of Section 3.2 to RoboTwin C2R. Success increases with a small number of planner-selected VLA calls and approaches the full harness result, while the direct frozen-policy baseline remains at $50.4\%$. The complete cross-benchmark attribution in Figure 9 shows that $66\%$ of successful RoboTwin rollouts finish after a VLA primitive and $34\%$ after an analytic primitive. These outcomes are consistent with the terminal contact-rich interactions required by bimanual tasks. Appendix N reports the corresponding primitive usage statistics, and Appendix K.4 details the clean-to-randomized protocol.

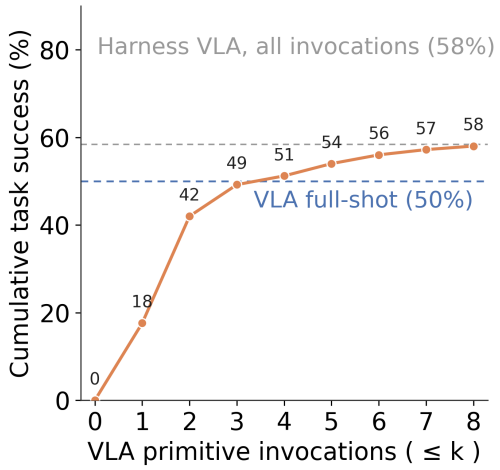



Figure 8: **Adaptive VLA invocation on RoboTwin C2R.** Cumulative task success as a function of the maximum number of VLA primitive invocations allowed per episode. The blue dashed line marks the direct frozen-policy baseline, and the gray dashed line marks full Harness VLA performance.

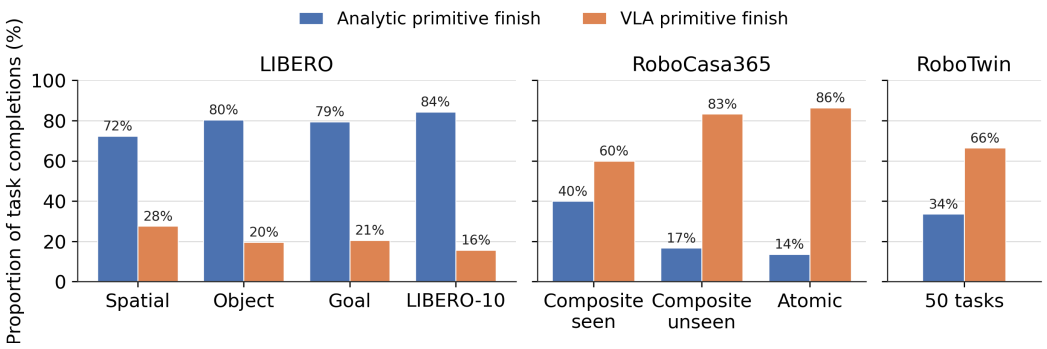



Figure 9: **Task completion attribution across benchmarks.** Bars show the fraction of successful rollouts whose final benchmark completion predicate fires after an analytic primitive (blue) or after a VLA primitive (orange). LIBERO Pro-family tasks are mostly finished by analytic primitives after the VLA has established stable contact, whereas RoboCasa365 and RoboTwin C2R contain more terminal contact-rich operations such as fixture actuation, constrained placement, or bimanual object interaction.

# F Additional Mechanism Analysis and Rollout Examples

**Analysis setting.** Unless otherwise noted, the analyses in Section 3.2 use Harness VLA (CC) as a representative instantiation, without loss of generality: Codex and CC share the same harness, memory interface, primitive library, frozen VLA interface, and evaluation protocols, differing only in the planner backbone. LIBERO-family analyses use the same $\pi_{\text{RLinf}}$ checkpoint as the main evaluation, while non-LIBERO analyses use their benchmark-specific frozen VLA primitive.

## F.1 Semantic Re-Grounding

The massive gap between Harness VLA and end-to-end VLAs in Table 1 is achieved without altering the visuomotor backbone. $\pi_{\text{RLinf}}$ already solves the standard variants of these tasks, but Figure 3 shows that its behavior is weakly conditioned on the task description and current scene binding. In the task-perturbed Object-Pro case, the visual scene remains similar while the instruction redirects the target, yet $\pi_{\text{RLinf}}$ repeats the standard behavior instead of following the new task description. In the swap-perturbed Goal-Pro case, the object layout changes, yet $\pi_{\text{RLinf}}$ still moves the object toward the training-time region. Harness VLA makes semantic grounding explicit at the planner level: the planner $\Pi$ parses the task description, resolves the current contact target from the live RGB-D observation, uses analytic primitives for staging and repositioning, and invokes or re-invokes `VLA_ACT` only for the local contact-rich phase (Key Finding 2). Thus, semantic and scene-level reasoning is handled by the planner, while the frozen VLA remains responsible only for executing the contact-rich operation under the planner-provided binding.

## F.2 Planner-Staged Invocation and Retry

In Harness VLA, the planner does not call the VLA as a one-shot black box. Although the VLA is invoked sparsely rather than as a continuous controller, each call is a planner-chosen local contact-rich attempt whose staging can determine whether the frozen policy succeeds. The planner $\Pi$ therefore treats `VLA_ACT` as a local contact-rich primitive whose invocation can be re-staged and retried. Given a desired contact target (the object, fixture, or local interaction region to be acted on), the planner uses analytic primitives to place the robot in a feasible pre-contact configuration, invokes the VLA, observes the resulting contact state, and decides whether the next step should continue the task or reframe the local attempt.

**First, staging restores a VLA-compatible local state.** Under deployment perturbations such as semantic retargeting and spatial-layout shifts, the original VLA viewpoint or pre-contact pose may no longer expose the correct contact target in a familiar configuration. By re-staging the robot around the current scene, the agentic planner $\Pi$ brings the target back into a VLA-compatible local obser-

vation while preserving the correct semantic binding. This mechanism explains why the harness can improve a frozen VLA without changing its parameters: it learns where the VLA should begin acting, rather than asking the policy to absorb the full distribution shift by itself.

**Second, retry localizes contact failures.** Because VLA execution is stochastic and short-horizon contact is physically brittle, a single failed attempt need not terminate the entire rollout. Harness VLA localizes such errors to the current contact-rich subtask: the planner can observe an incomplete or unstable outcome, re-stage the robot, and re-invoke VLA_ACT, rather than letting a transient failure propagate through a monolithic long-horizon policy. Thus, repeated VLA calls are not continuous low-level control; they are sparse, planner-selected attempts that make contact-rich execution recoverable.

**Planner-staged invocation and retry.** Figure 10 gives representative case studies for Key Finding 2 in Section 3.2: the planner observes an incomplete or unstable contact outcome, re-stages the robot or base, and calls VLA_ACT again for the next local contact attempt.

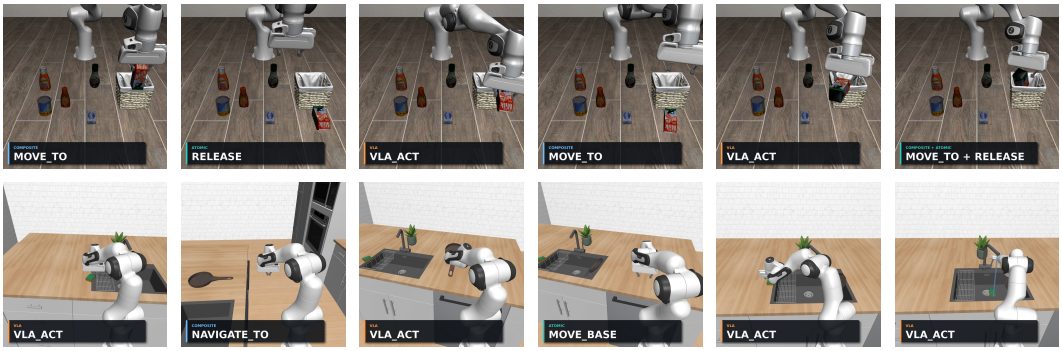



Figure 10: **Representative rollout frames for adaptive VLA invocation.** Top row: a Harness VLA rollout on LIBERO-PRO OBJECT task 4. The planner repeatedly invokes VLA_ACT around the milk carton after intermediate grasping or placement attempts leave the object outside or only partially inside the basket; after re-staging the end-effector and retrying the local contact-rich operation, the milk carton is finally placed stably inside the basket. Bottom row: a RoboCasa365 PRESOAKPAN rollout. The planner adjusts the mobile base and arm pose around the pan, retries VLA_ACT until a stable grasp is obtained, places the pan into the sink, and later invokes VLA_ACT again to actuate the faucet. These examples show that repeated VLA calls are not continuous control, but planner-selected contact attempts embedded inside analytic navigation, staging, and verification.

