MAESTRO: Orchestrating Robotics Modules with Coding Agents for Zero-Shot Generalist Robots

Junyao Shi*, **Rujia Yang***, **Kaitian Chao***, **Selina Bingqing Wan**
Yifei Shao, Jiahui Lei, Jianing Qian, Long Le
Pratik Chaudhari, Kostas Daniilidis, Chuan Wen, Dinesh Jayaraman
* equal contribution
University of Pennsylvania
maestro-robot.github.io

Abstract:
Today's dominant route to generalist robots scales robotics data to train end-to-end policies. We instead use a vision-language model (VLM) as a coding agent that writes robot programs using a curated library of perception, planning, and control tools. MAESTRO's orchestration harness packages multi-step perception and action procedures into callable tools and guides their use through a plan–act–replan loop, using visual and execution feedback to verify outcomes and recover from failures. With this tool library and harness, MAESTRO matches or surpasses state-of-the-art vision-language-action (VLA) models zero-shot in our tabletop comparison and performs diverse mobile manipulation tasks, without collecting new teleoperation data or invoking a VLA tool. Ablations show that expanding the tool library improves average task progress with the full harness, while adding tools without it does not reliably improve progress. Additional case studies examine cross-trial code improvement, VLA tool use, and autonomous seed demonstrations for downstream data augmentation. We argue that such agentic systems offer a complementary, far less-explored axis to data scaling for general-purpose robots. Project site: maestro-robot.github.io. Code: github.com/Beethoven-Q/maestro.

Keywords: VLM agents, robotic manipulation, zero-shot generalization

1 Introduction

The prevailing view in robotics holds that general-purpose capabilities require training a single end-to-end model on massive robotics-specific datasets gathered through teleoperation [1–4]. The scarcity of such data, relative to the text and images available to language and vision models, is widely cited as why robotic systems lag behind, motivating large efforts to scale data collection for vision-language-action (VLA) models.

We show that, without collecting new teleoperation data, MAESTRO matches and often surpasses state-of-the-art VLAs for zero-shot tabletop manipulation, a benchmark domain for generalist robots. Our tabletop and mobile evaluation spans pick-and-place, deformable- and articulated-object manipulation, spatial reasoning, tool use, object-affordance reasoning, semantic and memory challenges, long-horizon manipulation and loco-manipulation, and active exploration. Unsurprisingly, each module MAESTRO composes is itself a robotics *data island*, a learned component (e.g., segmentation, depth, grasping) trained on a massive dataset or a non-learned one (e.g., motion planning, structure-from-motion) distilling decades of community engineering. MAESTRO assembles these islands into one working system, a VLM orchestrating them through a robotics-specific harness that monitors and recomposes tools rather than trusting any one. Across the three ablation tasks, the full toolset and harness achieve 76.9% average task progress. Without the harness, this falls to 33.0% (43.9 percentage points lower); with the full harness but no added robotics tools, it falls to 32.3% (44.6

10th Conference on Robot Learning (CoRL 2026), Austin TX, USA.



Figure 1: **MAESTRO** receives language instruction and leverages a set of tools to complete diverse tasks in a zero-shot setting.

points lower; Fig. 7). It scales modular robotics engineering, with VLMs replacing the human labor it once required.

MAESTRO also inherits the advantages of manually engineered modular systems: interpretability, debuggability, and extensibility. New tools, including VLAs, integrate with minimal effort, and adapting to new embodiments or improving from limited experience requires only localized code edits, not retraining or large-scale data collection.

These advantages currently cost higher latency and compute than end-to-end VLA policies, a transitional limitation easing as VLM inference hardware improves, and one MAESTRO can sidestep by distilling its own autonomously generated trajectories into lightweight imitation policies, a potential self-improving loop [5]. While more robotics data can only help, MAESTRO suggests it is far from the only route to generalist robot policies: pre-trained VLMs may offer a viable, attractive alternative.

In summary, our contributions are:

1. We introduce MAESTRO, a framework that makes a VLM coding agent into a generalist robot policy by combining a curated robotics tool library with an orchestration harness for composing tools, verifying outcomes, and recovering from execution failures.
2. We demonstrate that a VLM coding agent can match or surpass the evaluated VLA and Base CaP baselines zero-shot in our tabletop comparison, without collecting new teleoperation data or invoking a VLA tool. We also demonstrate its capabilities across diverse mobile manipulation tasks.
3. Through systematic ablations, we show that expanding the tool library improves average task progress with the full harness, while adding tools without it does not reliably improve progress (Fig. 7).
4. We present case studies demonstrating cross-trial code improvement, more efficient execution through VLA tool use, and autonomous seed-demonstration collection for downstream data augmentation.

Rollout videos for all real-world experiments and an interactive agent dashboard (perception, code, and rollout views) are on our project site.

2 Related Work

Modern vision-language-action (VLA) foundation models for robotics typically start from pre-trained VLM/LLM backbones and fine-tune on large quantities of robotics-specific data [2, 4, 6–12], including hierarchical and reasoning-augmented variants [7, 8]; in all of these, action selection still flows through a fine-tuned VLA. We are instead interested in directly using off-the-shelf VLMs and LLMs in the robotic control loop. Among such applications of "VLMs and LLMs as policies", directly generating low-level robotic actions has had limited success restricted to simple tasks [13], likely due to the large distribution shift from the web data used to train such models; the most successful approaches instead converge on generating "code as policies". CaP [14] first demonstrated

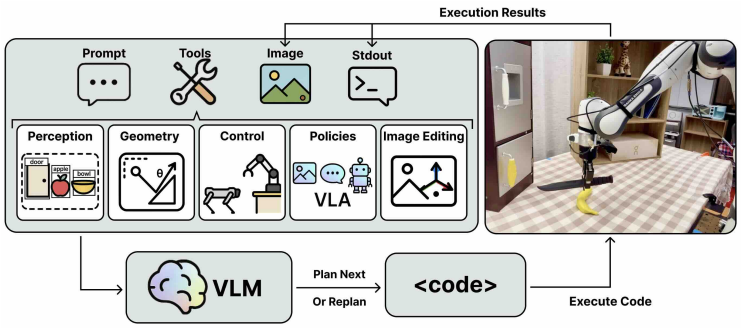


Figure 2: Given prompt and images, VLM plans by writing and executing code that integrates perception, spatial reasoning, control, learned visuomotor policies, and image editing. Execution results (images and stdout) provide feedback for reacting and replanning, forming a closed-loop perception–action–learning cycle. This enables adaptive long-horizon manipulation, as shown at right (instruction: Grasp knife by the handle and cut banana in the middle).

a text-based LLM orchestrating a small set of perception and control APIs into a program, but the generated program is static and open-loop [14–16]. More recent work closes the loop with VLMs, generating and revising code from visual feedback within a trial [8, 17, 18].

MAESTRO studies how a VLM-based harness can orchestrate a broad robotics toolset across manipulation tasks, with ablations evaluating the roles of the toolset and harness. This modular approach is similar to the good old-fashioned engineering (GOFE) discussion [5], which emphasizes combining model-based robotics with learned components.

Direct VLM-based robot control remains challenging, particularly for precise and dexterous manipulation. Many leading VLA systems achieve broad manipulation capabilities but rely on substantial, costly real-world teleoperation [1–4, 10, 12]; complementary data sources include simulation [19–21] and human videos [22–24]. The original zero-shot code-generation setup in Gemini Robotics [2, 4] illustrates these challenges: its reported zero-shot dress-folding trials were unsuccessful, with insufficiently precise grasp generation identified as the main difficulty. We adopt the Gemini Robotics agent as our *Base CaP* reference system and the state-of-the-art $\pi_{0.5}$ model [10] as a strong VLA baseline; Sec. 3.1 describes how Base CaP works and how MAESTRO departs from it by composing external robotics models as callable tools rather than relying on the VLM for perception. We show that scaling the *right set of tools and harnesses* for a coding agent is a complementary axis to scaling robot data, and MAESTRO can further call VLAs as tools where they excel. App. F extends this discussion to VLMs and LLMs as modular components and to concurrent code-as-policies efforts.

Concurrent code-as-policies efforts include CaP-X [25], which benchmarks and improves embodied coding agents, and GaP [26], which builds graph policies from modular robot skills. MAESTRO focuses on a broad robotics toolset and its orchestration harness for real-world generalist manipulation, with ablations that examine their respective roles.

3 Method

We call our approach **M**anagerial **A**gent for **E**xecuting **S**ensorimotor **T**asks in **RO**botics, or **MAESTRO** for short. MAESTRO is a robotic system centered on an agent that writes and executes code to leverage a toolkit spanning perception, geometry, control, pre-trained policies, and image editing.

To specify a task, MAESTRO receives a system prompt, a scene image, and task instructions. This design forms an adaptive perception–action–learning loop; we begin from the Base-CaP starting point (Sec. 3.1) and build up from there. An overview of this loop and the API is shown in Fig. 2, with the per-trial execution loop formalized in Algorithm 1 and the full execution-and-evolution loop in Algorithm 2 (both in App. A).

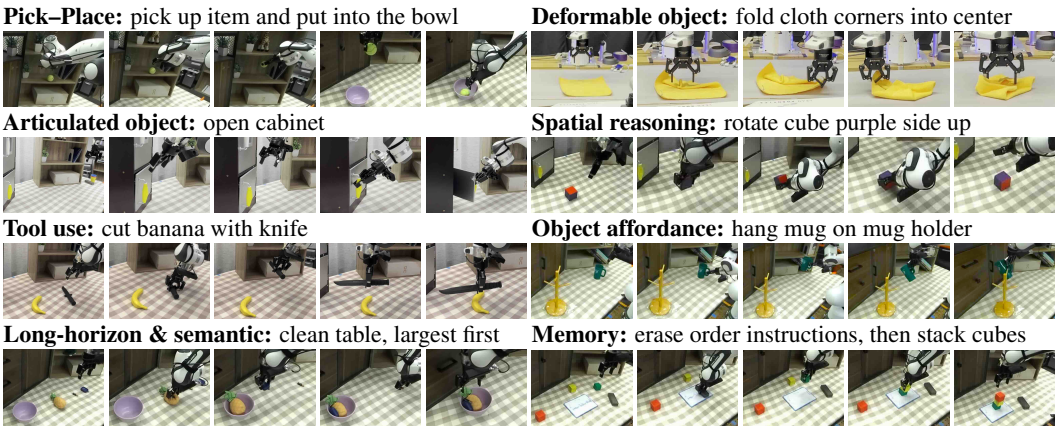


Figure 3: **Tabletop manipulation evaluation tasks.**

3.1 Background: Base CaP coding agents

MAESTRO builds on the design pattern we call *Base CaP*, our name for a representative state-of-the-art closed-loop, code-as-policies coding agent for manipulation, of which the Gemini Robotics agent [2, 4] is the strongest public example. A Base-CaP system gives a VLM a small inventory of robot APIs and has it write and execute Python in a plan-react-replan loop: decompose the instruction, emit code for a substep, observe the resulting image and `stdout`, then advance or rewrite. Crucially, it relies on the VLM's own *default* abilities: a three-tool inventory in which the VLM itself produces bounding-box detections, 2D points, and top-down grasp poses, alongside basic Cartesian and gripper control [2, p. 11, Fig. 12]. This places a hard ceiling on the system, which perceives and grasps only as well as a generalist VLM can. We take this configuration as our starting point, the S0 baseline of the scaling study (Sec. 4.4) and the "Shared CaP pattern" column of Table 1. The rest of this section keeps this closed-loop structure but scales it with curated best-quality modules (Sec. 3.2) and the robotics-specific orchestration harness they require (Sec. 3.3); concurrent code-as-policies efforts that converge on related ideas are discussed in App. F.

3.2 Scaling the tool library: from Base CaP (S0) to MAESTRO (S4)

For tabletop manipulation, we experiment with the widely used DROID [27] platform visualized in Fig. 1, a 7-DoF Franka Panda robotic arm equipped with a Robotiq 2F gripper, supported by a wrist-mounted camera and a third-person camera. For this setup, MAESTRO's tool library is a cumulative progression over the Base-CaP starting point that mirrors the scaling study (Sec. 4.4): each stage replaces a capability the Base-CaP VLM handles with its default abilities (**S0**: 2D pointing and bounding boxes, imagined top-down grasps, Cartesian and gripper control) by the curated, best-quality module for that capability. **S1** adds a coarse-to-fine perception hierarchy (Grounded-SAM segmentation, FoundationStereo [28] metric depth, and ReKep-inspired task-relevant keypoints [29]); **S2** adds GraspGen [30] 6-DoF grasping and cuRobo [31] collision-free motion planning; **S3** adds geometry primitives (vector and rotation operations) that supply the metric spatial reasoning current VLMs lack; and **S4** treats the $\pi_{0.5}$ [10] VLA as a callable policy for short in-distribution skills, run under the monitoring harness of Sec. 3.3. For the mobile manipulator the same library is extended with embodiment-specific modules (base state estimation, active perception, navigation, a fine-grained "nudge" tool, and a semantic map). Table 1 contrasts the full library with Base CaP, and App. A.1 (with Sec. A.3 and Sec. A.4) gives the per-stage and per-module details.

3.3 Robotics-specific orchestration: the harness

Unlike agentic systems in other domains, a robotics agent cannot assume its tools are reliable: perception is noisy, grasps slip, motions are blocked, and learned policies stall. The orchestration problem is thus not just selecting tools but recovering when *any* tool fails, by re-perceiving, re-

planning, or composing a different tool. We use *harness* for the interface layer that encodes how to use the raw robotics modules correctly, knowledge the VLM cannot reconstruct on its own. A robotics agent needs this more than a text-based coding agent: it acts in an open physical world it does not control, and its tool outputs are physical quantities (a 6-DoF grasp, a point cloud) the VLM can neither read nor trust without verification. Designing this harness is central to MAESTRO, and removing it collapses the without-harness curve of Sec. 4.4 even as tools are added.

Concretely, we organize MAESTRO's tool interfaces and execution support into five recurring design principles; Table 1 lists concrete additions over our reproduced Base CaP baseline: (**a**) **encapsulation**, bundling an error-prone upstream-to-downstream module chain behind a single call; (**b**) **procedural knowledge**, injected into the prompt as priorities, guardrails, and worked composition examples; (**c**) **closed-loop control**, monitoring, success detection, and replanning; (**d**) **infrastructure**, resource, lifecycle, and distributed-execution management so heterogeneous modules can coexist; and (**e**) **feedback**, rendering otherwise-unreadable tool outputs (e.g., predicted points and 6-DoF poses) back into the VLM's observation so it can verify them. We describe the harness each tool cluster (S1–S4) requires.

Per-cluster harness (S1–S4). *S1 perception:* localizer-priority guidance and a detect-once-and-cache rule that avoids repeating an $\sim$30 s scan (**b**, **d**), active perception for informative views (**d**), image-editing feedback that renders predicted points and 6-DoF poses back into the observation (**e**), and an encapsulated keypoint tool behind a single call (**a**). *S2 grasp and control:* the clearest case for encapsulation (**a**), a `get_grasp_pose`/`plan_grasp` pair hiding active perception, segmentation, depth, GraspGen, gripper-flange calibration, and collision-free execution, with a sanity-check guardrail (**b**) and GPU-server infrastructure (**d**). *S3 geometry:* purely procedural (**b**), in-context worked examples of composing vectors and rotations. *S4 VLA:* `execute_with_pi05` runs $\pi_{0.5}$ under a parallel $\sim$2 Hz Qwen-2.5-VL monitor that interrupts on success or replan (**a**, **c**), with a resource manager for exclusive camera access and daemon lifecycle (**d**); this monitor is precisely what turns a non-terminating policy into a tool the agent can call and reliably stop. App. A details each.

Underlying all of these is the closed-loop control loop (**c**): the plan-react-replan cycle through which every recovery happens. This loop is shared with Base CaP and supports MAESTRO's per-cluster harness, and we describe it next.

Given a task instruction and an image at the start of an episode, MAESTRO first ***plans***, decomposing the task into substeps and generating code for the first; it then ***reacts***, using the instruction, code output, robot state, and post-execution images to judge whether the subgoal was achieved; if so it ***plans*** the next substep, and if not it ***replans***, diagnosing the failure and rewriting code for the same substep. This ***plan–react–replan*** loop lets MAESTRO continuously adapt to the environment and its own mistakes until the task completes (Alg. 1, Fig. 8, App. A; interactive replay on our project site).

Fig. 9 (App. A) traces one such iteration on the whiteboard-erase task: the react step (Alg. 1) localizes a failed eraser grasp and rewrites one line of `actor_code` to lower the grasp height before the retry succeeds. The loop closes at the code level rather than the retry level: a post-execution image and the robot state let the VLM localize the failure and apply a code edit instead of re-executing.

Monitor reliability. A separate human audit of 372 non-idle step-level success/failure judgments across 61 logged trials found 89.8% accuracy, 85.0% precision, and 99.0% recall for success detection. App. M details the audit.

Cross-trial evolution. MAESTRO can also autonomously improve across repeated trials on the same task: after each trial it feeds a structured execution record (generated code, reasoning trace, and a post-execution root-cause analysis) back to the VLM as in-context history, producing targeted corrections that compound without human intervention (formalized in Alg. 2; open-cabinet case study in App. C, results in App. B).

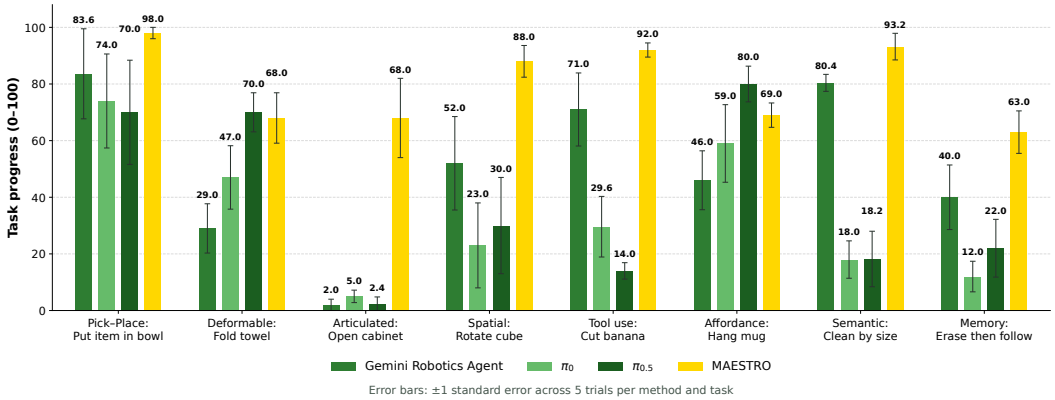


Figure 4: Tabletop manipulation results: mean task progress (0–100; higher is better). Error bars show ± 1 standard error across five trials per method and task; numerical values are reported in Table 2.

4 Experiments

Our experiments evaluate MAESTRO's zero-shot manipulation capabilities and the contributions of its tools and harness. We describe the real-world setup (Sec. 4.1), compare against VLA and Base CaP baselines on tabletop tasks (Sec. 4.2), and evaluate mobile manipulation (Sec. 4.3). We then analyze the tools and harness through ablations (Sec. 4.4) and present case studies of cross-trial code improvement, VLA tool use, and autonomous seed-demonstration collection (Sec. 4.5). Additional analyses examine individual module contributions, tool use, failure modes, and monitor reliability (Apps. H–J and M).

4.1 Real-World Experiment Setup

We evaluate MAESTRO in the real world along three axes, manipulation-challenge category, evaluation-setting diversity, and robot embodiment, using Gemini Robotics-ER 1.5 [4] as its VLM.

Tool availability and experimental use. The default MAESTRO library includes the $\pi_{0.5}$ VLA as a first-class callable tool (Sec. 3.2). The MAESTRO results in the main-text experiments do not invoke this tool; its use is evaluated in the dedicated appendix study (App. K), which compares the full library with a no-VLA ablation.

Embodiments. We evaluate MAESTRO on two embodiments: a Franka-based DROID tabletop platform [27], which best supports comparison against prior generalist policies (tabletop being the most widely studied domain), and a Unitree Go2-W wheeled quadruped with a top-mounted AgileX PiPER arm for mobile manipulation. Full hardware and camera configurations are in App. D.

Tasks. We identify the key challenge axes facing today's generalist robot policies and evaluate MAESTRO on eight tabletop and four mobile manipulation tasks. The eight tabletop tasks span pick-place, deformable object, articulated object, spatial reasoning, tool use, object affordance, long-horizon semantic, and memory challenges (Fig. 3); the four mobile tasks span long-horizon manipulation, long-horizon loco-manipulation, active exploration, and object-affordance reasoning (Fig. 5). Each figure visualizes every task with its instruction.

Evaluation Protocol. We adopt the STAR-Gen taxonomy of generalization [32]. For each task, we evaluate five settings held fixed across methods: one initial setting and four settings generated using STAR-Gen, with perturbations to task-relevant object appearance, object poses, action verbs, and object identity. These are distinct settings rather than repeated runs of an identical scene. The tabletop comparison comprises eight tasks and four methods, for 40 scored trials per method and 160 trials in total. Following π_0 [1], we score task *progress* per trial for quantitative results (App. E).

Human reviewers assigned all task-progress and task-completion scores independently of the agent's monitor.

4.2 Zero-Shot Tabletop Manipulation Results

Baselines. We benchmark MAESTRO against the strongest open-source generalist robot policies from both the Code-as-Policies (CaP) and VLA paradigms. For CaP, we adopt the approach recently described in the Gemini Robotics technical report [2], which uses Gemini to enable zero-shot robot control via code generation. Our baseline implementation follows the system-prompt structure, three-tool inventory (top-down grasp prediction, 2D bounding-box detection, 2D pointing, all performed by the VLM itself rather than via external robotics models), and intra-episode replan loop documented in [2, Fig. 12, App. B.3.2]. This corresponds to the "Shared CaP pattern" column of Table 1 (App. A). We implement this using the latest Gemini Robotics-ER 1.5 [4] and denote it as the **Gemini Robotics Agent**. For VLAs, we compare against the state-of-the-art models π_0 [1] and $\pi_{0.5}$ [10]. Specifically, we use the π_0-FAST-DROID checkpoint for π_0 (π_0-FAST fine-tuned on the DROID dataset) and the $\pi_{0.5}$-DROID checkpoint for $\pi_{0.5}$, which is similarly fine-tuned.



Figure 5: **Mobile manipulation evaluation tasks.**

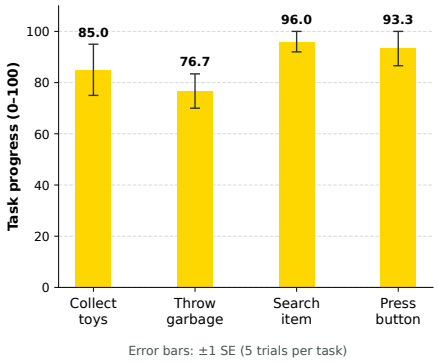


Figure 6: Mobile manipulation results: mean task progress (0–100; higher is better). Error bars show ± 1 standard error across five trials per task; numerical values are reported in Table 3.

Results. Fig. 4 summarizes the tabletop results, with detailed numbers in App. G. MAESTRO has the highest mean task progress on six of eight tasks in the selected five-trial comparison. $\pi_{0.5}$ has the highest mean on fold towel and hang mug on mug holder. VLA baselines struggle on several perturbed settings and lack an explicit memory mechanism. Their mean progress is low on open cabinet; on rotate cube, the selected trials yield means of 23.0 for π_0 and 30.0 for $\pi_{0.5}$, compared with 88.0 for MAESTRO. The Gemini Robotics Agent, our CaP baseline, plans well but struggles to turn high-level plans into low-level actions: its agentic loop is built around Gemini's own perception [2, p. 11], so it is limited to whatever bounding boxes, points, and top-down grasps the VLM produces on its own, and cannot, for instance, localize a cabinet handle without active perception or grasp and orient an eraser or towel corners without external segmentation, depth, and grasp modules.

MAESTRO bridges this gap less through the tool modules than through the orchestration harness around them (Sec. 3.3): it hides error-prone chains like point-cloud reconstruction and grasp prediction behind single calls (encapsulation), injects procedural guidance on module choice and exact geometry for reorientation, and renders tool outputs back for the VLM to verify points and 6-DoF poses. When a step fails, its closed-loop plan-react-replan cycle re-perceives and recomposes tools, e.g., recovering from a failed top-down grasp of a shelved tennis ball via active perception and the grasp model for a safer, reachable pose. This lets MAESTRO surpass prior CaP systems in dexterity and task coverage on tasks typically considered better suited to VLA-style approaches, though hang mug on mug holder

still exceeds its affordance and orientation reasoning. Rollout videos for all eight tabletop tasks are on our project site.

4.3 Zero-Shot Mobile Manipulation Results

Baselines. No open-source zero-shot CaP or VLA generalist policies exist for our wheeled-quadruped embodiment, so we evaluate MAESTRO alone, reporting per-task progress and failure modes.

Results. Fig. 6 reports mean mobile manipulation progress and ± 1 standard error across five trials per task; Table 3 gives the numerical means and SEs (App. G). The two *long-horizon* tasks show lower progress rates due to their multi-stage object interactions; a cached semantic map substantially improves performance by letting the agent re-track objects and complete sub-tasks without redundant search. *Active exploration* achieves high progress because we provide multi-view images at replan sessions to support spatial reasoning, and *press button* benefits from precise keypoint selection. Remaining failures primarily stem from low-level execution; we decompose them by module class in App. J. Rollout videos for all four mobile tasks are available on our project site.

We analyze MAESTRO's failure modes in detail, broken down per embodiment and module class with concrete trial examples, in App. J.

4.4 How important is each component of MAESTRO?

Table 1 (App. A) lists the tools and harness components that MAESTRO adds over Base CaP. We examine their contributions by expanding the tool library from Base CaP (S0) through advanced perception (S1), grasping and control (S2), and geometry and spatial reasoning (S3). At each stage we compare four conditions: **Full harness (I+E)**, **Interface only (I)**, **Execution only (E)**, and **No harness**. Following the five harness principles defined in Sec. 3.3, Interface retains encapsulation and procedural knowledge, while Execution retains closed-loop control, infrastructure, and feedback. We group the latter three because they work together during and after execution. We report per-task progress for opening a cabinet, rotating a cube so a target face points up, and grasping a knife by its handle to cut a banana. Aggregate values are unweighted means of these three task means.

Results. Fig. 7 shows that the full harness has the highest or tied mean in all nine S1–S3 task-stage comparisons. Its three-task average rises from 32.3% at S0 to 44.0% at S1, 57.7% at S2, and 76.9% at S3, although individual tasks need not improve at every stage. At S3, Interface only reaches 59.3%, Execution only 34.0%, and No harness 33.0%. The partial-harness variants generally fall between full and no harness, but Execution only is lower than No harness for open cabinet at S3 (6% versus 11%). These results support the value of the full harness in this setting and show that adding tools alone does not reliably improve progress.

Complementary analyses corroborate this: per-module *leave-one-out* ablations (App. H) provide additional module-level comparisons, and a tool-call-level analysis (App. I) shows MAESTRO composes task-specific rather than fixed workflows, with 47 of 63 initially failed subtask instances later receiving a logged success judgment (74.6%). This conditional, log-derived recovery statistic is distinct from human-scored trial completion. The scaling study evaluates S0–S3; the contribution of the S4 VLA tool is evaluated separately in App. K.

4.5 Beyond single-trial execution

Beyond single-trial execution, we report case studies of cross-trial code evolution, VLA tool use, and autonomous seed-demonstration collection (App. B). The detailed setups and results appear in App. C, App. K, and App. L.