### F.3 ANALYTIC DECOMPOSITION AROUND CONTACT-RICH PHASES

Analytic primitives do not replace the VLA on contact-rich operations. Instead, they handle the surrounding non-contact structure of the task: free-space transport, pre-contact staging, wrist or base reorientation, retreat, and post-contact repositioning. This lets the planner reserve VLA_ACT for the local contact-rich phases where learned visuomotor control is needed, including grasping, constrained placement, button pressing, faucet turning, drawer manipulation, and coffee-machine operation.

Once the robot has established stable contact with the target, the planner $\Pi$ can use analytic primitives to move, rotate, or navigate the robot toward the next relevant region, while invoking the VLA again when the next contact-rich phase begins. Appendix N reports the primitive usage statistics. Thus, the analytic vocabulary does not solve contact-rich manipulation by itself; it expands the conditions under which the same frozen VLA can be reused. By handling the non-contact context around each local interaction, the planner prevents the VLA from being responsible for long-horizon composition, scene-level grounding, and every intermediate motion in the rollout.

Figure 5 provides the aggregate attribution for this division of labor by separating successful rollouts according to the primitive class that triggers the final benchmark completion predicate. LIBERO Pro-family tasks are usually completed after analytic transport, release, or repositioning once contact

has been established. In RoboCasa365, the final predicate often depends directly on a contact-rich operation, so successful rollouts more frequently finish inside the VLA primitive.

**Analytic decomposition and contact recovery.** Figure 11 gives representative examples for Key Finding 3 in Section 3.2: analytic primitives localize execution, expose failed or incomplete contacts, and move the robot back into a configuration where VLA_ACT can be invoked again.




Figure 11: **Representative rollout frames for analytic decomposition around contact-rich phases.** Top row: on a LIBERO-10-PRO swap task, the agent first invokes VLA_ACT and starts moving toward the basket, then detects during MOVE_TO that the VLA has not actually grasped the cream-cheese box. The planner moves back, retries VLA_ACT, and, after a successful grasp, completes the subtask with MOVE_TO and RELEASE. Bottom row: on the RoboCasa STEAMINMICROWAVE composite-seen task, the agent successfully grasps the bowl with VLA_ACT, searches and repositions until the microwave is localized, invokes VLA_ACT to place the bowl inside, pushes it in with MOVE_TO, closes the door, and finally uses MOVE_TO and NAVIGATE_TO to press the switch.

## G REAL-WORLD DEPLOYMENT DETAILS

**Hardware and execution.** We demonstrate online and Flash execution on two dual-Franka installations. In the online setup, base and wrist cameras provide policy observations, and a calibrated RealSense D455 supplies metric localization. Codex with GPT-5.5 invokes a frozen $\pi_{0.5}$ desk-clearing policy for grasping, handoff, and placement, using phase-specific stopping conditions. Following Appendix A, analytic primitives handle pre-contact staging, free-space transport, and reorientation, while VLA_ACT executes local contact-rich phases. This division reuses the same policy without fine-tuning.

**Reset and reward.** Real-world resets and success judgments require a human in the loop. To reset the scene, an operator restores the initial object arrangement and confirms that a new attempt can begin. Objects are not reset within an evaluation attempt. Task success is recorded only after an operator inspects the current scene and confirms that the task goal has been achieved. The remaining execution and memory workflow follows Section 2.

**Qualitative comparisons.** Figure 6 compares online Harness VLA with fullshot execution of the frozen desk-clearing policy. The top row shows the learned grasp–handoff–basket-placement routine. The remaining rows show Harness VLA redirecting bowls to a box, transferring plates in a specified order, assigning bowls to destinations based on a visible dirty marker, and recovering from a missed grasp.

Figure 7 compares fullshot VLA execution with two Flash procedures. The fullshot baseline grasps a container, hands it off, and places it in the bin. The Flash demonstrations instead pour steel beads into a receiving bowl and wipe a held plate with a sponge before stowing the objects. Both procedures execute from task cards without online planning.

Together, these cases illustrate the mechanisms analyzed in Section 3.2. Online planning separates destination selection and action order from the learned routine, while local retry enables recovery from a missed grasp (Key Findings 1 and 2). Flash Mode illustrates how analytic transport and

reorientation compose the same contact skills into new bimanual procedures (Key Finding 3). The capability extension comes from semantic re-grounding and primitive composition, with the VLA weights unchanged. These recordings provide qualitative demonstrations rather than comparative success-rate estimates.

## H Conclusion and Limitations

We introduced Harness VLA, an asymmetric hierarchical framework that casts a frozen VLA as a single contact-rich primitive interface within an LLM-driven agent, delegating transport, posture, navigation, and release phases to the planner. From the perspective of agent harness engineering, Harness VLA shows that reliable manipulation can come not only from training a stronger policy, but also from surrounding a frozen policy with an auditable execution loop, fixed primitive contracts, memory, feedback, and task-level verification. Evaluations across standard and heavily perturbed benchmarks confirm that Harness VLA achieves state-of-the-art robustness. These results demonstrate that pretrained VLAs are most effective when isolated to contact-rich visuomotor control; abstracting semantic and spatial bindings away from the VLA prevents the catastrophic failures frequently observed in monolithic deployments.

**Limitations and future work.** Our current framework is limited by an open feedback loop between the high-level planner and low-level VLA. Additionally, the system lacks joint fine-tuning via environmental rewards and human preferences—an issue necessitating future sample-efficient reinforcement learning (e.g., GRPO). Finally, the absence of fine-grained image captioning constrains structural reasoning in highly cluttered, long-horizon tasks. A complementary future direction is to combine our fixed-vocabulary composition strategy with automatic skill-discovery systems such as ASPIRE (Lu et al., 2026): when repeated primitive compositions reveal a missing abstraction, an agent could propose, validate, and admit a new reusable skill while retaining the auditable primitive interface and VLA-backed contact specialization studied here.

## I File-Mediated REPL Protocol

The harness of Section 2.2 implements the execution loop of Section 2.1 as a synchronous file-mediated Read-Eval-Print Loop (REPL). A long-running environment worker owns the live simulator state, while the planner $\Pi$ interacts with it only through serialized primitive invocations and persisted observations. The planner does not directly query ground-truth object poses or controller internals. Worker-side contact diagnostics and stopping checks are described in Appendix A.

At turn $t$, the planner reads the current observation $o_t$, the task language $\ell$, and retrieved context from Task Specific Memory and Global Memory. It then emits one primitive invocation $c_t \in \mathcal{P}$ by writing a JSON object to `command.json`. The object contains the primitive name in its `action` field and the corresponding keyword arguments. The worker consumes this file, executes the selected primitive in the live environment, and writes the next indexed observation $o_{t+1}$ together with lightweight execution records. The planner waits for these files before selecting the next primitive. Thus, each physical action is followed by observation and diagnosis before the rollout continues. Table 8 lists the files exchanged by the REPL.

The index `NN` increases monotonically. The initial observation is written at `NN=00`; each executed primitive produces the next indexed state, perception files, and diagnostic log. These records make the rollout auditable without exposing oracle object coordinates to the planner.