5 Limitations

MAESTRO's closed-loop harness catches and recovers from individual tool failures rather than assuming every module always succeeds, which is much of why it outperforms a naive composition

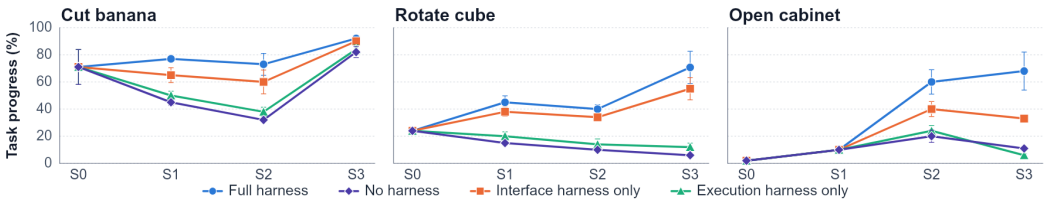


Figure 7: **Scaling MAESTRO's tools and harness together.** Mean task progress for cut banana, rotate cube, and open cabinet as tool groups are added to Base CaP (S0): S1 (+perception), S2 (+grasp/control), and S3 (+geometry). The four conditions are Full harness (I+E), Interface only (I), Execution only (E), and No harness. Error bars show per-task standard errors, computed as sample SD divided by $\sqrt{n}$ for each task–stage–condition cell. S0 uses the same five baseline trials per task for all four curves.

of the same tools. Recovery has limits, however: when a module is systematically wrong in a way MAESTRO cannot observe or route around, the error propagates to the task. These residual failures concentrate in perception and low-level control, and more delicate control will require richer low-level behaviors than the current toolset supports. A second limitation is runtime: querying a VLM at every plan-react-replan step makes MAESTRO considerably slower than end-to-end VLAs such as $\pi_{0.5}$, though we view this as transitional. App. J details the full per-embodiment failure breakdown and latency outlook, which we leave to future work.

6 Conclusion

We present MAESTRO, a VLM-driven coding agent for tabletop and mobile manipulation, equipped with a curated tool library and the orchestration infrastructure to compose and adapt these tools. Without collecting new teleoperation data, it outperforms the evaluated VLAs and Base CaP system in our tabletop comparison. Ablations highlight the importance of the harness: adding tools alone does not reliably improve performance. Its modular interfaces allow improved tools and VLMs to be incorporated as they become available, offering a complementary route to scaling robot data.

Acknowledgments

This work was supported in part by DARPA TIAMAT HR0011249042, NSF CAREER 2239301, ONR N00014-22-1-2677, NSF SLES 2331783, and NAIRR Pilot 240077. PC acknowledges a DURIP grant from the Office of Naval Research.

References

[1] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[2] G. R. Team, S. Abeyruwan, J. Ainslie, J.-B. Alayrac, M. G. Arenas, T. Armstrong, A. Balakrishna, R. Baruch, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

[3] J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. Fan, Y. Fang, D. Fox, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

[4] G. R. Team, A. Abdolmaleki, S. Abeyruwan, J. Ainslie, J.-B. Alayrac, M. G. Arenas, A. Balakrishna, N. Batchelor, et al. Gemini robotics 1.5: Pushing the frontier of generalist robots with advanced embodied reasoning, thinking, and motion transfer, 2025. URL https://arxiv.org/abs/2510.03342.

[5] K. Goldberg. Good old-fashioned engineering can close the 100,000-year "data gap" in robotics. *Science Robotics*, 10(105):eaea7390, 2025. doi:10.1126/scirobotics.aea7390. URL https://www.science.org/doi/abs/10.1126/scirobotics.aea7390.

[6] M. Zawalski, W. Chen, K. Pertsch, O. Mees, C. Finn, and S. Levine. Robotic control via embodied chain-of-thought reasoning. *arXiv preprint arXiv:2407.08693*, 2024.

[7] W. Chen, S. Belkhale, S. Mirchandani, O. Mees, D. Driess, K. Pertsch, and S. Levine. Training strategies for efficient embodied reasoning. *arXiv preprint arXiv:2505.08243*, 2025.

[8] L. X. Shi, B. Ichter, M. Equi, L. Ke, K. Pertsch, Q. Vuong, J. Tanner, A. Walling, et al. Hi robot: Open-ended instruction following with hierarchical vision-language-action models. *arXiv preprint arXiv:2502.19417*, 2025.

[9] Q. Zhao, Y. Lu, M. J. Kim, Z. Fu, Z. Zhang, Y. Wu, Z. Li, Q. Ma, et al. Cot-vla: Visual chain-of-thought reasoning for vision-language-action models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 1702–1713, 2025.

[10] P. Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

[11] J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. Fan, Y. Fang, D. Fox, et al. Gr00t n1.5: An improved open foundation model for generalist humanoid robots, 2025. URL https://research.nvidia.com/labs/gear/gr00t-n1_5/. NVIDIA technical report.

[12] J. Lee, J. Duan, H. Fang, Y. Deng, S. Liu, B. Li, B. Fang, J. Zhang, et al. Molmoact: Action reasoning models that can reason in space. *arXiv preprint arXiv:2508.07917*, 2025.

[13] T. Kwon, N. D. Palo, and E. Johns. Language models as zero-shot trajectory generators. *IEEE Robotics and Automation Letters*, 9(7):6728–6735, July 2024. ISSN 2377-3774. doi:10.1109/lra.2024.3410155. URL http://dx.doi.org/10.1109/LRA.2024.3410155.

[14] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. In *arXiv preprint arXiv:2209.07753*, 2022.

[15] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, et al. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530, 2023. doi:10.1109/ICRA48891.2023.10161317.

[16] H. Ha, P. Florence, and S. Song. Scaling up and distilling down: Language-guided robot skill acquisition. In *Proceedings of the 2023 Conference on Robot Learning*, 2023.

[17] J. Duan, W. Yuan, W. Pumacay, Y. R. Wang, K. Ehsani, D. Fox, and R. Krishna. Manipulate-anything: Automating real-world robots using vision-language models. *arXiv preprint arXiv:2406.18915*, 2024.

[18] P. Zhi, Z. Zhang, Y. Zhao, M. Han, Z. Zhang, Z. Li, Z. Jiao, B. Jia, et al. Closed-loop open-vocabulary mobile manipulation with gpt-4v. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4761–4767. IEEE, 2025.

[19] M. Dalal, M. Liu, W. Talbott, C. Chen, D. Pathak, J. Zhang, and R. Salakhutdinov. Local policies enable zero-shot long-horizon manipulation. *International Conference of Robotics and Automation*, 2025.

[20] T. G. W. Lum, M. Matak, V. Makoviychuk, A. Handa, A. Allshire, T. Hermans, N. D. Ratliff, and K. Van Wyk. Dextrah-g: Pixels-to-action dexterous arm-hand grasping with geometric fabrics. *arXiv preprint arXiv:2407.02274*, 2024.

[21] T. Lin, K. Sachdev, L. Fan, J. Malik, and Y. Zhu. Sim-to-real reinforcement learning for vision-based dexterous manipulation on humanoids. *arXiv:2502.20396*, 2025.

[22] J. Shi, Z. Zhao, T. Wang, I. Pedroza, A. Luo, J. Wang, J. Ma, and D. Jayaraman. Zeromimic: Distilling robotic manipulation skills from web videos. In *International Conference on Robotics and Automation (ICRA)*, 2025.

[23] R. Yang, Q. Yu, Y. Wu, R. Yan, B. Li, A.-C. Cheng, X. Zou, Y. Fang, et al. Egovla: Learning vision-language-action models from egocentric human videos. *arXiv preprint arXiv:2507.12440*, 2025.

[24] L. Y. Zhu, P. Kuppili, R. Punamiya, P. Aphiwetsa, D. Patel, S. Kareer, S. Ha, and D. Xu. Emma: Scaling mobile manipulation via egocentric human data. *IEEE Robotics and Automation Letters*, 2026.

[25] L. Fu, J. Yu, K. El-Refai, E. Kou, H. Xue, H. Huang, W. Xiao, G. Wang, et al. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026. URL https://arxiv.org/abs/2603.22435. Concurrent work; NVIDIA + UC Berkeley + CMU + Stanford. Project page: https://capgym.github.io/.

[26] K. Chen, S. Xie, L. Fu, J. Yu, W. Pacini, S. Bajamahal, H. Kim, J. Drake, et al. GaP: A graph-as-policy multi-agent self-learning harness for variational automation tasks. *arXiv preprint arXiv:2607.05369*, 2026. URL https://arxiv.org/abs/2607.05369.

[27] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srirama, et al. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.

[28] B. Wen, M. Trepte, J. Aribido, J. Kautz, O. Gallo, and S. Birchfield. Foundationstereo: Zero-shot stereo matching. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5249–5260, 2025.

[29] W. Huang, C. Wang, Y. Li, R. Zhang, and L. Fei-Fei. Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. *arXiv preprint arXiv:2409.01652*, 2024.

[30] A. Murali, B. Sundaralingam, Y.-W. Chao, W. Yuan, J. Yamada, M. Carlson, F. Ramos, S. Birchfield, et al. Graspgen: A diffusion-based framework for 6-dof grasping with on-generator training. *arXiv preprint arXiv:2507.13097*, 2025.

[31] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. Van Wyk, V. Blukis, A. Millane, H. Oleynikova, et al. curobo: Parallelized collision-free minimum-jerk robot motion generation. *arXiv preprint arXiv:2310.17274*, 2023.

[32] J. Gao, S. Belkhale, S. Dasari, A. Balakrishna, D. Shah, and D. Sadigh. A taxonomy for evaluating generalist robot manipulation policies. *IEEE Robotics and Automation Letters*, 2026.

[33] Gemini Robotics Team. Gemini Robotics-ER 1.6, 2026. URL https://deepmind.google/blog/gemini-robotics-er-1-6/.

[34] J. Yang, H. Zhang, F. Li, X. Zou, C. Li, and J. Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*, 2023.

[35] F. Liu, K. Fang, P. Abbeel, and S. Levine. Moka: Open-world robotic manipulation through mark-based visual prompting. *arXiv preprint arXiv:2403.03174*, 2024.

[36] C. Bai, T. Xiao, Y. Chen, H. Wang, F. Zhang, and X. Gao. Faster-lio: Lightweight tightly coupled lidar-inertial odometry using parallel sparse incremental voxels. *IEEE Robotics and Automation Letters*, 7(2):4861–4868, 2022.

[37] S. Macenski, F. Martín, R. White, and J. G. Clavero. The marathon 2: A navigation system. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2718–2725. IEEE, 2020.

[38] W. Liang, S. Wang, H.-J. Wang, O. Bastani, Y. J. Ma, and D. Jayaraman. Tether: Autonomous functional play with correspondence-driven trajectory warping. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2603.03278.

[39] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, J. Fan, et al. Eureka: Human-level reward design via coding large language models. In *International conference on learning Representations*, volume 2024, pages 26516–26560, 2024.

[40] Y. J. Ma, W. Liang, H. Wang, S. Wang, Y. Zhu, L. Fan, O. Bastani, and D. Jayaraman. Dreureka: Language model guided sim-to-real transfer. In *Robotics: Science and Systems (RSS)*, 2024.

[41] W. Liang, S. Wang, H.-J. Wang, O. Bastani, D. Jayaraman, and Y. J. Ma. Eurekaverse: Environment curriculum generation via large language models. In *Conference on Robot Learning (CoRL)*, 2024.

[42] L. Le, J. Xie, W. Liang, H.-J. Wang, Y. Yang, Y. J. Ma, K. Vedder, A. Krishna, et al. Articulate-anything: Automatic modeling of articulated objects via a vision-language foundation model. *arXiv preprint arXiv:2410.13882*, 2024.

[43] G. J. Gao, T. Li, J. Shi, Y. Li, Z. Zhang, N. Figueroa, and D. Jayaraman. Vlmgineer: Vision language models as robotic toolsmiths. *arXiv preprint arXiv:2507.12644*, 2025.

[44] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023.

[45] H. Huang, F. Lin, Y. Hu, S. Wang, and Y. Gao. Copa: General robotic manipulation through spatial constraints of parts with foundation models. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9488–9495. IEEE, 2024.

[46] S. Patel, X. Yin, W. Huang, S. Garg, H. Nayyeri, L. Fei-Fei, S. Lazebnik, and Y. Li. A real-to-sim-to-real approach to robotic manipulation with vlm-generated iterative keypoint rewards. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8258–8266. IEEE, 2025.

[47] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*, 2022.

[48] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, et al. Inner monologue: Embodied reasoning through planning with language models. In *arXiv preprint arXiv:2207.05608*, 2022.

[49] Y. Feng, J. Han, Z. Yang, X. Yue, S. Levine, and J. Luo. Reflective planning: Vision-language models for multi-stage long-horizon robotic manipulation. *arXiv preprint arXiv:2502.16707*, 2025.

[50] S. Nasiriany, F. Xia, W. Yu, T. Xiao, J. Liang, I. Dasgupta, A. Xie, D. Driess, et al. Pivot: Iterative visual prompting elicits actionable knowledge for vlms. *arXiv preprint arXiv:2402.07872*, 2024.

[51] W. Yuan, J. Duan, V. Blukis, W. Pumacay, R. Krishna, A. Murali, A. Mousavian, and D. Fox. Robopoint: A vision-language model for spatial affordance prediction for robotics. *arXiv preprint arXiv:2406.10721*, 2024.

[52] N. Di Palo and E. Johns. Keypoint action tokens enable in-context imitation learning in robotics. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

[53] Y. Yin, Z. Wang, Y. Sharma, D. Niu, T. Darrell, and R. Herzig. In-context learning enables robot action prediction in llms. In *ICRA*, 2025.

[54] Z. Liu, A. Bahety, and S. Song. Reflect: Summarizing robot experiences for failure explanation and correction. *arXiv preprint arXiv:2306.15724*, 2023.

[55] J. Duan, W. Pumacay, N. Kumar, Y. R. Wang, S. Tian, W. Yuan, R. Krishna, D. Fox, et al. Aha: A vision-language-model for detecting and reasoning over failures in robotic manipulation, 2024. URL https://arxiv.org/abs/2410.00371.

[56] Y. J. Ma, J. Hejna, C. Fu, D. Shah, J. Liang, Z. Xu, S. Kirmani, P. Xu, et al. Vision language models are in-context value learners. In *International Conference on Learning Representations*, volume 2025, pages 33984–34009, 2025.

[57] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Appendix

A Additional Method Details

Plan–react–replan execution and safety. MAESTRO's generated code runs against a constrained, curated API rather than free-form actuation. Execution errors and planning failures are returned to the agent in `stdout` for replanning; cuRobo checks modeled collisions and inverse-kinematics feasibility. An interactive replay on a representative trial, including a case where MAESTRO detects mid-execution that the whiteboard was not fully erased and autonomously triggers a recovery, is available on the Agent Dashboard at our project site.

Implementation. MAESTRO's code is available at github.com/Beethoven-Q/maestro.

A.1 Tool library by scaling stage (S0–S4)

This subsection gives the staged description of MAESTRO's tabletop tool library summarized in Sec. 3.2, presented as a cumulative progression over the Base-CaP starting point that mirrors the scaling study of Sec. 4.4: each stage replaces a capability the Base-CaP VLM handles with its default abilities by the curated, best-quality module for that capability. Table 1(a) lists the same library compactly, and Sec. A.3 and Sec. A.4 give the full per-module mechanics.

S0 (Base CaP): the VLM's default abilities. The starting point exposes only what the VLM can do by default (Sec. 3.1): 2D pointing and bounding boxes for localization, VLM-imagined top-down grasps, and basic Cartesian and gripper control. Each cluster below replaces one of these defaults with the best-quality module available for that capability.

S1 (+advanced perception). We replace the VLM's default perception with a *coarse-to-fine* hierarchy of specialized vision modules that lets the agent trade speed against precision: raw RGB and proprioception (fastest, coarsest); Grounded-SAM segmentation-mask centroids and 2D pointing (medium); FoundationStereo [28] metric depth, which supplies the reliable 3D positions and dense object point clouds that the VLM's monocular guesses cannot; and VLM-selected task-relevant keypoints (ReKep-inspired [29]) for precise but slower localization of features a VLM struggles to point at directly. Two harness capabilities, *active perception* and *image-editing feedback*, make this hierarchy usable in practice; we detail them with the rest of the perception harness in Sec. 3.3. None of these abilities exist in Base CaP, which perceives only as well as the generalist VLM itself, with no metric depth and no point clouds.

S2 (+grasping and control). Base CaP can only execute the top-down grasps the VLM imagines and moves the arm by direct Cartesian servoing with no notion of obstacles; we replace both. For grasping, GraspGen [30] is a learned grasp model that predicts ranked 6-DoF grasps on an object's point cloud, expressing the side, angled, and functional-part grasps a top-down pose cannot. Because grasp quality hinges on a clean point cloud, the grasp pipeline first drives a planned multi-view scan of the wrist camera and fuses the views into a single object point cloud before prediction. For control, we add cuRobo [31] point-cloud-based collision-free motion planning, letting MAESTRO reach those grasps and move through cluttered scenes while avoiding unintended contact, which is essential whenever the arm must make and break contact with objects or operate around obstacles. Both capabilities are entirely absent at S0–S1, where the agent has neither a 6-DoF grasp source nor any collision awareness.

S3 (+geometry and spatial reasoning). In our initial experiments, the agent struggled with metric spatial reasoning even with improved perception. We therefore provide explicit tools for computing distances, vectors, and rotations from perception outputs. These tools are not included in the Base CaP baseline.

S4 (+VLA). Finally, MAESTRO treats a learned visuomotor policy, $\pi_{0.5}$ [10], as a callable tool for fast, reliable execution of short skills such as pick-and-place. A raw VLA has no self-termination signal, so exposing it as a usable tool depends on the monitoring harness of Sec. 3.3; the agent selects

the VLA only when it fits the subtask (App. B) and falls back on the procedural modules above otherwise.

Mobile manipulation. For the mobile embodiment, the same library is extended with embodiment-specific modules: mobile-base state estimation, active-perception tools (look left / right / ground, view the carry-on basket, log object locations), navigation, a fine-grained "nudge" tool for local base adjustment, and a semantic map that caches observed object locations to support persistent reasoning over long-horizon tasks.

A.2 Per-stage harness details

This subsection gives the full mechanics of the per-cluster harness summarized in Sec. 3.3, one paragraph per scaling stage; Table 1(b) lists the same infrastructure compactly, and the tags (**a**)–(**e**) refer to the five harness design principles of Sec. 3.3.

S1 perception harness. The perception harness adds: priority guidance on which localizer to try first and when to escalate from pointing to segmentation to keypoints, plus a "detect all objects once and cache" rule that avoids repeating an $\sim$30 s scan (**b**, **d**); active perception that lets the agent reposition the wrist camera, zooming in or looking around, so the downstream segmentation, depth, and keypoint tools receive informative observations, since off-the-shelf vision models are only as good as the views they are given (**d**); image-editing feedback that draws predicted points and 6-DoF poses back onto the observation image, so the VLM can visually confirm a result it cannot read as raw numbers (**e**); and an encapsulated keypoint tool that hides a multi-step pipeline behind one call: best-view selection, segmentation, candidate-point sampling, image rendering, depth projection, and a second VLM that picks the matching point (**a**).

S2 grasp and control harness. Grasping is the clearest case for *encapsulation* (**a**): a long, error-prone tool chain must be hidden behind a single call. A single `get_grasp_pose` call internally runs active perception (a planned spherical look-around of the wrist camera with multi-view fusion into a clean object point cloud), then segmentation, depth, and GraspGen, returning a ranked 6-DoF grasp; a companion `plan_grasp` fuses that grasp with control, applying a gripper-flange calibration offset and collision-free planning before executing (**a**, plus calibration knowledge the VLM should not manage). Prompt-level guidance adds a grasp sanity-check guardrail and side-vs-top-down strategy (**b**). Because these modules are heavy, the harness runs them on a GPU server, handling file sync and client–server communication, and a scene-scan cache persists multi-view captures for reuse (**d**). The agent simply asks for a grasp.

S3 geometry harness. The geometry primitives are already atomic, so their harness is purely procedural (**b**): in-context worked examples of composing vectors and rotations (e.g. building an axis from two keypoints and rotating about it) and guidance on when to use them for orientation, so the agent reaches for exact geometry instead of guessing poses.

S4 VLA harness. $\pi_{0.5}$ runs fast but has no signal for when it has finished. MAESTRO wraps it in `execute_with_pi05`, which launches the policy as a background daemon and spawns a parallel Qwen-2.5-VL monitor that checks task-relevant completion conditions on the live image at $\sim$2 Hz and interrupts the policy on success or when a replan is needed (**a**, **c**). A resource manager grants the VLA exclusive camera access and manages daemon lifecycle so it can run alongside the main coding agent (**d**). This monitor is precisely what turns a non-terminating policy into a tool the agent can call and reliably stop.

Curating for quality, not for external models per se. The best-quality modules MAESTRO adopts are usually specialized, externally-developed systems, but that is a consequence of curating for quality rather than a goal in itself, since the generalist VLM is rarely the best tool for a precise robotics subtask.

Mobile-manipulation replanning. In mobile manipulation settings, before performing failure analysis and rewriting code, MAESTRO is also prompted to actively look around and perceive the environment to build a more complete understanding of its surroundings before it replans.

Table 1: Comparison of MAESTRO's tool library and harness against *Base CaP*, the closed-loop coding-agent pattern MAESTRO builds on (Sec. 3.1; the Gemini Robotics agent [2, Fig. 12, App. B.3.2] is its strongest public instance). Components in **bold** are introduced or substantially scaled in MAESTRO; cells marked *None* indicate no equivalent in Base CaP. Both parts are organized by the scaling stage, from S0 (Base CaP) through S4, at which each capability enters: part (a) lists the curated robotics modules added at each stage (Sec. 3.2), and part (b) lists the harness infrastructure each stage requires (Sec. 3.3). Tags (**a**)–(**e**) in part (b) name the five harness design principles of Sec. 3.3.

Stage / component	Base CaP	MAESTRO
(a) Tool library — curated robotics modules composed as callable tools, by scaling stage (Sec. 3.2)		
S0 (Base CaP)	VLM-internal 2D pointing, bounding boxes, imagined top-down grasps; Cartesian + gripper control [2, 4]	*Starting point; each row below replaces a default with a curated module*
S1: advanced perception	VLM performs perception itself [2, 6, 17]	**Grounded-SAM segmentation centers; FoundationStereo [28] metric depth; ReKep-inspired task-relevant keypoints [29]; active perception (zoom / look around with wrist camera); point and 6-DoF-pose image-editing feedback**[†]
S2: grasping & control	Imagined top-down grasp only; direct Cartesian moves, no collision awareness [2, 4]	**GraspGen [30] 6-DoF grasps (side / angled / functional-part) with multi-view fusion; cuRobo [31] collision-free motion planning**
S3: geometry & spatial reasoning	*None* (language-only)	**Distance, vector construction, rotation between vectors, rotation about an axis**
S4: VLA	*None*	**$\pi_{0.5}$ [10] VLA with high-frequency closed-loop monitoring (Qwen-2.5-VL)**
Additional modules for mobile manipulation		
Perception (mobile)	Build global/local map [18]	Mobile-base state estimation; **active perception (look left / right / ground, view carry-on basket, log object location)**
Locomotion (mobile)	Navigation [18]	Navigation; **fine-grained "nudge" tool for local adjustment**
(b) Harness — orchestration that makes the tools usable, grouped by stage (Sec. 3.3)		
Closed-loop plan-react-replan (shared substrate, c)	✓ Intra-episode, stdout + image after each code block	✓ Intra-episode, monitor-driven per-step revision
Code execution + stdout feedback	✓	✓
S1: perception harness	*None*	**Pointing/segmentation/keypoint priority + detect-once caching (b, d); point and 6D-pose feedback drawn into the observation (e); encapsulated keypoint tool (a)**
S2: grasp & control harness	*None*	**`get_grasp_pose`/`plan_grasp` tools bundling active perception, segmentation, depth, GraspGen and cuRobo (a); grasp guardrail + side/top-down guidance (b); GPU client–server + scan cache (d)**
S3: geometry harness	*None*	**In-context vector/rotation composition examples + orientation guidance (b)**
S4: VLA harness	*None*	**`execute_with_pi05` + Qwen-2.5-VL $\sim$2 Hz monitor interrupting on completion/replan (a, c); resource manager for camera handoff and daemon lifecycle (d)**
Cross-trial reflection / code evolution	*None* (intra-episode only)	**Per-trial video-grounded success/failure analysis fed in-context to the next trial (c)**
Multi-embodiment scope	ALOHA 2 only	**Franka tabletop + mobile manipulator**

[†]Concurrent with this work, Gemini Robotics-ER 1.6 [33] has begun adding Python-based image manipulation for ambiguity resolution.

A.3 Tabletop Manipulation Modules

We equip our tabletop manipulation agent with a suite of tools spanning perception, reasoning, control, pre-trained visuomotor policies, and image editing. The tools are provided in **bolded text** below.

Perception. For perception tools, we adopt a "coarse-to-fine" approach. At the fastest and simplest level, the agent can access **raw sensory inputs** (RGB images and robot proprioception). Because raw

Initial Plan: Please finish the Receive Instruction, Describe the Scene, and Steps Planning process. At the end, you should enclose the first step code to be executed.

Robot Task Instructions: You are a helpful franka panda robot arm...

Task Execution Procedure: <1. Receive Instruction> ··· <2. Describe the Scene> ··· <3. Steps Planning> ··· <4. Steps Execution>

Robot Hardware and Physical Constraints: world frame orientation...

Robot API Documentation:
class RobotApi: """class variable defined that can be accessed at any point of the program""":
    def get_task_relevant_keypoints(self, object_name, prompt) → List[3d_points]
    def move_gripper_to(self, position, orientation)
    def get_grasp_pose(self, object_name, subtask_instruction, point_to_grasp)
    def rotate_by(self, current_orientation, rotation)
    ···
class ImageEditApi: """Interface for editing the image for a better understanding."""
    ···

Example Code
Here is an example code for the task 'Pick up the red cube and rotate.' You should learn how and when to use the functions in the API and try to imitate the code to complete the new task.
```python
points = robot.get_task_relevant_keypoints("red cube", "center of red cube")
grasp_results = robot.get_grasp_pose("cube", "pick up cube", points[0]['3d_position'])
···
target_rotation = robot.rotate_by(curr_orientation, [0, -np.pi, 0])
```

Replan: Describe the execution result and compare it with the goal and the expert demo, is this step successful? If successful, output the code for next step. If not, summarize what caused the failure, find the problem of your code, write code to return to some free state, and then rewrite a better code for the task.

Figure 8: A summarized overview of MAESTRO's system prompt.

Algorithm 1 MAESTRO execution loop (single trial). The cross-trial evolution loop that wraps this procedure is given in Algorithm 2.

Require: Task $\mathcal{T}$, initial image I_0, tool library $\mathcal{M}$, system prompt $\mathcal{P}_{\text{sys}}$, VLM π, substep budget S
Symbols: c (actor code generated by the VLM), o (stdout from running c), I (post-execution image), r (post-execution robot state), h (within-trial trace), σ (react status flag, e.g. TASKCOMPLETE).