**Task Specific Memory.** Task Specific Memory stores the reusable structure of a solved task instance. It contains a procedural JSONL trace and a semantic JSON summary. The trace records what primitive invocations were issued; the summary records why the strategy worked and what should be avoided. A simplified summary is:

```
  {"task":"put the black bowl on the wooden tray",
   "success":true,
   "trace_file":"task_specific_memory_put_black_bowl_on_tray_s0.jsonl
",
   "strategy":"use VLA for grasping, then analytic transport and
release",
```

Table 8: Files used by the file-mediated REPL. The main text abstracts the observation as RGB-D and robot state; the appendix also lists diagnostic records used for synchronization and auditability.

| File or artifact | Role |
|---|---|
| `command.json` | Planner-issued primitive invocation $c_t$. |
| `state_NN.json` | Step-indexed task language, robot proprioception, and benchmark success signal. |
| RGB-D / world-map files | Benchmark-specific perceptual evidence for semantic identification and metric re-grounding. |
| `log_NN.json` | Diagnostic record containing the accepted command, primitive status, step counts, and failure information when available. |
| `done_NN.flag` or terminal file | Synchronization signal indicating that the worker has finished the current primitive. |
| Task Specific Memory trace | Append-only JSONL procedural memory for one task. Each line is one primitive command. |
| Task Specific Memory summary | JSON semantic memory summarizing the outcome, strategy, recovery decisions, and failure modes. |
| Global Memory | Cross-task success rules and failure models for using the primitive library. |

```
  "avoid":["do not reuse reference xyz values",
           "verify placement with the benchmark success signal"]}
```

The paired procedural trace stores the primitive order:

```
{"action":"vla_act","prompt":"grasp the black bowl","max_chunks":2}
{"action":"move_to","xyz":[0.12,-0.08,0.92],"gripper":null}
{"action":"release"}
```

The trace is a task-level solution skeleton, not an open-loop trajectory. It records the ordering of analytic and VLA-backed primitives, the placement of VLA invocations, and the transition points between contact-rich execution, transport, release, and verification. Spatial arguments in the stored trace are treated as reference-scene bindings. At deployment time, the planner reuses the memory structure but re-grounds objects, fixtures, support surfaces, and target poses from the current observation.

**Global Memory.** Global Memory stores task-independent operating knowledge for the primitive library. diagnose before retrying. A compact example is:

```
  Success rule:
  Use VLA primitives for contact-rich phases such as irregular
grasping
  or fixture interaction. After a stable grasp, prefer analytic motion
  for long transport and precise placement.

  Failure model:
  If the gripper closes but the object does not move with the end
effector,
  treat the attempt as an empty grasp. Re-localize the object and re-
stage
  before retrying.

  Failure model:
  Do not terminate from visual proximity alone. Check the benchmark
success
  signal and the latest execution record.
```

**Iterative memory construction.** Memory is constructed during interaction rather than written only after the rollout. After each primitive, the planner reads the new observation and diagnostic record, then classifies the outcome as progress, recoverable failure, or unrecoverable failure. Successful rollouts are stored as Task Specific Memory. Recoverable failures remain in the trace and are explained in the semantic summary, so that subsequent steps document the correction. Failed attempts are also retained as negative evidence and may contribute failure models to Global Memory.

Across attempts, the memory is refined rather than simply accumulated. A later attempt can replace the procedural trace if it yields a shorter or more reliable solution, while earlier failure observations remain useful as constraints on future planning. This separation lets Harness VLA transfer how a task should be solved without replaying where objects happened to be in the reference scene.

## J Primitive Vocabulary and Environment-Specific Extensions

This appendix expands the primitive vocabulary of Appendix A. We use the same primitive names across all benchmarks. Differences across environments are expressed only through availability, arm binding, and implementation backend; they are not treated as new primitive names unless the embodiment exposes a new degree of freedom.

Table 9: Primitive availability across the three benchmark embodiments. The exploratory RESET utility supports bootstrapping and is not counted as a manipulation primitive.

| Primitive | LIBERO | RoboCasa365 | RoboTwin C2R |
|---|---|---|---|
| MOVE_TO | yes | yes | yes |
| MOVE_POSE | yes | via composition | – |
| ROTATE_WRIST | yes | – | yes |
| ROTATE_PITCH | yes | yes | – |
| SET_GRIPPER | yes | yes | yes |
| RELEASE | yes | yes | yes |
| VLA_ACT | yes | yes | yes |
| NAVIGATE_TO | – | yes | – |
| MOVE_BASE | – | yes | – |
| `arm` binding | – | – | yes |

**Universal analytic primitives.** MOVE_TO is the shared end-effector transport primitive: it takes a world-frame Cartesian target and delegates to the solver available in the current environment. The internal backend may be an operational-space servo, a Jacobian-based controller, or an IK planner, but the exposed primitive semantics are identical. MOVE_POSE extends MOVE_TO by co-varying position with an orientation component such as pitch; when an environment does not expose it directly, the same behavior is expressed as a short composition of ROTATE_PITCH and MOVE_TO. ROTATE_WRIST and ROTATE_PITCH apply yaw and pitch set-points while holding the current spatial position. SET_GRIPPER drives the gripper to an open or closed state, and RELEASE is the corresponding open-gripper primitive with a release post-condition. Environment-specific gripper conventions are hidden behind the primitive interface.

**Mobile-base and bimanual details.** RoboCasa365 adds two mobile-base primitives because kitchen-scale tasks require staging outside a fixed-arm workspace. NAVIGATE_TO is a composite primitive that drives the base toward a world-frame planar goal, while MOVE_BASE is an atomic primitive that applies a local base-velocity set-point for fine repositioning. RoboTwin C2R adds no new manipulation primitive name; instead, each primitive can be bound to the left arm, right arm, or a bimanual task pattern through the `arm` argument. Handover-style tasks are therefore represented as compositions of VLA_ACT, analytic transport, and RELEASE under this dual-arm binding rather than as a separate primitive.

**VLA_ACT.** VLA_ACT is the single learned primitive in the vocabulary. It binds to the frozen VLA used by the current benchmark and executes action chunks conditioned on a prompt and live observations. The planner selects backend-specific stopping parameters and a chunk budget; Appendix A

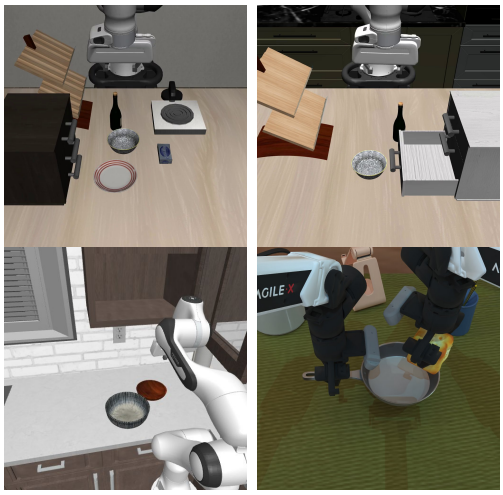


Figure 12: Overview of representative environments across the four benchmark families used in our evaluation. Each benchmark captures a distinct manipulation setting: structured tabletop manipulation (LIBERO), robustness under distribution shift (LIBERO-Pro), long-horizon kitchen manipulation (RoboCasa365), and bimanual manipulation under clean-to-randomized settings (RoboTwin C2R).

distinguishes stopping criteria from contact diagnostics and describes their information sources. The same primitive therefore covers grasping, placement, fixture actuation, insertion, and bimanual contact while preserving the planner's responsibility for semantic grounding, spatial re-binding, navigation, and re-staging.

**JSON invocation examples.** All primitives share a compact JSON command format. Representative calls are:

```
{"action": "move_to",      "xyz": [-0.101, 0.202, 1.05],
 "arm": "auto", "gripper": "open", "tol": 0.012, "max_steps": 80}
{"action": "navigate_to", "xy": [1.20, -0.35], "tol": 0.05}
{"action": "move_base",    "forward": 0.10, "lateral": 0.00,
 "turn": -0.15, "steps": 12}
{"action": "vla_act",      "prompt": "grasp the black bowl",
 "arm": "auto", "max_chunks": 30, "stop": "object_lifted"}
```

The exact numerical tolerances and stop predicates are benchmark-specific, but the planner always interacts through these primitive names.

## K Details about the Evaluation Benchmark

This appendix records the benchmark composition, task splits, rollout protocol, and success criteria used in our evaluation. Figure 12 shows the four benchmark families. We evaluate on four benchmark families: LIBERO, LIBERO-Pro, RoboCasa365, and RoboTwin C2R. LIBERO, LIBERO-Pro, and RoboCasa365 use a few-shot protocol in which seed $s_0$ (seed 0) for each task serves only as the exploratory reference seed for Task Specific Memory construction. On this seed, the agent searches for a successful primitive sequence and stores the resulting audit summary and JSONL command trace as Task Specific Memory. Seed $s_0$ is not counted in reported evaluation. Reported evaluation rollouts are run on held-out seeds that retrieve and re-ground the corresponding Task Specific Memory under new initial states. RoboTwin C2R uses a separate zero-shot clean-to-randomized protocol.