$h \leftarrow [\,]$
$c \leftarrow \pi_{\text{plan}}(\mathcal{P}_{\text{sys}}, \mathcal{T}, I_0)$ ▷ plan task; emit code for the first substep
for substep $k = 1, \ldots, S$ **do**
    $(o, I, r) \leftarrow \text{EXECUTE}(c;\, \mathcal{M})$ ▷ run substep code to completion
    append (c, o, I, r) to h
    $(\sigma, c) \leftarrow \pi_{\text{react}}(\mathcal{T}, c, o, I, r, h)$ ▷ Sec. 3.3: judge substep, then emit next-substep code or replanned code
    if $\sigma = \text{TASKCOMPLETE}$ **then break**
    end if
end for

depth is often noisy, we use FoundationStereo [28] to estimate more reliable **depth maps** from RGB. For **object center point**, given a language description, we employ Grounded-SAM to produce a mask and its center point, while a Gemini-based pointing tool returns single 2D points corresponding to language queries. For tasks requiring higher precision—such as grasping towel corners—we provide a **salient task-relevant points** tool inspired by ReKep [29]: after generating a segmentation mask, we overlay a uniform point grid and ask GPT-o3 to select the most relevant points. This tool hierarchy balances granularity and runtime: raw images are coarse but fast, while salient points are slower but more precise. MAESTRO autonomously reasons about which tools to use and how to use them for each task, achieving a balance between execution speed and task performance. As an alternative, we also provide **language-based pointing** using Gemini Robotics-ER 1.5 [4].

Reasoning. In our initial experiments, MAESTRO struggled with task-relevant spatial reasoning even with advanced perception tools. We therefore provide tools for explicit spatial calculations: **measuring distances**, **constructing vectors using points**, **computing relative rotations between vectors**, and **rotating a vector by specified angles**.

Control. For low-level manipulation, we provide Cartesian end-effector control tools: **move gripper to**, **open gripper**, and **close gripper**. We use cuRobo [31] for point-cloud-based motion planning

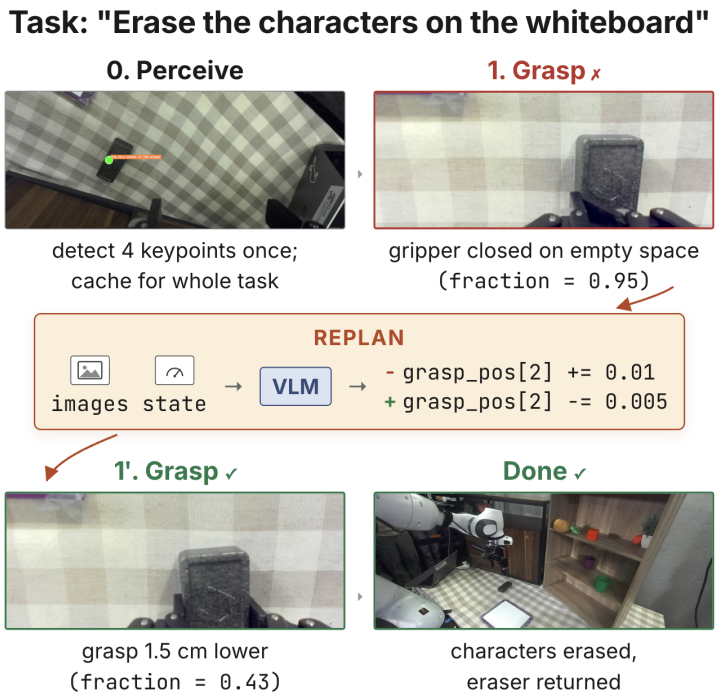


Figure 9: One replan iteration on "Erase the characters on the whiteboard". The first `actor_code` grasps 1 cm too high (top); from the post-execution image and robot state, ***replan*** rewrites a single line (`grasp_pos[2]`) and the retry succeeds, completing the task with no further replans (bottom).

with collision checks. At the grasping stage, collision checking is disabled only for the gripper links identified by the robot and gripper URDFs; checks remain active for the rest of the robot. This contact exception does not model the grasped object as part of the robot, so collisions involving held objects remain a limitation (App. J).

Learned Visuomotor Policies. We equip MAESTRO with two types of learned visuomotor policies: a **grasp model** and a **VLA**. For grasping, we provide GraspGen [30], while for general-purpose low-level visuomotor control we provide the state-of-the-art $\pi_{0.5}$ model [10]. A key challenge when integrating end-to-end VLAs as tools is determining when to interrupt their execution, since they continue running inference until externally stopped. We use a separate completion monitor running at 2 Hz, hosting Qwen-2.5-VL-72B-Instruct locally to check task completion with a yes/no question and interrupt execution when completion is detected.

Image Editing. Prior work [34, 35] shows that adding visual marks can improve the grounding and reasoning abilities of VLMs. Thus, we provide MAESTRO with image-editing tools that can **draw points** and **overlay 6D poses** on images, enabling clearer spatial references and richer visual reasoning during manipulation. Concretely, MAESTRO draws the 2D points returned by its pointing tools back onto the observation so the VLM can confirm a target selection before acting (e.g. verifying which object or contact point was chosen in spatial-reasoning and pick-place tasks), and overlays a candidate 6D grasp pose so the agent can check the gripper's approach direction against object affordances (e.g. confirming a knife is grasped by the handle, or a mug at an orientation that permits hanging). The edited image is fed back into the next replanning step, closing the loop on these checks.

A.4 Additional Modules for Mobile Manipulation

To extend beyond tabletop tasks, we equip MAESTRO with additional tools that enable mobile manipulation.

Algorithm 2 MAESTRO execution and evolution loop (full version). The inner plan-react-replan loop is the single-trial Algorithm 1 (above); the outer `for` loop over trials is the cross-trial evolution mechanism of Sec. 3.3.

Require: Task $\mathcal{T}$, initial image I_0, tool library $\mathcal{M}$, system prompt $\mathcal{P}_{\text{sys}}$, VLM π, max trials N, per-trial substep budget S
Symbols: c (actor code generated by the VLM), o (stdout from running c), I (post-execution image), r (post-execution robot state), h_n (trial-n trace), $\mathcal{H}$ (across-trial history), σ (react status flag, e.g. TASKCOMPLETE).

$\mathcal{H} \leftarrow \emptyset$ ▷ across-trial history (Sec. 3.3)
for trial $n = 1, \ldots, N$ **do**
  $h_n \leftarrow [\,]$
  $c \leftarrow \pi_{\text{plan}}(\mathcal{P}_{\text{sys}}, \mathcal{T}, I_0, \mathcal{H})$ ▷ plan task; emit code for the first substep
  for substep $k = 1, \ldots, S$ **do**
    $(o, I, r) \leftarrow \text{EXECUTE}(c; \mathcal{M})$ ▷ run substep code to completion
    append (c, o, I, r) to h_n
    $(\sigma, c) \leftarrow \pi_{\text{react}}(\mathcal{T}, c, o, I, r, h_n)$ ▷ Sec. 3.3: judge substep, then emit next-substep code or replanned code
    if $\sigma =$ TASKCOMPLETE **then break**
    end if
  end for
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{h_n\}$ ▷ persist trace as in-context history for next trial (Sec. 3.3)
end for

Perception. For a mobile manipulator, obtaining the full **robot 6D state** requires more than raw proprioception. We therefore employ Faster-LIO [36], a lightweight LiDAR–Inertial Odometry method, to provide robust state estimation of the mobile base even under visually challenging conditions. This enhanced perception ensures reliable navigation and spatial reasoning across larger workspaces. In addition, mobile manipulation benefits from active perception to build a more complete understanding of the surroundings and gather task-relevant information on demand. To this end, we provide MAESTRO with active perception tools—**look left**, **look right**, **look to the ground**, and **view carry-on basket**, **remember object location**—which allow it to scan the environment, adjust its viewpoint, and access the basket contents as needed.

Locomotion. We supply two tools: (1) the **nudge** tool, which applies small velocity adjustments for precise positioning near a target location, and (2) the **navigation** tool, which leverages Nav2[37] to move the robot safely to a target pose on the map. This dual interface allows MAESTRO to fluidly switch between global navigation and fine-grained local control.

Manipulation. To support transport tasks, we provide a dedicated **put in basket** tool that enables the robot to efficiently carry multiple objects during long-horizon mobile manipulation. For unloading, MAESTRO leverages the **view carry-on basket** perception tool to inspect the basket's contents and then automatically generates the code needed to move objects out of the basket.

B Beyond Single-Trial Execution: Evolution, VLA Tool Use, and Autonomous Data Collection

Beyond the single-trial zero-shot evaluation, this appendix presents three case studies. **Cross-trial evolution:** from the least successful initial open-cabinet run (35% progress), three rounds of video-grounded code refinement reach full task completion (App. C, Fig. 10); this single selected trajectory illustrates adaptation rather than estimating its average benefit across tasks. **VLA as a tool:** the default library includes $\pi_{0.5}$ [10] for short pick-and-place skills. On *clean up the table, largest object first*, using it raises average progress from 89% to 94% and reduces wall-clock completion time by $2.3\times$ while retaining MAESTRO's high-level planning (App. K). **Autonomous data collection:** on two Tether [38] tasks, MAESTRO supplies autonomous seed demonstrations with manual resets. The

reported success rates are comparable to human-teleoperated Tether on these tasks (95% versus 100% on *Bowl from Shelf*, and 88% on *Wiping with Cloth*); Fig. 15 and App. L give the comparison.

C Evolution Across Trials: Open-Cabinet Case Study

This appendix expands the evolution result summarized in App. B, tracing the trial-by-trial code evolution that the outer loop of Algorithm 2 (App. A) produces on the open-cabinet task.

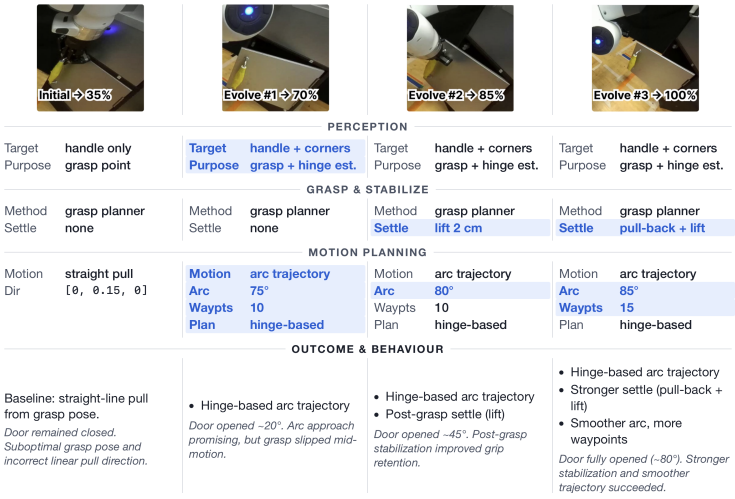


Figure 10: **Self-evolution of MAESTRO's generated policy on the "open cabinet door" task.** Top: representative frames with completion progress. Middle: semantic code abstraction with changed parts highlighted in blue. Bottom: strategy changes and outcome per trial. Across four autonomous trials, the agent evolves from a naive straight-line pull to a hinge-based arc trajectory with grasp stabilization, achieving full task completion.

We present a case study of MAESTRO's evolution mechanism (the outer loop of Algorithm 2, App. A) on the open-cabinet task. The trajectory starts from the least successful initial run (35% progress). Fig. 10 shows the policy abstraction, strategy change, and outcome at each refinement stage.

Trial 0 (Initial, 35%). MAESTRO identified the handle but used a top-down grasp and a fixed straight-line pull ([0.0, 0.15, 0.0]) that ignored the hinge kinematics, leaving the door closed. Reflection correctly attributed the failure to the linear motion primitive.

Trial 1 (Evolve #1, 70%). MAESTRO restructured the motion strategy: it added perception of the top-right and bottom-right door corners to estimate the hinge axis, then replaced the straight pull with a hinge-arc trajectory (75°, 10 waypoints). The door began to open, but the handle slipped mid-arc because the grasp could not sustain the rotational forces.

Trial 2 (Evolve #2, 85%). To counter the slip, MAESTRO added a post-grasp stabilization step (a 2 cm upward lift to tighten the grip), widened the arc to 80°, and refined the rotation to incremental orientation updates in the 2D plane. The door reached about 45° before the handle slipped again.

Trial 3 (Evolve #3, 100%). MAESTRO replaced the lift with a combined retract-and-lift (2 cm + 2 cm) that pre-tensions the grasp, widened the arc to 85°, increased waypoints from 10 to 15 (smoother, lower-jerk trajectory), and added incremental yaw tracking so the gripper follows the arc. These changes eliminated grasp slip and the door opened to about 80°.

Across four autonomous trials, MAESTRO progressed from a linear pull to a kinematically grounded arc policy with robust grasp stabilization, showing that video-grounded self-reflection enables improvements in generated robot programs across trials without human guidance.

D Extended Experimental Setup

Embodiments. Our tabletop platform follows the DROID setup [27]: a 7-DoF Franka Emika Panda arm with a Robotiq 2F gripper, a wrist-mounted camera, and a third-person camera. Our mobile platform is a Unitree Go2-W wheeled quadruped with a top-mounted AgileX Robotics PiPER manipulator arm and a calibrated wrist-mounted camera for egocentric perception.

The STAR-Gen perturbation protocol and per-task scoring rubric used in evaluation are detailed in App. E below.

E Task Evaluation Rubric

We performed rigorous evaluation on all tasks following STAR-Gen [32] to generate new trial for zero-shot generalization capabilities. 5 trials each task (1 initial setup following 4 generated by STAR-Gen), and for each STAR-Gen generated trial, vary all of the following:

- Object placement
- Object instance
- Scene / Lab setting
- Language instruction (paraphrase)

We use a task-specific progress rubric on a 0–100 scale, assigning credit to verifiable substeps to distinguish how far the agent progressed in incomplete long-horizon trials and identify where it failed. Human reviewers assigned every task-progress and task-completion score independently of the agent's monitor.