Across all benchmarks, task success is determined by the benchmark-provided binary completion predicate. A rollout is counted as successful if the task completion predicate is satisfied before the episode horizon or maximum step budget is exhausted. Primitive-level post-conditions, such as

the return condition of VLA_ACT or RELEASE, only determine when an individual primitive returns control to the planner; they are not used as substitutes for the final task success predicate.

### K.1 LIBERO Evaluation Benchmark

LIBERO (Liu et al., 2023b) is a language-conditioned manipulation benchmark organized into multiple task suites. We evaluate on four standard suites: LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-10. LIBERO-Spatial contains tasks that vary spatial relations, LIBERO-Object varies the target object identity, LIBERO-Goal varies the goal predicate under related scenes, and LIBERO-10 contains longer-horizon compositional manipulation tasks.

Each suite contains 10 language-conditioned tasks. For each task, seed $s_0$ (seed 0) is used only to explore the task and construct Task Specific Memory. Reported evaluation uses ten held-out seeds, denoted $s_1$–$s_{10}$, which retrieve this Task Specific Memory and ground it under new initial states. Thus, each LIBERO suite contains 10 tasks $\times$ 10 evaluation seeds $=$ 100 reported rollouts, and the four suites contain 400 reported rollouts in total (Table 10).

Table 10: LIBERO evaluation protocol.

| Suite | Tasks | Eval seeds per task | Reported rollouts |
|---|---|---|---|
| LIBERO-Spatial | 10 | 10 | 100 |
| LIBERO-Object | 10 | 10 | 100 |
| LIBERO-Goal | 10 | 10 | 100 |
| LIBERO-10 | 10 | 10 | 100 |
| Total | 40 | – | 400 |

Success rates are computed using the predicate-based rule defined at the beginning of this appendix.

### K.2 LIBERO-Pro Evaluation Benchmark

LIBERO-Pro (Zhou et al., 2025) extends the LIBERO task families with controlled perturbations. We evaluate four task families: Spatial, Object, Goal, and LIBERO-10. Each task family is evaluated under two perturbation settings, denoted **T** and **S**. **T** refers to the task or instruction-redirection setting, where the instruction is redirected to another valid target object or goal condition. **S** refers to the swap or position-swap setting, where object initial positions are swapped or rearranged while the instruction remains fixed.

We evaluate eight LIBERO-Pro cells: Spatial-T, Spatial-S, Object-T, Object-S, Goal-T, Goal-S, LIBERO-10-T, and LIBERO-10-S. Each cell contains 10 tasks. As in LIBERO, seed $s_0$ (seed 0) is used only to explore the task and construct Task Specific Memory; reported evaluation uses seeds $s_1$–$s_{10}$, which retrieve the stored Task Specific Memory and ground it under new initial states. Each cell therefore contains 10 tasks $\times$ 10 evaluation seeds $=$ 100 reported rollouts, for a total of 800 reported rollouts.

Table 11: LIBERO-Pro evaluation protocol.

| Evaluation cell | Tasks | Eval seeds per task | Reported rollouts |
|---|---|---|---|
| Spatial-T | 10 | 10 | 100 |
| Spatial-S | 10 | 10 | 100 |
| Object-T | 10 | 10 | 100 |
| Object-S | 10 | 10 | 100 |
| Goal-T | 10 | 10 | 100 |
| Goal-S | 10 | 10 | 100 |
| LIBERO-10-T | 10 | 10 | 100 |
| LIBERO-10-S | 10 | 10 | 100 |
| Total | 80 | – | 800 |

Table 11 summarizes the evaluation matrix. Success rates are computed using the predicate-based rule defined at the beginning of this appendix.

### K.3 RoboCasa Evaluation Benchmark

RoboCasa365 extends the evaluation to kitchen household manipulation. We use the RoboCasa365 `target50` split, which consists of three task groups: Atomic-Seen, Composite-Seen, and Composite-Unseen. Atomic-Seen contains 18 atomic tasks corresponding to short-horizon kitchen operations. Composite-Seen contains 16 composite tasks whose templates are also present in the pretraining set. Composite-Unseen contains 16 composite tasks whose templates are held out from pretraining and appear only in target evaluation. Here, "seen" and "unseen" refer to whether the task template appears in the pretraining set, not whether the exact episode, trajectory, or scene has been observed.

RoboCasa365 uses five held-out evaluation seeds, $s_1$–$s_5$, for every task in all three splits. Seed $s_0$ (seed 0) is used only for exploration and Task Specific Memory construction and is excluded from evaluation. This gives $18 \times 5 = 90$ rollouts for Atomic-Seen and $16 \times 5 = 80$ for each composite split, totaling $250$ rollouts over $50$ tasks (Table 12). All three planners, GPT-5.5, Opus-4.7, and GPT-6 Astra, use this protocol. Overall success gives equal weight to each task and, since all tasks have five evaluation seeds, equals the success rate pooled over all $250$ rollouts.

Table 12: RoboCasa365 evaluation protocol.

| Split | Tasks | Eval seeds per task | Reported rollouts |
|---|---|---|---|
| Atomic-Seen | 18 | 5 | 90 |
| Composite-Seen | 16 | 5 | 80 |
| Composite-Unseen | 16 | 5 | 80 |
| Total | 50 | 5 | 250 |

Success rates are computed using the predicate-based rule defined at the beginning of this appendix.

### K.4 RoboTwin Clean-to-Randomized Evaluation Benchmark

RoboTwin C2R is a bimanual manipulation benchmark with 50 tasks. The task set covers pick-and-place, stacking, ordering, handover, dual-arm transport, articulated-object interaction, pressing and clicking, rotation, scanning, and container-placement behaviors.

RoboTwin C2R uses a separate Clean-to-Randomized protocol. For each task, the Task Specific Memory trace is obtained from one official scripted-expert-verified seed in the `demo_clean` setting. Evaluation is then performed directly in the official `demo_randomized` setting on five scripted-expert-verified randomized seeds. The expert verification step is used only to ensure that the sampled task instances are feasible under the official task definition; it is independent of our method and does not use Harness VLA rollouts for seed selection. No additional trace search, fine-tuning, or task-level adaptation is performed in the randomized setting. This protocol evaluates zero-shot transfer from a clean-setting trace to randomized task instances.

We evaluate all 50 RoboTwin C2R tasks. Each task is evaluated on 5 seeds in the `demo_randomized` setting, resulting in $50 \times 5 = 250$ reported rollouts (Table 13).

Table 13: RoboTwin clean-to-randomized evaluation protocol.

| Protocol | Tasks | Eval seeds per task | Reported rollouts |
|---|---|---|---|
| RoboTwin C2R | 50 | 5 | 250 |

Success rates are computed using the predicate-based rule defined at the beginning of this appendix, with the completion predicate provided by the official RoboTwin C2R task-specific evaluator.

### K.5 BENCHMARK SUMMARY

Table 14 summarizes the scale of each evaluation benchmark. LIBERO and LIBERO-Pro use seed $s_0$ (seed 0) only to construct Task Specific Memory and report evaluation on ten held-out seeds $s_1$–$s_{10}$ per task. RoboCasa365 uses the same reference-seed convention and evaluates every task on five held-out seeds $s_1$–$s_5$ across all three splits. RoboTwin C2R uses clean-to-randomized evaluation where the Task Specific Memory is obtained from one expert-verified `demo_clean` seed and evaluated on expert-verified `demo_randomized` seeds.

Table 14: Summary of evaluation benchmarks.

| Benchmark | Tasks | Trials/task | Reported rollouts |
|---|---|---|---|
| LIBERO | 40 | 10 | 400 |
| LIBERO-Pro | 80 | 10 | 800 |
| RoboCasa365 | 50 | 5 | 250 |
| RoboTwin C2R | 50 | 5 | 250 |

## L VLA MODEL INSTANTIATIONS

We instantiate different vision-language-action models across benchmarks, which are uniformly abstracted as the single contact-rich primitive VLA_ACT within the Harness VLA framework.

### L.1 $\pi_{\mathrm{RLinf}}$: RLINF-RELEASED LIBERO CHECKPOINT

For LIBERO and LIBERO-Pro, we use the RLinf-released `pi05_libero130_fullshot` checkpoint, denoted $\pi_{\mathrm{RLinf}}$, as the frozen vision-language-action policy. It is based on the $\pi_{0.5}$ architecture, and we directly adopt this official $\pi_{0.5}$-SFT checkpoint as a frozen VLA_ACT contact-rich execution primitive within the Harness VLA framework.

**Architecture.** $\pi_{\mathrm{RLinf}}$ follows the $\pi_{0.5}$ vision-language-action architecture, which encodes multimodal inputs including visual observations $I_t$, language instructions $\ell$, and robot state $q_t$ into a unified transformer representation. The model is initialized from a pretrained vision-language backbone and aligned to robot action spaces via supervised learning.

Consistent with $\pi_{0.5}$, $\pi_{\mathrm{RLinf}}$ supports hierarchical inference, where high-level semantic subtask prediction and low-level action generation are jointly modeled within a single policy.

**Action Modeling.** $\pi_{\mathrm{RLinf}}$ adopts the two-stage inference paradigm introduced in $\pi_{0.5}$. Given an observation and language instruction, the model first predicts a high-level subtask $\hat{\ell}$ (e.g., "pick up the plate"), which is then used to condition low-level action generation.

The low-level policy produces continuous action chunks $a_{t:t+H}$, represented either via FAST tokenization or flow-based continuous modeling, enabling stable contact-rich manipulation.

**Training.** The model is supervised fine-tuned on the LIBERO-130 dataset following the $\pi_{0.5}$ training protocol, resulting in the official $\pi_{0.5}$-SFT checkpoint. In this work, no additional training or adaptation is performed, and the model is used in a fully frozen manner during evaluation.

**Performance.** On the LIBERO benchmark, $\pi_{\mathrm{RLinf}}$ achieves a success rate of 95.3%, demonstrating strong in-distribution manipulation capability. On LIBERO-Pro, which introduces instruction perturbations and compositional variations, performance drops to 50.0%, indicating sensitivity to distribution shifts.

**Role in This Work.** In this paper, $\pi_{\mathrm{RLinf}}$ is used as a frozen low-level execution module within the Harness VLA framework, serving as the contact-rich manipulation primitive for LIBERO and LIBERO-Pro tasks.

### L.2 RLDX-1

RLDX-1 is used in RoboCasa365 for kitchen manipulation tasks. It is a large-scale vision-language-action (VLA) foundation model designed for general dexterous manipulation across diverse robotic embodiments. In this work, we directly use the official RLDX-1 checkpoint and keep the model fully frozen during evaluation, treating it as a VLA_ACT contact-rich execution primitive within the Harness VLA framework.

**Architecture.** RLDX-1 adopts a Multi-Stream Action Transformer (MSAT) as its core action modeling architecture. The system first encodes multi-frame video observations and language instructions using a Vision-Language Model (VLM) based on Qwen3-VL 8B, and extracts action-relevant representations via cognition tokens.

A memory module is further introduced to aggregate historical cognition features, producing history-aware representations. The action model builds on MSAT, which decouples cognition and action streams and optionally introduces a physics stream when physical signals are available. These streams are jointly modeled via cross-stream self-attention, enabling unified processing of vision, language, state, and physical signals.

**Action Modeling.** RLDX-1 is trained using a flow-matching diffusion transformer for continuous action prediction. The model learns a velocity field that maps noisy action trajectories to clean action sequences, and generates future actions via iterative denoising.

During inference, the model produces action chunks in a chunk-wise manner and executes partial chunks sequentially to enable stable closed-loop control. The model also jointly models physical signals when available, improving contact-rich manipulation capability.

**Training.** This work directly uses the official RLDX-1 checkpoint and keeps all parameters frozen during evaluation, without any additional training or fine-tuning. The model has already undergone multi-stage training in its original pipeline and is used here as a unified execution policy.

**Performance.** On the RoboCasa365 benchmark, RLDX-1 achieves 60.0% on Atomic-Seen tasks, 21.3% on Composite-Seen tasks, and 5.0% on Composite-Unseen tasks, with an overall weighted success rate of 30.0%. These results show strong performance on atomic contact-rich manipulation tasks, while performance significantly degrades on compositional and out-of-distribution settings.

**Role in This Work.** In this paper, RLDX-1 is used as a frozen low-level execution module within the Harness VLA framework, serving as the contact-rich manipulation primitive for RoboCasa365 kitchen manipulation.

### L.3 LingBot-VLA

LingBot-VLA (Wu et al., 2026) is the vision-language-action model behind our RoboTwin backend for bimanual manipulation. It is a large-scale VLA foundation model designed for continuous robotic control across diverse real-world embodiments. In this work, we use our RoboTwin-post-trained LingBot-VLA checkpoint as a frozen low-level execution module within the Harness VLA framework.

**Architecture.** The model is built upon a pre-trained Qwen2.5-VL vision-language backbone and is extended with a Mixture-of-Transformers (MoT) architecture that separates vision-language reasoning and action generation into dedicated transformer pathways. These pathways are coupled via shared self-attention, enabling unified multimodal sequence modeling while mitigating cross-modal interference. An action expert module is introduced to predict continuous control signals conditioned on multimodal embeddings.

**Action Modeling.** LingBot-VLA adopts a flow-matching formulation for continuous action prediction. To improve temporal consistency in long-horizon manipulation, it employs chunked action decoding, where a fixed-length action sequence is predicted autoregressively in a single forward pass. The chunk size is set to $T = 50$, enabling stable and temporally coherent control.

**Training.** The model is first pre-trained on large-scale real-world dual-arm teleoperation data collected across 9 robotic embodiments, providing broad cross-embodiment generalization. It is then further adapted via supervised fine-tuning (SFT) on RoboTwin manipulation trajectories to specialize in bimanual manipulation tasks. After this post-training stage, the checkpoint is kept frozen in all direct VLA and Harness VLA evaluations.

**Training Configuration.** We summarize the key hyperparameters governing post-training in Table 15. These parameters correspond to the configuration used for all LingBot-VLA post-training experiments reported in this work.

Table 15: Post-training configuration of LingBot-VLA on RoboTwin.

| **Category** | **Configuration** |
|---|---|
| *Optimization* | |
| Optimizer | AdamW |
| Learning rate | $1 \times 10^{-4}$ |
| Vision encoder LR | $1 \times 10^{-6}$ |
| Weight decay | 0 |
| Loss function | L1 Flow Matching (L1_FM) |
| *Sequence Modeling* | |
| Chunk size | 50 |
| Max sequence length | 2048 |
| Flow steps | 10 |
| Max action dimension | 75 |
| Max state dimension | 75 |
| *Training Setup* | |
| Global batch size | 256 |
| Image resolution | $224 \times 224$ |
| Camera views | top + wrist left + wrist right |
| *System* | |
| Precision | mixed precision (bf16/fp32) |
| Distributed training | FSDP2 |

**Performance.** Under the RoboTwin randomized evaluation setting, LingBot-VLA achieves a 50.4% success rate in the direct frozen-agent configuration (i.e., as a standalone policy without agent-level decomposition or external planning). This direct baseline differs from the external $\pi_{0.5}$ comparison in Table 7: LingBot-VLA is the RoboTwin-specialized frozen VLA backend used by Harness VLA, whereas $\pi_{0.5}$ is a representative external VLA baseline. The result indicates that LingBot-VLA already provides a strong and stable contact-rich manipulation capability before agent-level decomposition.

**Role in This Work.** In this paper, LingBot-VLA is used as a frozen execution module within Harness VLA, serving as the low-level contact-rich manipulation primitive for RoboTwin bimanual control.

### L.4 Unified Abstraction

Across all benchmarks, heterogeneous vision-language-action models are uniformly abstracted as interchangeable contact-rich execution primitives. The LLM planner is responsible for semantic grounding, spatial decomposition, and long-horizon task planning, while each VLA is invoked solely for localized interaction execution conditioned on the current observation.