The evaluation metrics along with STAR-Gen variations for table-top manipulation tasks are detailed below:

1. Pick-place (*"Put item in bowl."*)
   [25%] Approach the item.
   [50%] Grasp the item.
   [60%] Lift up the item.
   [75%] Approach the target location.
   [100%] Place correctly.
2. Deformable object (*"Fold the four corners of the towel into the center."*)
   For this task, STAR-Gen introduces significant geometric and interactive variability across trials (e.g., varying object shape and location), we could not rely on one fixed metric. Instead, the task completion rubric was dynamically fine-tuned for each generated trial, ensuring a fair evaluation of progress for sub-goals of each scenario.
   (a) *"Fold the four corners of the towel into the center."* (color of the towel is different between two trials)
   [25%] One corner folded in.
   [50%] Two corners folded in.
   [75%] Three corners folded in.
   [100%] All four corner folded in successfully.
   (b) *"Fold the T-shirt into a rectangle."*
   [15%] One t-shirt sleeve folded in.
   [30%] Both t-shirt sleeves folded in.
   [60%] Bottom of t-shirt folded in.
   [100%] Fold entire t-shirt inward successfully.
   (c) *"Place one corner of the towel to its diagonal corner."*
   [30%] Approach one corner.

[60%] Grasp and lift up the corner.
[100%] Place at the diagonal corner successfully.

(d) *"Unfold the towel into a square."*
[30%] Accurately approach the folded corner.
[60%] Grasp and lift up the corner.
[100%] Place at the table to finish unfolding.

3. Articulated object (*"Open cabinet."*)
[10%] Approach the cabinet.
[40%] Grasp the handle.
[60%] Attempt to pull.
[70%] Open the door slightly.
[100%] Open the door completely.
4. Spatial reasoning (*"Rotate cube purple side up."*)
[10%] Approach the cube.
[30%] Grasp the cube.
[60%] Rotate the cube.
[100%] Purple side faces up.
5. Tool use (*"Cut banana with knife."*)
[20%] Approach the knife.
[50%] Grasp and lift up the knife.
[60%] Approach the banana.
[80%] Position in a good cutting orientation.
[100%] Cut the banana.
6. Object affordance (*"Hang mug on mug holder."*)
[25%] Approach the mug.
[50%] Grasp and lift up the mug.
[75%] Approach the mug holder.
[100%] Hang on to the stand bar.
7. Memory & long-horizon & semantic (*"Erase instructions on whiteboard, then follow instruction to stack cubes."*)
[20%] Pick up the eraser.
[30%] Erase the white board partially.
[50%] Erase the white board completely.
[60%] Return the eraser.
[80%] Stack the second cube on top of the first cube.
[100%] Stack the third cube on top of the second cube.

The evaluation metrics for mobile manipulation tasks are detailed below:

1. Long-horizon manipulation (*"Collect all toys on table."*)
[25%] Collect one toy.
[50%] Collect two toys.
[75%] Collect three toys.
[100%] Collect all four toys.
2. Long-horizon loco-manipulation (*"Throw the ball into garbage can."*)
[16.7%] Find the ball.
[33.3%] Move to the ball.
[50%] Pick up the ball.
[66.7%] Find the trash can.
[83.4%] Move to the trash can.
[100%] Drop the ball into the trash can.
3. Active exploration (*"Search item and return when grasped."*)
[20%] Explore around the area.

[40%] See the object.
[60%] Move to the object.
[80%] Grasp the object.
[100%] Return to the initial position.

4. Object affordance (*"Press button to open door."*)
[33.3%] Identify the correct label.
[66.6%] Approach to the correct button.
[100%] Press button to open the door.

F Extended discussion on prior work

A wide range of prior work has explored the use of large models — Vision-Language Models (VLMs) and Large Language Models (LLMs) — to support specific components of robotic systems. In the common paradigm, these models are embedded into manually designed, typically rigid, modular pipelines, where large models take on one or several well-defined roles while the remaining pipeline is built through "good old-fashioned engineering" by humans. These roles span reward and environment design [39–43], constraint-function design [29, 44–46], high-level planning [16, 47], self-reflection [48], visual question answering [35, 49–51], in-context learning [52, 53], and task-progress evaluation [54–56].

While effective within their intended scope, these approaches remain constrained by their rigid workflows. Because large models automate only a small portion of the pipeline, substantial manual effort is still required to design the rest. This rigidity also limits scalability: systems tuned and fixed for specific settings struggle to generalize to diverse, in-the-wild scenarios. As a result, these methods fall short of the requirements for scalable, general-purpose robotic manipulation.

Rather than reworking the agentic architecture, MAESTRO builds on the closed-loop VLM coding-agent pattern of recent CaP-based manipulation systems such as Gemini Robotics [2, 4, 33]. Our contribution is the design of the harness around that shared pattern: rather than relying on the VLM for perception, MAESTRO composes curated external robotics models as callable tools (Table 1), letting the system inherit ongoing advances in those subfields without retraining. The result is a system that can autonomously decide *how*, *when*, and *which* modules to employ. In doing so, we show that, alongside the dominant data-scaling route to generalist manipulation, investing in this kind of agentic system is a complementary axis that already yields VLA-level performance without new teleoperation data.

Hierarchical and reasoning-augmented VLAs. Several VLA variants augment the action-generation backbone with language or reasoning while still selecting actions through a fine-tuned VLA: Hi Robot [8] decomposes user instructions into natural-language sub-commands that a learned VLA executes, and ECoT-Lite [7] distills efficient embodied chain-of-thought reasoning directly into the VLA.

Concurrent code-as-policies efforts. Several recent efforts explore approaches related to MAESTRO. CaP-X [25] from NVIDIA proposes a benchmark and framework for code-as-policies coding agents, validated on a Franka arm in simulation and on real hardware, and shares our focus on multi-turn interaction, structured feedback, and skill synthesis. GaP [26] uses a multi-agent coding harness to construct graph policies from modular robot skills and refine them through simulation rehearsal for variational automation. Gemini Robotics 1.5 [4] introduces an ER/VLA stack with generic third-party tool-calling, and Gemini Robotics-ER 1.6 [33] adds Python-based image manipulation for ambiguity resolution. We view this convergence as encouraging evidence that the route is worth pursuing; MAESTRO's complementary scope is its curated, robotics-specific module library and the orchestration harness built around composing those models in a real-world generalist system across diverse tasks and embodiments.

G Detailed Tabletop Manipulation Results

This section provides additional details on the tabletop manipulation results summarized in Fig. 4 of the main text. Table 2 reports the mean and standard error of task progress scores across all five evaluation trials for each task and method.

Qualitative analysis. VLM-based agents excel at high-level semantic reasoning and planning. However, our CaP baseline, the Gemini Robotics Agent, still struggles to turn these high-level plans into effective low-level actions. Its agentic loop is designed around Gemini's own perception capabilities [2, p. 11], so on harder tasks the agent is constrained by whatever bounding boxes, points, and top-down grasps the VLM can produce on its own. In open cabinet, this means the agent cannot localize the handle without an active-perception scan, and its top-down grasping strategy no longer applies. Similar limitations show up in fold towel and erase whiteboard, where the agent can plan but cannot localize towel corners or correctly grasp and orient the eraser without external segmentation, depth, and grasp-prediction modules in the loop.

MAESTRO bridges this gap by coupling VLM-level semantic reasoning with a broad suite of specialized tool modules. It leverages grasp models for reliable pick-and-place, active perception modules for accurate point-cloud reconstruction of cabinet handles, task-relevant keypoints for towel corner localization, and geometric reasoning to rotate objects to precise orientations (e.g., turning a cube to place a target color face upward). When errors occur, MAESTRO's "react and replan" mechanism analyzes execution history and images, then revises its code and module orchestration. For example, to pick up the tennis ball on the shelf, it first attempts a top-down grasp but, after detecting rolling and collision risks, switches to active perception to refine the point cloud and leverages the grasp model tool to generate a safer, reachable grasp pose. Nevertheless, the "hang mug on mug holder" task reveals MAESTRO's current limitations: when complex chain-of-thought reasoning about object affordances and spatial orientation is required, it can still struggle. In this case, it fails to correctly align the mug handle's hole with the branch orientation of the holder.

Table 2: Tabletop manipulation results: mean $\pm$ standard error of task progress (0–100; higher is better) across methods and tasks.

Challenge	**Task Description**	**Gemini Robotics Agent**	π_0	$\pi_{0.5}$	**MAESTRO**
Pick–Place	Put item in bowl	83.6 ± 15.9	74.0 ± 16.6	70.0 ± 18.4	$\mathbf{98.0 \pm 2.0}$
Deformable object	Fold the four corners of the towel into the center	29.0 ± 8.7	47.0 ± 11.2	$\mathbf{70.0 \pm 6.9}$	68.0 ± 8.9
Articulated object	Open cabinet	2.0 ± 2.0	5.0 ± 2.2	2.4 ± 2.4	$\mathbf{68.0 \pm 14.0}$
Spatial reasoning	Rotate cube purple side up	52.0 ± 16.5	23.0 ± 15.0	30.0 ± 17.0	$\mathbf{88.0 \pm 5.6}$
Tool use	Cut banana with knife	71.0 ± 12.9	29.6 ± 10.7	14.0 ± 2.9	$\mathbf{92.0 \pm 2.5}$
Object affordance	Hang mug on mug holder	46.0 ± 10.4	59.0 ± 13.7	$\mathbf{80.0 \pm 6.3}$	69.0 ± 4.3
Long-horizon & semantic	Clean up the table, largest object first	80.4 ± 3.0	18.0 ± 6.6	18.2 ± 9.8	$\mathbf{93.2 \pm 4.7}$
Memory & long-horizon & semantic	Erase instructions on whiteboard, then follow instruction to stack cubes	40.0 ± 11.4	12.0 ± 5.4	22.0 ± 10.2	$\mathbf{63.0 \pm 7.5}$

Fig. 11 visualizes every individual trial of our tabletop manipulation experiment. For each task, the figure shows the original trial setting alongside the semantic perturbations introduced by STAR-Gen [32], which systematically varies object instances, object poses, scene backgrounds, and language instructions across trials. It also shows the resulting performance of each method on every trial. This per-trial visualization provides a more granular view than the aggregate statistics in Fig. 4 and Table 2, revealing how each method responds to specific types of variation.

Across the evaluated settings, MAESTRO has the highest mean progress on six of eight tasks, while trial-level outcomes vary by task and setting. In contrast, π_0 and $\pi_{0.5}$ exhibit high variance across trials, with their markers scattered widely along the scale bar. In the *Pick–Place* column (leftmost), the VLA baselines achieve perfect scores in some evaluated settings but substantially lower scores in

Table 3: Mobile manipulation results: mean $\pm$ standard error (SE) of task progress (0–100; higher is better) for MAESTRO, with $n = 5$ trials per task. SEs are computed from the reported rounded SDs as $\mathrm{SE} = \mathrm{SD}/\sqrt{5}$.

Task Category	**Task Description**	**MAESTRO**
Long-horizon manipulation	Collect all toys on table	85.0 ± 10.0
Long-horizon loco-manipulation	Throw green ball into garbage can	76.7 ± 6.7
Active exploration	Search for item and return	96.0 ± 4.0
Object affordance	Press button to open door	93.3 ± 6.7

others. This variation is reflected in their larger standard errors in Table 2: ± 16.6 and ± 18.4 for π_0 and $\pi_{0.5}$, compared to ± 2.0 for MAESTRO. On open cabinet, the baselines have low mean progress (2.0–5.0) and small SEs (2.0–2.4), reflecting uniformly low progress. On rotate cube, MAESTRO achieves 88.0 ± 5.6, compared with 52.0 ± 16.5 for the Gemini Robotics Agent, 23.0 ± 15.0 for π_0, and 30.0 ± 17.0 for $\pi_{0.5}$; its mean is higher and its SE is lower than those of the evaluated baselines.

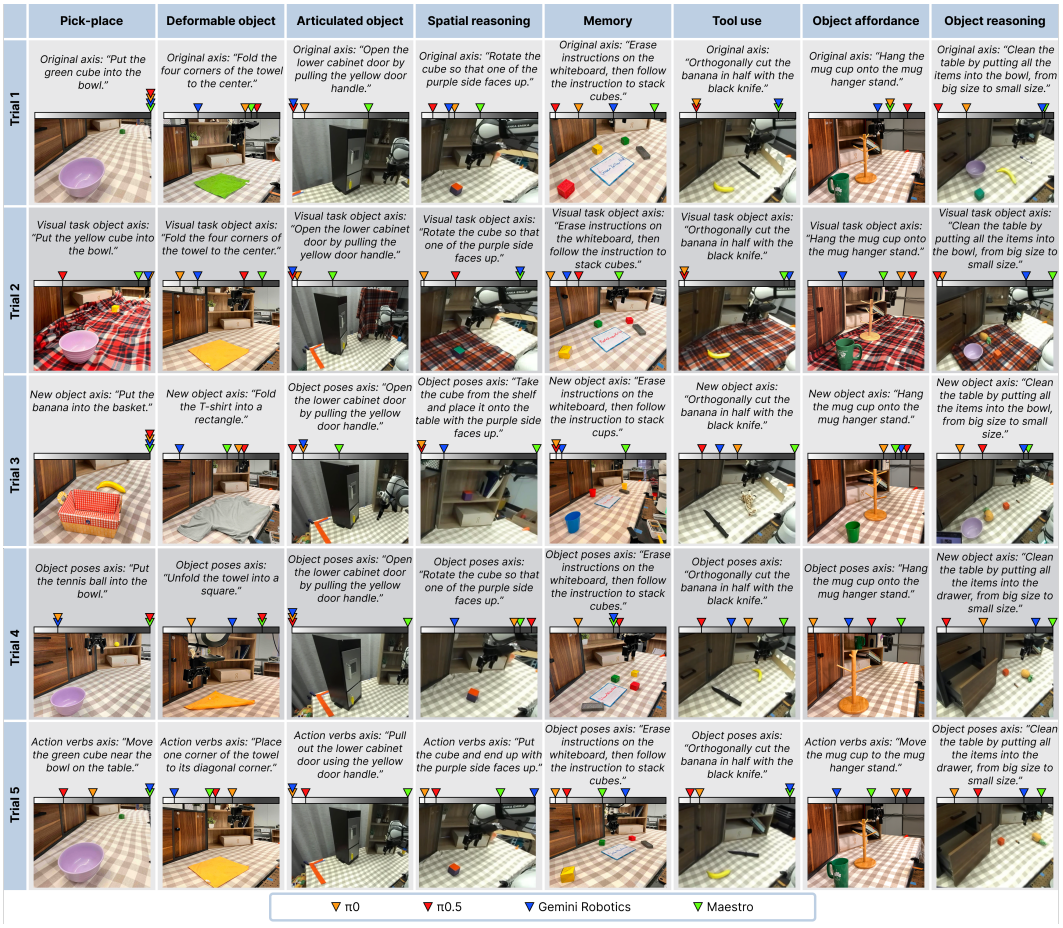


Figure 11: Per-trial visualization of tabletop manipulation results. Each row corresponds to a task and shows the original trial setting alongside the five STAR-Gen evaluation trials. Above each trial image, a gradient scale bar ranges from white (0 progress) on the left to black (100 progress) on the right. Colored triangular markers (▼) on this bar denote each method's score on that trial, with colors corresponding to the methods indicated in the legend.