# M AGENT PROMPT SPECIFICATION

This appendix specifies the task prompts used by the LLM planner in Harness VLA. The prompt is not merely a natural-language task instruction. It is the operating manual given to the agent before each rollout: it defines the file-mediated interaction protocol, the observation files that may be used for perception, the allowed primitive vocabulary, the interface to the frozen VLA primitive, the use of Task Specific Memory, and the output artifacts that must be written for reproducibility.

All benchmark prompts follow a shared-core design. A single benchmark-independent prompt template defines the agent's responsibilities, and each benchmark instantiates the slots corresponding to the success predicate, robot embodiment, camera files, primitive schemas, VLA backend, Task Specific Memory paths, and known recovery rules. This shared structure is important because the empirical comparison in the paper evaluates the same agentic harness across LIBERO / LIBERO-Pro, RoboCasa365, and RoboTwin C2R rather than hand-crafting unrelated controllers for each environment.

## M.1 SHARED PROMPT CORE

The shared prompt is written in the second person because it is directly addressed to the agent. Its first paragraph defines the agent role:

```
You are an LLM-in-the-loop hybrid manipulation agent for {BENCHMARK}.
A benchmark driver is already running and waiting for your commands.
Your job is to complete the task by reading the task state, localizing
objects from perception, choosing and executing available primitives,
invoking the VLA when contact-rich behavior is needed, and writing a
reproducible audit.
```

The remainder of the shared prompt is organized into the modules summarized in Table 16. Each module is present in all benchmark prompts, while benchmark-specific prompts fill in concrete fields such as `state.libero_terminated`, `state.success`, `eval_success`, camera names, and primitive schemas.

Table 16: Shared modules in the agent task prompt. Each benchmark-specific prompt keeps this structure and fills in environment-specific details.

| Prompt module | Information given to the agent |
|---|---|
| Role and success signal | Closed-loop control; optimize the benchmark predicate, not a visual guess. |
| Perception isolation | No ground-truth poses or simulator internals; localize from RGB-D and world maps. |
| File-based REPL | Write one JSON command, wait for execution, read refreshed artifacts, then iterate. |
| Primitive vocabulary | Allowed primitive schemas and controller semantics, including gripper, arm, and step conventions. |
| VLA division of labor | VLA for contact-rich phases; analytic primitives for grounding, staging, transport, release, and recovery. |
| Task language | State-file task language is authoritative; do not infer tasks from filenames or indices. |
| Seed 0 Task Specific Memory | JSON audit for strategy and failure modes; JSONL trace for primitive execution order. |
| Global Memory | Reusable success rules and failure observations that provide additional context beyond seed 0 Task Specific Memory. |
| Closed-loop recovery | Verify state, logs, RGB, and geometry after every primitive; diagnose before retrying. |
| Budget and reset policy | Track budget and reset policy; reset is disabled in strict evaluation. |
| Output discipline | Write audit and command trace for both successful and failed rollouts. |

### M.2 Perception and File-Mediated Control

The shared prompt prohibits direct access to ground-truth object poses and simulator internals for spatial reasoning. Backend-reported contact diagnostics and success flags are described separately in Appendix A. The agent receives object names and proprioception from the state file, but not object coordinates. It must localize entities by choosing pixels in RGB images and indexing the corresponding precomputed world map, thereby grounding spatial arguments in perceptual inputs rather than hidden object poses. The common localization instruction is:

```
1. Identify the relevant object, fixture, target surface, or relation
   landmark from RGB.
2. Pick pixels on the visible surface of that entity.
3. Index the matching precomputed world map at those pixels.
4. Sample multiple stable pixels and use a robust statistic, typically
   the median.
5. Avoid rims, object edges, table gaps, holes, reflections, and
   background pixels.
6. Re-localize whenever the robot, camera, object, base, fixture, or
   grasp state changes.
```

This perception rule is paired with the same REPL-style execution contract used throughout the paper:

```
1. Write one JSON command to {WORKDIR}/command.json.
2. Wait until the driver finishes that primitive, typically via
   done_NN.flag, log_NN.json, or a benchmark-specific terminal file.
3. Read the new state_NN.json, log_NN.json, images, depth maps, and
   world maps.
4. Decide the next command from the new evidence.
```

Thus, the prompt enforces the same closed-loop behavior used in the framework description: every primitive call is treated as an experiment whose result must be observed before the next command is issued.

### M.3 Seed 0 Task Specific Memory

The most important memory-related part of the prompt is the instruction for using seed 0 Task Specific Memory. Task Specific Memory is not a plain demonstration to replay. It is a structured memory object that separates semantic strategy from concrete primitive execution. Across benchmarks, the prompt tells the agent to look for two complementary files:

```
{TASK}\_s0.json (Task Specific Memory audit JSON)
{TASK}\_s0.jsonl (Task Specific Memory command JSONL)
```

The JSON file is the audit and strategy summary for the seed 0 rollout. It records the outcome of the reference run and provides high-level notes about the solution strategy, useful primitive choices, recovery decisions, and failure modes observed during exploration. It is not replayed as an action trace; instead, the agent uses it to interpret the reference solution before consulting the JSONL command trace for the concrete primitive order. Depending on the benchmark, this JSON includes the rollout outcome, success status, command or step counts, strategy notes, failure observations, and a summary of the final state.

The JSONL file is the executable command trace. Each line stores one JSON primitive issued by the agent during the reference rollout. The agent reads this trace to recover the procedural structure of the solution: the ordering of primitive calls, the choice of analytic versus VLA-backed actions, the number and placement of VLA invocations, and the transition points between perception, staging, contact-rich execution, transport, release, and verification. The trace is used as a structural prior rather than a trajectory to replay; all spatial arguments are re-grounded from the current observation before execution.

The prompt gives the agent the following rule:

```
Use the JSON audit to understand why the strategy worked and what to
avoid. Use the JSONL trace to understand what was executed and in what
order. Reuse the Task Specific Memory procedural structure, but never
replay literal
coordinates. Previous xyz, xy, quat, pixel locations, base poses, and
fixture coordinates belong to the seed 0 scene. Re-localize every
current
object, destination, support surface, relation landmark, and fixture
from
the current images and world maps.
```

This rule is the prompt-level implementation of Task Specific Memory in Section 2.2. It lets the planner transfer the structure of a successful solution while grounding all geometry in the current rollout.

### M.4 GLOBAL MEMORY

Global Memory complements seed 0 Task Specific Memory. Task Specific Memory is task-specific procedural context: it records the JSON audit and JSONL command trace for one reference rollout. Global Memory is task-independent. It stores reusable success rules and failure models for the fixed primitive library, including known VLA operating conditions, empty-grasp failures, false visual success, unstable staging, and recovery patterns. It is used as contextual guidance during closed-loop execution, not as an action trace to replay.

```
Use Global Memory to check:
1. known success rules for VLA and analytic primitives;
2. known failure models before repeating or repairing an action;
3. empty grasps, wrong-object attempts, false visual success, and
   unstable staging;
```

### M.5 BENCHMARK-SPECIFIC PROMPT INSTANTIATIONS

Tables 17 and 18 summarize how the shared prompt is instantiated for each benchmark. We separate the instantiation into interface-level fields and environment-context fields. The former specifies the success predicate, frozen VLA entry point, and required audit artifacts; the latter records the embodiment and control assumptions that specialize the shared prompt for each benchmark.

Table 17: Interface-level prompt instantiations across benchmarks. In the main text, the heterogeneous VLA primitive names are abstracted as the unified VLA_ACT interface.

| Benchmark | Success signal | VLA interface | Output artifacts |
|---|---|---|---|
| LIBERO / LIBERO-Pro | `libero_terminated` | VLA_ACT | Command JSONL; audit JSON |
| RoboCasa365 | `success` | VLA_ACT | Command JSONL; audit JSON |
| RoboTwin C2R | `eval_success` | VLA_ACT | Command JSONL; audit JSON |

**LIBERO / LIBERO-Pro.** The LIBERO-family prompt is shared by standard LIBERO and LIBERO-Pro:

```
You are an LLM-in-the-loop hybrid driver for the LIBERO PRO/LIBERO
benchmark.
```

Table 18: Environment context supplied by the benchmark-specific prompts.

| Benchmark | Environment-specific prompt specialization |
|---|---|
| LIBERO / LIBERO-Pro | Single-arm tabletop manipulation; fixed agentview and moving wrist RGB-D/world-map observations; agent-side transport, release, and visual verification after contact-rich steps. |
| RoboCasa365 | Mobile kitchen manipulation; base motion is available for reaching distant fixtures; the agent re-localizes after base movement; fixture-facing staging and continuation of capped but progressing contact attempts are treated as part of the operating policy. |
| RoboTwin C2R | Dual-arm manipulation; manual motion commands are bound to a specified arm; head and left/right wrist observations provide perception; manual primitives support observation refresh, non-grasp motion, release, termination, and recovery around contact-rich attempts. |

This instantiation specializes the shared prompt for single-arm tabletop manipulation. It defines the LIBERO observation files used for perception-based grounding, including the fixed agentview RGB-D/world-map files and the moving wrist-camera files used for close-range re-localization. The unified VLA_ACT interface is used for contact-rich steps, including grasping and closed-loop articulated-object, button, or knob manipulation. After contact is established, the agent remains responsible for target identification, scene re-localization, free-space transport, release, and progress verification.

**RoboCasa365.** The RoboCasa prompt instantiates the shared structure for mobile kitchen manipulation:

```
  You are an LLM-in-the-loop hybrid driver for the RoboCasa365 kitchen
  benchmark.
```

This instantiation adds mobile-base staging to the shared manipulation loop. The agent grounds objects and fixtures from RGB-D/world-map observations, uses `navigate_to` for coarse base placement, and uses `move_base` for small local corrections. Because base motion changes the robot viewpoint and the relative arm workspace, the prompt emphasizes re-localization after navigation before continuing manipulation.

The unified VLA_ACT interface provides the contact-rich primitive. The VLA instruction is the full task language, and the planner decides whether to invoke it after local staging or during broader whole-body interaction. A capped but still-progressing VLA call is handled as a continuation case rather than an immediate failure, so the same VLA call can be continued instead of being interrupted by manual commands.

**RoboTwin C2R.** The RoboTwin prompt instantiates the shared core for dual-arm manipulation:

```
  You are an LLM-in-the-loop hybrid manipulation agent for the
RoboTwin
  benchmark.
```

This instantiation specializes the prompt for a dual-arm setting. Manual motion commands include explicit left/right arm binding, and the observation stream includes head and left/right wrist views with corresponding depth and world-map files. The driver reports the official RoboTwin success signal through `eval_success` and writes a terminal `final.json` when the rollout exits.

The contact-rich interface is exposed as VLA_ACT. This primitive is used for grasp formation, re-grasping, handover grasps, bimanual grasp formation, and other contact-rich phases. Manual primitives are used around these VLA attempts for observation refresh, non-grasp motion, release, termination, and recovery. RoboTwin additionally requires a diagnosis Markdown file for post-hoc analysis.

### M.6 COMPACT PROMPT SKELETON

For completeness, the following listing shows the compact shared skeleton underlying all four prompt files. The benchmark-specific prompts fill the bracketed slots with the concrete values described above.

```
You are an LLM-in-the-loop hybrid manipulation agent for {BENCHMARK}.
The benchmark driver is already running in {WORKDIR}. Complete the
task
by reading state and perception files, localizing task entities from
RGB
and world maps, invoking only the allowed primitives, using the frozen
 VLA
for contact-rich behavior, and writing a reproducible audit.

1. ROLE AND SUCCESS SIGNAL
You are a closed-loop controller. Optimize {SUCCESS_SIGNAL}, not a
visual
guess. Continue until success, budget exhaustion, or unrecoverability.


2. PERCEPTION ISOLATION
Do not query simulator object poses or hidden task initialization. Use
RGB for semantic identity and depth/world maps for metric localization
.
Re-localize after every object, camera, robot, base, or grasp change.

3. FILE-BASED REPL
Write one JSON command to {WORKDIR}/command.json. Wait for the driver
result. Read state_NN.json, log_NN.json, images, depth maps, and world
maps. Then decide the next command.

4. PRIMITIVE VOCABULARY
Use only {PRIMITIVE_SCHEMAS}. Preserve exact syntax and controller
semantics, including gripper sign, arm binding, chunk budgets, and
step
costs.

5. DIVISION OF LABOR BETWEEN YOU AND THE VLA
Use {VLA_PRIMITIVE} for grasping, re-grasping, articulated contact,
insertion, pressing, seating, and other contact-rich phases. Use
analytic
primitives for grounding, staging, free-space transport, release,
verification, and recovery.

6. TASK LANGUAGE
Read task_language from the state file. It is authoritative. Do not
infer
the task from filenames, object lists, task indices, or neighboring
Task Specific Memory files.

7. SEED 0 TASK SPECIFIC MEMORY AND GLOBAL MEMORY
Read the task-matched seed 0 Task Specific Memory audit JSON to
understand
why the task-specific strategy worked and what failed. Read the seed 0
Task Specific Memory JSONL to recover what was executed and in what
order. Read Global Memory for cross-task success rules and failure
models.
Use Task Specific Memory as the task-specific procedural skeleton, but
 use
Global Memory and current perception to decide when to re-ground,
verify,
recover, or stop. Never replay literal coordinates.

8. CLOSED-LOOP VERIFICATION AND RECOVERY
```

```
After every command, inspect state, logs, RGB, and world maps.
Diagnose
wrong-object selection, poor stance, VLA miss, short placement, hidden
predicate failure, or unrecoverable displacement before acting again.

9. BUDGET, RESET, AND TERMINATION
Track the benchmark budget and reset policy. Do not reset in strict
evaluation. Stop only on success, budget exhaustion, or
unrecoverability.

10. OUTPUT DISCIPLINE
Write the required audit JSON and command-trace JSONL. The audit JSON
records the benchmark success status and the final outcome fields used
for evaluation and success-rate computation. The JSONL trace records
the
agent's executed primitive commands in order, enabling inspection and
analysis of the agent's decision process.

11. OPERATING LOOP
Read prompt, state, task language, perception, Task Specific Memory,
and
Global Memory. Localize entities. Execute one primitive. Observe.
Recover.
Repeat. Write outputs.
```

This structured prompt design keeps the common Harness VLA operating protocol separate from benchmark-specific assumptions, providing a reusable template for constructing agent prompts in additional manipulation benchmarks.

## N PRIMITIVE USAGE STATISTICS

Table 19 aggregates the primitive calls issued by Harness VLA (CC) across LIBERO Pro-family, RoboTwin C2R, and RoboCasa365 runs. We report canonical primitive names following the taxonomy and availability summary in Table 9: all backend-specific VLA calls are merged into the unified VLA_ACT primitive, implementation-level motion macros are folded into their exposed analytic primitive, and non-manipulation helpers such as rendering, reset, notes, and no-ops are excluded. Percentages are computed within each environment's total manipulation-primitive calls.

The usage pattern supports the intended asymmetric decomposition. In LIBERO, analytic primitives dominate: MOVE_TO alone accounts for $61.8\%$ of calls, while VLA_ACT accounts for $15.8\%$. This matches the tabletop structure of the tasks: the VLA is used primarily to establish contact-rich grasps or fixture interactions, after which analytic transport, gripper control, and release complete much of the rollout. RoboCasa365 shifts the mix toward mobile staging and longer-horizon interaction: NAVIGATE_TO and MOVE_BASE together account for $19.4\%$ of calls, while VLA_ACT rises to $35.3\%$ because kitchen tasks require learned grasps, fixture actuation, and constrained placements across larger scenes. RoboTwin C2R has the highest VLA share ($47.4\%$), reflecting bimanual grasping and handover-like contact, yet analytic primitives still provide a slight majority of calls for planned arm motion, release, and final arrangement.

Table 20 summarizes the same evidence at the class level. Across all three embodiments, the frozen VLA is not used as a monolithic end-to-end controller; it is invoked as a contact-rich primitive inside a larger analytic scaffold. The exact ratio changes with embodiment and task family, but the qualitative division remains stable: analytic primitives handle reproducible geometry and staging, while VLA_ACT supplies the learned local interactions that are difficult to script.