H Leave-One-Out Module Ablations

This section reports the per-module leave-one-out ablations referenced in Sec. 4.4. Whereas the main-text scaling study *adds* tool groups on top of Base CaP, here we start from the full MAESTRO and *remove* one module category at a time, keeping all other components fixed.

While Table 1 details the comprehensive list of improvements and additions we made over Base CaP, here we focus on two key categories of design choices among them. **MAESTRO w/o advanced perception** evaluates the impact of the *task-relevant keypoint* and *active perception* modules; **MAESTRO w/o geometry modules** tests the impact of the *geometry* and *linear algebra* modules. For systematic comparison, we fairly isolate each factor by following a "change one at a time" approach, creating variants where only one module category is reverted to the Base-CaP baseline while keeping all other components identical to MAESTRO.

Task. We use the "**fold towel**" and "**rotate cube**" tasks. In fold towel, the robot must fold the corners into the center, so the task requires reasoning about object geometry and specific points of interaction on the object. In rotate cube, the robot must rotate the cube until a particular color side faces up, requiring extensive spatial reasoning about rotation and object affordance.

Table 4: Leave-one-out ablation on the Fold Towel and Rotate Cube tasks. Values are mean task progress out of 100 $\pm$ standard error (SE), with $n = 5$ trials per method and task and $\mathrm{SE} = \mathrm{SD}/\sqrt{5}$. The full MAESTRO baseline reuses Table 2; ablation SEs are calculated from the reported rounded SDs.

Method	Fold Towel	Rotate Cube
MAESTRO	68.0 ± 8.9	88.0 ± 5.6
MAESTRO w/o advanced perception	40.0 ± 3.2	25.0 ± 0.0
MAESTRO w/o geometry modules	67.5 ± 1.6	42.5 ± 14.2

Results. Table 4 indicates task-dependent effects of the removed module categories. Removing advanced perception reduces the mean score on fold towel, which requires identifying and interacting with towel corners. The geometry removal has little effect on fold towel in these trials, but reduces mean progress on rotate cube. Both rotate-cube ablations still achieve partial progress. These results support the usefulness of the modules for the evaluated tasks.

I Tool-Use Trace Analysis

To verify that MAESTRO does more than execute a single canonical pipeline, we logged MAESTRO's tool calls during its tabletop trials. We recorded 2,062 tool calls drawn from 18 modules across 62 trials spanning 9 task categories, summarized in Fig. 12 and rendered as a full passmap in Fig. 13. The per-task call graphs (Fig. 14) show that dominant tool sequences differ across task categories and are revised after failures. This pooled analysis includes the VLA-as-tool study in App. K. An interactive version of the passmap, with task-level filtering, free-form node layout, and saved views, is hosted on our project site.

Workflows are task-conditioned, not universal. The most-called modules and the dominant transitions differ across categories: deformable tasks rely on `estimate_feature_point` cycles for keypoint re-tracking; articulated tasks chain `naive_pointing` $\rightarrow$ `get_object_bbox` $\rightarrow$ `plan_grasp` $\rightarrow$ `move_gripper_to`; long-horizon tasks add `rescan_wrist` $\rightarrow$ `get_image` re-perception cycles between subtasks. Fig. 14 makes this contrast visible at the graph level: it is the graph topology, not just the call counts, that changes with the task, so MAESTRO composes a task-specific workflow rather than selecting a fixed pipeline.

Replanning produces observable in-trial corrections. Beyond cross-trial improvement (App. B), the passmap also exposes within-trial corrections, edges that exist only after a failed subtask attempt. Across the 71 trials in this analysis, 47 of 63 initially failed subtask instances later received a logged success judgment (74.6%). This conditional, log-derived statistic is distinct from human-scored trial completion; Table 5 reports the per-task breakdown. The recovered "erase and stack" trial, in which MAESTRO detects an incomplete erase and re-enters the wiping subroutine before continuing, is traced at the tool-call level below.

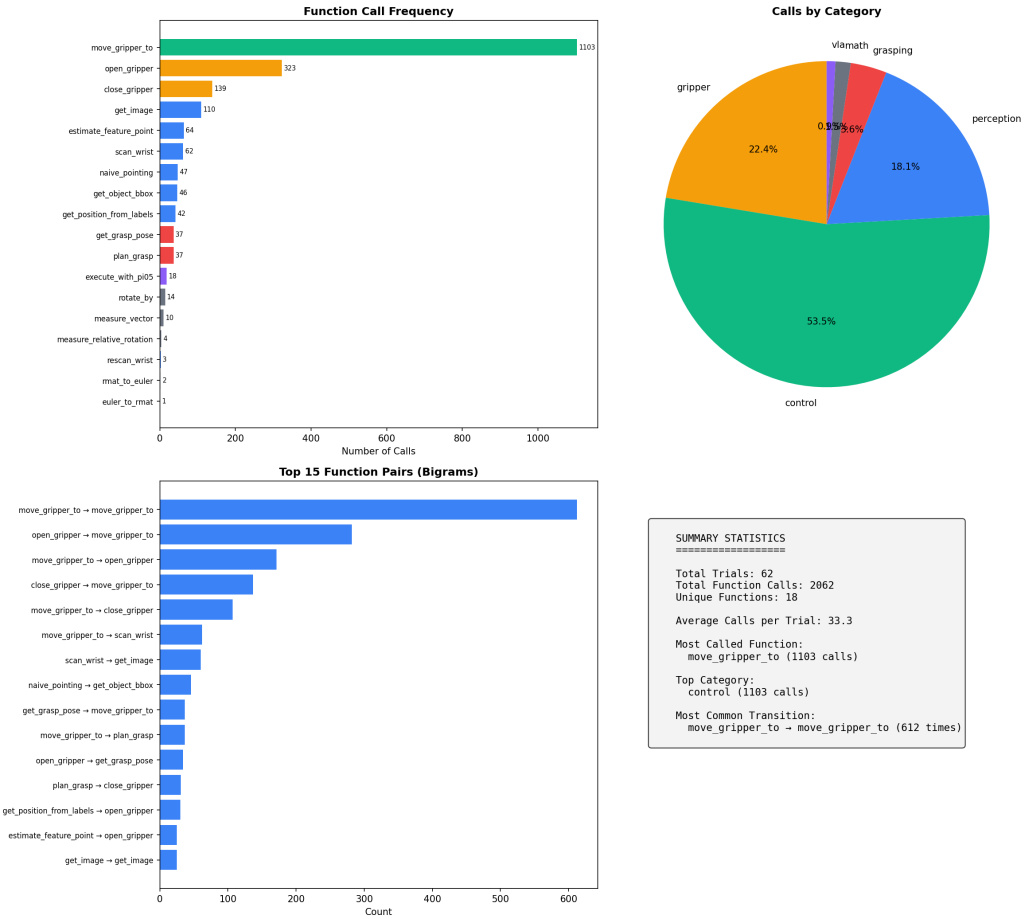


Figure 12: **Aggregate tool-use statistics across 62 MAESTRO tabletop trials** (2,062 calls across 18 modules in 6 categories). Top left: per-module call frequencies. Top right: calls grouped by module category. Bottom left: top function bigrams. The full interactive passmap with per-task filtering is hosted on our project site.

Table 5: Per-task rates of later logged success after an initial failure. For each task category, we report the number of trials, the number of subtask instances whose first attempt failed (X), the subset with a later logged success judgment (Y), and the conditional rate Y/X. A "subtask" is a unique step index within a trial; inclusion in Y requires a later attempt of the same step index in the same trial to reach `Success: True`. These log-derived judgments are distinct from human-scored trial completion.

Task category	**Trials**	**First-attempt failures** (X)	**Later logged success** (Y)	**Rate** (Y/X)
Pick-place	7	2	1	50.0%
Deformable	6	3	3	100.0%
Articulated	5	2	2	100.0%
Tool use	7	8	4	50.0%
Spatial reasoning	9	14	12	85.7%
Object affordance	5	4	2	50.0%
Long-horizon	10	15	11	73.3%
Long-horizon (with VLA)	6	7	6	85.7%
Memory & long-horizon	16	8	6	75.0%
Total	**71**	**63**	**47**	**74.6%**

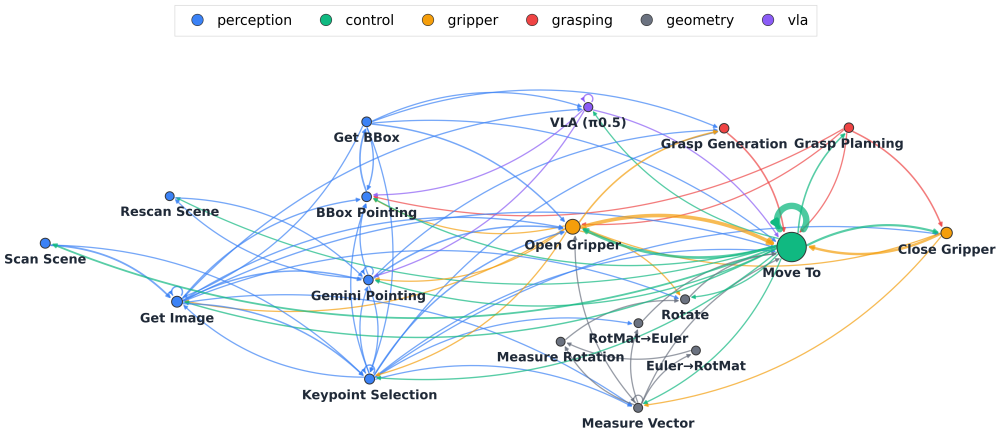


Figure 13: **Full MAESTRO tool-use passmap** aggregated across 62 tabletop trials (2,062 calls, 18 modules). Nodes are tool modules color-coded by category (perception, control, gripper, grasping, geometry, VLA); node size is proportional to call count and edge width to transition frequency. The interactive version on our project site supports per-task filtering, drag-and-drop layouts, and highlighting of common workflows.

Trace case study: "erase instructions on the whiteboard, then stack cubes" with recovery. MAESTRO's opening tool sequence is `scan_scene` → `get_image` → `naive_pointing` → `plan_grasp` → `move_gripper_to` for the eraser. After the first wipe pass, the post-execution reflection (cued by the wrist-camera video) flags residual ink, and MAESTRO re-enters the wiping subroutine (`rescan_wrist` → `get_image` → `naive_pointing` → `move_gripper_to`) before transitioning to the stacking subtask. The corresponding rollout, including the reflection prompt and the revised plan, is available in the Agent Dashboard on our project site ("Erase (with replan)").

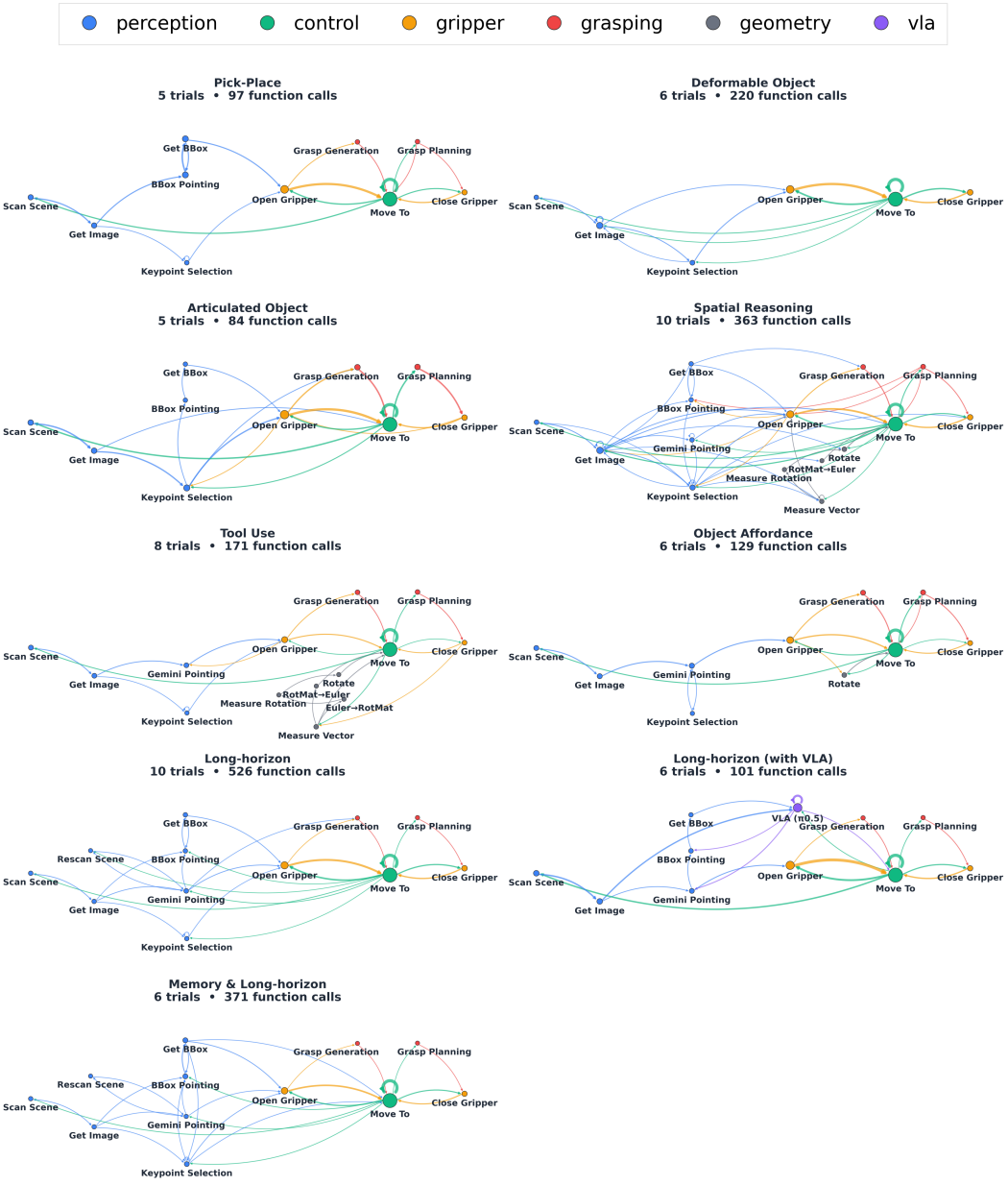


Figure 14: **Per-task tool-use passmap thumbnails.** Each panel shows the call graph for one task category (e.g. pick-place, deformable, articulated, long-horizon-with-VLA); node size encodes call count and edge width encodes transition frequency. The dominant edges and most-called modules differ across categories, illustrating that MAESTRO composes task-specific workflows rather than selecting a fixed pipeline. Full-resolution per-task passmaps and an interactive viewer with task-level filtering are available on our project site.