## O FLASH MODE: EXECUTION, GENERATION, AND SELECTION

**Task-card execution.** Flash Mode reuses a successful primitive sequence through a task card that records the action order, VLA prompts, and non-spatial arguments. Spatial waypoints are stored relative to task-relevant objects or destinations. At deployment, a VLM identifies these targets in RGB

Table 19: Canonical primitive usage across benchmark environments. Each cell reports count and percentage of manipulation-primitive calls within that environment. A dash indicates that the primitive is not exposed in the corresponding environment.

| Canonical primitive | Kind | LIBERO | RoboTwin C2R | RoboCasa365 |
|---|---|---|---|---|
| MOVE_TO | Analytic composite | 6263 (61.8%) | 685 (40.9%) | 3004 (38.7%) |
| MOVE_POSE | Analytic composite | 203 (2.0%) | – | – |
| NAVIGATE_TO | Analytic composite | – | – | 701 (9.0%) |
| ROTATE_WRIST | Analytic atomic | 44 (0.4%) | 1 (0.1%) | – |
| ROTATE_PITCH | Analytic atomic | 58 (0.6%) | – | 66 (0.8%) |
| SET_GRIPPER | Analytic atomic | 1137 (11.2%) | 71 (4.2%) | 371 (4.8%) |
| RELEASE | Analytic atomic | 831 (8.2%) | 124 (7.4%) | 76 (1.0%) |
| MOVE_BASE | Analytic atomic | – | – | 808 (10.4%) |
| VLA_ACT | VLA | 1598 (15.8%) | 794 (47.4%) | 2746 (35.3%) |
| **Total** | | 10134 (100.0%) | 1675 (100.0%) | 7772 (100.0%) |

Table 20: Primitive usage grouped by class. Analytic primitives include both composite goal-reaching controllers and atomic set-point commands.

| Primitive class | LIBERO | RoboTwin C2R | RoboCasa365 |
|---|---|---|---|
| Analytic primitives | 8536 (84.2%) | 881 (52.6%) | 5026 (64.7%) |
| VLA primitive, VLA_ACT | 1598 (15.8%) | 794 (47.4%) | 2746 (35.3%) |

images; the system uses the corresponding depth and camera calibration to back-project their image locations into 3D coordinates. It then updates the waypoints and invokes the recorded primitives in order, without online agent planning. Grasp checks and a limited number of retries are predefined in the procedure.

**Candidate generation.** For each task, we compile one card from every available successful agent trajectory. The compiler verifies the benchmark success flag and matches the audit and command trace by suite, task, and seed. It parses the instruction into object relations, aligns them with ordered acquisition, release, and articulated-object interactions, and retains the primitive sequence, policy prompts, and non-spatial arguments. Saved segmentation observations supply reference anchors; missing anchors are reconstructed from the task relations and manipulation transactions. Each candidate consists of an action plan and an anchor specification. Compilation processes one trajectory; candidate evaluation is a separate stage.

**Spatial binding and replay.** In the LIBERO backend, a MOVE_TO or MOVE_POSE waypoint within $0.20\,\mathrm{m}$ of an anchor in the horizontal plane can be stored relative to that anchor. For reference waypoint $\mathbf{x}_i^{\mathrm{ref}} \in \mathbb{R}^2$ and anchor $\mathbf{a}_{j(i)}^{\mathrm{ref}}$, replay uses

$$\Delta\mathbf{x}_i = \mathbf{x}_i^{\mathrm{ref}} - \mathbf{a}_{j(i)}^{\mathrm{ref}}, \qquad \mathbf{x}_i^{\mathrm{run}} = \widehat{\mathbf{a}}_{j(i)} + \Delta\mathbf{x}_i, \tag{1}$$

where $\widehat{\mathbf{a}}_{j(i)}$ is obtained by identifying the anchor in an RGB image with a VLM and back-projecting its image location using depth and camera calibration. Recorded height and non-spatial arguments are preserved; unanchored waypoints retain their coordinates. Localization can be refined from a wrist image, with a held-object offset correction when available. Execution allows at most three grasp attempts, skips dependent transport after an unconfirmed grasp, and stops if a required anchor cannot be localized. No scene reset occurs within the rollout. These binding and retry rules describe the LIBERO implementation; hardware procedures use the corresponding robot interfaces.

**Applicable tasks.** Flash Mode assumes a fixed subgoal order, objects and destinations that can be localized directly, and failures recoverable by the predefined checks and retries. Sequence length alone does not determine applicability: a long fixed routine can be replayed, whereas even a short task requiring a new subgoal, reordered actions, or an unanticipated recovery needs online planning.

**Max-card selection.** Let $\mathcal{C}_\tau$ be the cards compiled from all successful runs of task $\tau$, and $\mathcal{S}_\tau$ its ten-seed batch. For benchmark outcome $Y(c, s) \in \{0, 1\}$, we select

$$\widehat{p}_\tau(c) = \frac{1}{|\mathcal{S}_\tau|} \sum_{s \in \mathcal{S}_\tau} Y(c, s), \qquad c^\star_\tau \in \arg\max_{c \in \mathcal{C}_\tau} \widehat{p}_\tau(c). \tag{2}$$

One card $c^\star_\tau$ is fixed across all seeds of task $\tau$. The same seeds are used for selection and reporting: Flash Mode (max) is therefore a retrospective best-card result, not an independent held-out evaluation, and can be optimistic relative to fresh layouts. Across the 80 task identities in LIBERO-Pro, 78 have cards; `goal_swap_t0` and `10_swap_t9` have no successful source trace and contribute $0/10$. Aggregating all 800 episodes gives 581 successes (72.63%). Both compilation and selection occur offline with frozen VLA weights.

**Comparison and timing protocol.** Table 4 compares Flash Mode with online planning using Codex. The Codex runs form a separate batch from Table 1. All three success rates use 800 episodes. Flash Mode reaches 72.63%, between Codex with high reasoning (78.50%) and without reasoning (62.50%). Codex time averages the ten-seed mean planner durations over 80 tasks, giving 529.23 s with high reasoning and 344.17 s without reasoning. Flash time averages the 78 nonempty source-episode tool-duration entries, giving 56.17 s; the two recorded zero-duration entries are retained, and the two missing plans have no timing entry. Model and service startup are excluded. These timing statistics measure different portions of execution and do not establish an end-to-end speedup.

## P Additional Planner Results: GPT-6 Astra

We follow the experimental setup in Appendix B, using GPT-6 Astra as the Codex planner at low reasoning effort on LIBERO-Pro and RoboCasa365. Tables 21 and 22 report the additional results, with the main-text planner results included for reference.

**LIBERO-Pro.** Table 21 reports 92.63% overall success (741/800) with GPT-6 Astra.

Table 21: LIBERO-Pro success rates (%) with GPT-6 Astra, following the protocol and notation of Table 1. Bold marks the best method in each column.

| Planner | Spat-T | Spat-S | Obj-T | Obj-S | Goal-T | Goal-S | L10-T | L10-S | Overall |
|---|---|---|---|---|---|---|---|---|---|
| Codex (GPT-5.5) | 81.0 | 69.0 | 94.0 | 91.0 | 75.0 | 66.0 | 52.0 | 49.0 | 72.1 |
| Claude Code (Opus-4.7) | 94.0 | 80.0 | 88.0 | 90.0 | 87.0 | 87.0 | 71.0 | 62.0 | 82.4 |
| Codex (GPT-6 Astra) | **100.0** | **98.0** | **100.0** | **99.0** | **88.0** | **99.0** | **85.0** | **72.0** | **92.63** |

**RoboCasa365.** Table 22 reports 59.20% overall success (148/250) with GPT-6 Astra, including 42.50% on Composite-Unseen.

Table 22: RoboCasa365 success rates (%) with GPT-6 Astra, following the protocol of Table 2. Overall averages over tasks. Bold marks the best method in each column.

| Planner | Atomic-Seen | Composite-Seen | Composite-Unseen | Overall |
|---|---|---|---|---|
| Codex (GPT-5.5) | **92.2** | **61.3** | 13.8 | 57.2 |
| Claude Code (Opus-4.7) | 80.0 | 47.5 | 15.0 | 48.8 |
| Codex (GPT-6 Astra) | 87.78 | 43.75 | **42.50** | **59.20** |