J Detailed Failure-Mode Analysis

MAESTRO's failures concentrate in a few module classes rather than in high-level planning: imprecise pointing and segmentation under difficult lighting, motion-planning infeasibility and over-conservative collision checking, geometrically valid but task-inappropriate grasps, and, on the mobile platform, occasional default navigational fallbacks. This appendix enumerates the full per-embodiment failure modes, with concrete trial examples, below.

The Agent Dashboard on our project site provides interactive trial replays that show how some of these failure modes surface in MAESTRO's plan-react-replan loop, including a recovered case where MAESTRO detects an incomplete erase and replans autonomously.

Mobile platform

1. **False Negatives.** In *collect all toys*, YOLO never detected the strawberry plushy, so the system reported the task 100% complete.
2. **Color Confusion.** In *collect all toys*, the system misidentified a red plushy as pink.
3. **Lighting and Motion Sensitivity.** In *press button*, motion blur combined with sunlight produced a "bad image" for the pointing module, sending the robot to the wrong position when approaching the door button.
4. **Pointing Precision.** In *throw garbage*, the VLM did not point to a reasonable location on the garbage can, so the dropped object landed half-on the rim.
5. **Global-to-Local Viewpoint Selection.** The agent struggles to switch from macro-navigation (using the global 3D semantic map) to micro-manipulation. In *throw garbage*, the initial observation correctly noted that the garbage can was on the floor, but on arrival the VLM defaulted to a "FRONT" camera pose and never pitched "DOWN," so the short can was invisible at execution time.
6. **Invalid Pose Generation.** The kinematic solver occasionally fails to produce a feasible end-effector orientation. In *throw garbage*, no valid release pose was found at the end of the trajectory, preventing the robot from dropping the green plush ball into the can.
7. **Default Navigational Fallback.** When given new action verbs, the VLM falls back on default navigational behaviors instead of adapting to the verb's meaning. In Task 3 Trial 5, the instruction was modified to "hand it to the nearest person"; rather than visually searching for a human, the VLM returned to map origin $[0, 0, 0]$.

Tabletop platform

1. **Incomplete Collision Modeling of Grasped Objects.** cuRobo [31] models the arm but not the grasped object, so the held object can still collide with surroundings in clutter even though the arm itself stays clear.
2. **Pointing Inaccuracy.** Both VLM-based pointing (Gemini Robotics-ER 1.5) and segmentation-based pointing (Grounded-SAM2) lack the precision to localize fine-grained regions reliably, especially when multiple points are needed. VLM-based pointing also confuses certain color pairs (red/yellow, blue/yellow); segmentation-based pointing degrades under bright lighting.

   When a perception error is visible in the returned observations, MAESTRO can acquire a new view, repeat perception, and replan. This does not cover undetected errors, which can propagate into later actions. The tool interface allows improved perception modules to replace existing ones without retraining the orchestration policy.
3. **Motion Planning Infeasibility.** VLM-selected end-effector poses are sometimes kinematically unreachable or unsolvable by cuRobo. cuRobo's collision checker is also over-conservative, flagging intended contacts (e.g., cutting, insertion) and blocking the interaction.
4. **Insufficient Semantic Grasp Selection.** GraspGen [30] returns geometrically valid grasps but does not reason about task affordances, so the robot may grasp a knife by the blade or a mug by an orientation that prevents the planned hang.
5. **Limited Spatial Reasoning for Complex Affordances.** Tasks requiring precise relative-orientation reasoning, such as hanging a mug on a holder, remain hard: the VLM fails to align the handle opening with the holder branch and to leave clearance along the insertion direction, often knocking the holder over. The available geometry and vector tools are not consistently leveraged for these multi-step alignments.

6. **VLM Inference Latency.** Each plan-react-replan iteration calls the VLM, making MAESTRO considerably slower than end-to-end VLA policies such as $\pi_{0.5}$. Reducing VLM inference latency and overall compute cost remains an important direction for future work.

J.1 Latency breakdown

Table 6: Component latency aggregated across a mixture of main and VLA experiments. The listed tool calls account for approximately 47% of total experiment wall time; the breakdown is not exhaustive, with remaining time including robot execution, waiting, logging, and other operations. Counts denote tool invocations, not independent trials; initial planning and replanning subdivide VLM planning and are not counted twice.

Component	Total (min)	Wall (%)	Calls	Mean (s)	Min (s)	Max (s)
VLM planning	152.8	22.1	570	16.1	5.1	77.6
Initial planning	21.1	3.1	61	20.8	9.6	40.3
Replanning	131.7	19.1	509	15.5	5.1	77.6
GroundedSAM segmentation	104.6	15.1	1433	4.4	0.9	24.8
Keypoint picking	37.9	5.5	59	38.5	8.6	124.8
Gemini pointing	14.3	2.1	41	20.9	3.7	39.4
$\pi_{0.5}$ policy rollout	13.7	2.0	17	48.4	22.1	69.8
Qwen success check	0.6	0.09	151	0.25	0.24	0.32
Motion planning	0.4	0.1	37	0.6	0.0	0.8
Grasp generation	0.3	0.0	74	0.3	0.0	0.7

K MAESTRO with a VLA Tool

This appendix gives the full setup, baselines, and per-method results for the VLA-as-tool experiment summarized in App. B, isolating the contribution of $\pi_{0.5}$ as a callable pick-and-place module within MAESTRO's toolset.

Setup. MAESTRO's default tool library includes $\pi_{0.5}$ as a callable pick-and-place module. This study evaluates its use within MAESTRO; the main-text experiments do not invoke it. To isolate the contribution of this module, we evaluate on **clean up the table, largest object first**, a long-horizon task that combines repeated pick-and-place steps with size-based semantic ordering. We compare MAESTRO with its full toolset (**MAESTRO w/** $\pi_{0.5}$) against an ablation that does not have access to $\pi_{0.5}$ (**MAESTRO w/o** $\pi_{0.5}$) and must execute every sub-skill through its own primitive controllers. Table 7 reports average task progress and average wall-clock completion time.

Table 7: Evaluation of MAESTRO with and without $\pi_{0.5}$ as a callable pick-and-place tool, on the *clean up the table, largest object first* task. Reported are average task progress (0–100; higher is better) with ± 1 standard error across five trials per method ($\mathrm{SE} = \mathrm{SD}/\sqrt{5}$, in percentage points), and average wall-clock completion time.

Method	**Progress (%)**	**Time**
MAESTRO w/o $\pi_{0.5}$	89 ± 5.1	$11' 18''$
MAESTRO w/ $\pi_{0.5}$	94 ± 4.9	$4' 59''$
$\pi_{0.5}$	17 ± 9.1	$2' 16''$
π_0	21 ± 6.6	$2' 58''$
Gemini Robotics	80 ± 3.0	$11' 53''$

Results. π_0 and $\pi_{0.5}$ alone reach only 21% and 17% progress, as neither can sustain the size-based ordering across the task. MAESTRO w/o $\pi_{0.5}$ already reaches 89%, and MAESTRO w/ $\pi_{0.5}$ further improves to 94% while cutting completion time from $11' 18''$ to $4' 59''$ (a $2.3\times$ speedup). The gain comes from MAESTRO offloading short pick-and-place steps to $\pi_{0.5}$'s feedforward inference while

retaining its own long-horizon planning and reactive replanning. VLAs and MAESTRO are thus complementary: VLAs serve as fast, reliable skill modules, and MAESTRO provides the orchestration to deploy them on tasks they cannot solve in isolation. Rollout videos for this experiment are available on our project site.

L Autonomous Seed Collection: Tether Case Study

This section provides additional background on the Tether framework [38] used in our autonomous data collection experiments (App. B).

Tether is a framework for scaling robotic demonstration data that amplifies a small number of seed demonstrations into a large, diverse dataset. The key insight is *correspondence-driven trajectory warping*: given a seed demonstration recorded in one scene configuration, Tether extracts a small set of task-critical waypoints from the seed (end-effector poses at frames where the gripper open/close state toggles) and uses DINOv2/Stable-Diffusion-based semantic correspondences to locate the matching waypoints in a new scene (e.g., with different object poses or instances). The 2D matches are backprojected to 3D via stereo, yielding per-waypoint displacements that are linearly interpolated along the seed trajectory to warp the full end-effector path (timing rescaled to preserve speed), producing a new demonstration that accomplishes the same task under different conditions. Starting from as few as 10 seed demonstrations, Tether can generate large datasets of diverse, high-quality trajectories across multiple tasks. We refer the reader to [38] for full details on the correspondence computation and warping procedure.

In our experiments, we replace Tether's human-teleoped seed demonstrations with trajectories autonomously generated by MAESTRO. This removes the need for human teleoperation in the seed collection stage, leaving only manual environment resets as the remaining human involvement: between rollouts, a human resets the workspace with deliberately varied object placements (and, where applicable, object instances), and Tether's warping adapts the MAESTRO-generated seed to each new configuration, so the augmented dataset grows by one warped trajectory per reset. The resulting pipeline demonstrates that MAESTRO-generated demonstrations are of sufficient quality to serve as effective seeds for downstream data augmentation.

We detail below the autonomous data-collection experiment summarized in App. B.

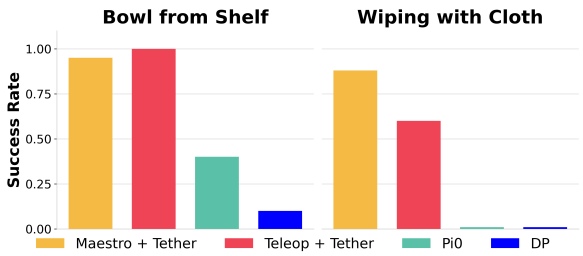


Figure 15: **Autonomous seed collection for Tether [38]: a two-task case study.** We compare *Bowl from Shelf* and *Wiping with Cloth*. MAESTRO + Tether uses 10 autonomous seed demonstrations; Tether uses 10 human-teleoperated seeds, and Diffusion Policy [57] trains on Tether's human-teleoperated data. The π_0 [1] baseline is evaluated zero-shot.

Setup. We evaluate on two tasks from the Tether benchmark: *Bowl from Shelf* (retrieving a bowl from a shelf) and *Wiping with Cloth* (wiping a surface with a cloth). We compare four methods: (1) **MAESTRO + Tether**, where MAESTRO autonomously generates 10 seed demonstrations that Tether then warps; (2) **Tether**, where a human teleoperates 10 seed demonstrations for Tether to warp; (3) π_0 [1], a state-of-the-art VLA run zero-shot; and (4) **Diffusion Policy (DP)** [57], trained on Tether's human-teleoped demonstrations.

Results. As shown in Fig. 15, on these two tasks MAESTRO + Tether achieves performance comparable to human-teleoped Tether (e.g., 95% vs. 100% on Bowl from Shelf, and 88% on Wiping with Cloth), while both significantly outperform π_0 and Diffusion Policy. Critically, MAESTRO + Tether requires *no human teleoperation* — MAESTRO autonomously generates the seed demonstrations, making the entire data collection pipeline fully automatic aside from environment resets. This result is a first indication that MAESTRO-generated demonstrations can plausibly substitute for human teleoperation when fed into data-augmentation frameworks like Tether, and suggests that MAESTRO could potentially be useful as an autonomous data source for downstream policy learning more broadly. See our project site for videos of the Tether data collection experiments.

M Monitor Reliability Audit

We audited the main VLM replanning judgments against human review of the corresponding trial videos. The audit contains 61 trials with usable logs and 509 logged judgments. We excluded 137 post-completion idle judgments, yielding 372 evaluated judgments; the exclusion rule omits steps with no robot command and includes 26 reviewer overrides.

Treating a claim of step success as positive, the counts are 204 true positives, 36 false positives, 130 true negatives, and 2 false negatives. Accuracy is $334/372 = 89.8\%$, success precision is $204/240 = 85.0\%$, and success recall is $204/206 = 99.0\%$. Among positive judgments asserting task completion, $22/55 = 40.0\%$ are incorrect; among positive intermediate-step judgments, $14/185 = 7.6\%$ are incorrect.