FLY-BY-CODE: EMBODIED CODING AGENTS FOR AERIAL MANIPULATION WITH ACTIVE VISUAL AND PHYSICAL FEEDBACK

Anonymous authors
Paper under double-blind review

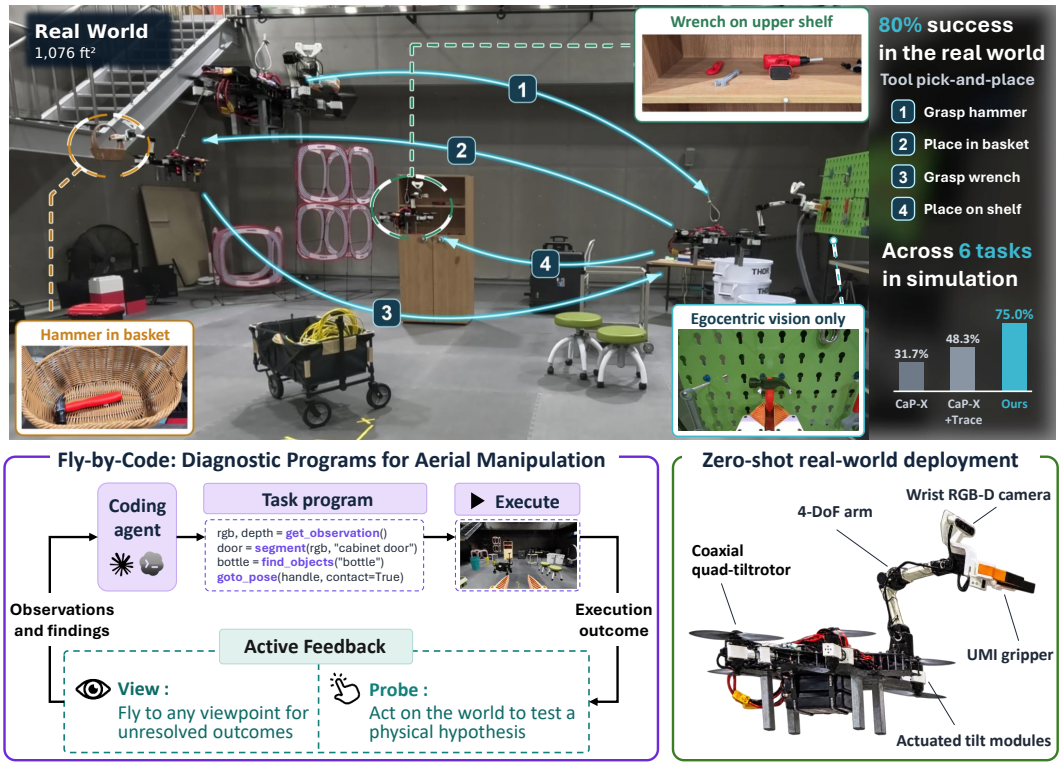


Figure 1: Fly-by-Code addresses visual and physical ambiguities under partial observability and limited interaction robustness via **Active Feedback** (AF). Rather than relying only on feedback passively collected during task execution, Fly-by-Code generates an executable diagnostic program to acquire the evidence needed for outcome assessment and program revision. The program either navigates to an informative viewpoint (**View**) to inspect an unobserved outcome or performs a targeted physical interaction (**Probe**) to distinguish between possible failure causes. Using shared robot APIs, task programs refined in simulation are transferred without source-code modification to a physical aerial manipulator, enabling multi-stage tasks such as the illustrated four-step tool-manipulation sequence.

ABSTRACT

Embodied coding agents emerge as a promising approach to construct robot policies from natural-language instructions. However, extending these agents to physically demanding settings introduces two fundamental challenges: partial observability and limited interaction robustness. These challenges can leave task-critical outcomes unobserved and failure causes unresolved, making execution feedback insufficient for completion assessment and program revision. To address these gaps, we introduce **Active Feedback** (AF), an active "View and Probe" framework. After each task execution, the agent synthesizes a sepa-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

rate diagnostic program to acquire informative viewpoints or perform targeted physical tests in simulation, returning evidence for completion assessment and program revision. We study AF in aerial manipulation, where payload constraints limit onboard sensing and exacerbate partial observability, while strong dynamic coupling and limited control margins make physical interactions fragile. Across six aerial manipulation tasks, AF increases final task success from 31.7% to 75% relative to terminal feedback, using the same coding-agent configuration and task-execution limit. The task programs synthesized in simulation transfer directly to a physical aerial manipulator via shared APIs without task-program source-code modifications and successfully execute multi-stage tasks in the real world. These results show that executable diagnosis helps embodied coding agents address gaps in execution feedback, improving both completion assessment and program refinement for autonomous aerial manipulation. **Project Page:** https://fbycode.github.io/flybycode/

1 INTRODUCTION

Embodied coding agents provide a promising approach to task-level coordination by translating natural-language instructions into executable programs that combine general control flow and computation with calls to robot perception and action APIs (Liang et al., 2023; Chen et al., 2024; Fu et al., 2026). In an iterative refinement loop, the agent executes a program, examines execution feedback, and revises the program until it determines that the task is complete. This refinement implicitly assumes that execution feedback provides sufficient evidence to assess the task outcome and, when execution fails, to identify how the program should be revised.

This assumption can break down under two challenging conditions: partial observability and limited interaction robustness. Partial observability can leave task-critical outcomes outside the robot's observations, while failures arising from physical interaction limitations can be difficult to distinguish from perception or manipulation errors. We refer to these two gaps in execution feedback as *missing outcome observations* and *unresolved failure causes*, respectively. Re-examining the same feedback cannot supply observations that were never collected or resolve failure causes that the available evidence does not distinguish. Effective refinement therefore requires the agent to identify what additional evidence is needed and how to acquire it.

Aerial manipulation provides a particularly demanding setting in which these limitations are amplified. An aerial robot equipped with a manipulator must physically interact with the environment while maintaining flight (Ollero et al., 2022). Spatially extended tasks can exceed the coverage of a fixed external camera, leaving the agent dependent on egocentric onboard sensing whose viewpoint changes with robot motion. At the same time, limited control margin and disturbances from manipulator motion and interaction wrenches can produce failures (Yadav et al., 2025; Hwang et al., 2025) that are difficult to distinguish from perception or manipulation errors. For example, an empty gripper after a placement attempt does not reveal whether the object reached the target, while a failed door-opening attempt may not reveal whether the grasp failed, the commanded motion was inappropriate, or the platform lacked sufficient physical capability.

Resolving these uncertainties may require acquiring new evidence: choosing a viewpoint that reveals an unseen outcome or performing a targeted physical interaction that tests a specific failure hypothesis. Motivated by this observation, we introduce **Active Feedback** (AF), an explicit *post-execution diagnostic stage*. Rather than immediately revising the task program from existing execution feedback, AF first lets the agent generate and execute a separate diagnostic program to acquire evidence for unresolved questions. Using the same robot APIs as the task program, the diagnostic program can acquire new viewpoints to reveal unseen outcomes (**View**) or perform targeted physical interactions in simulation to test hypotheses about interaction outcomes or embodiment constraints (**Probe**). The resulting observations and measurements are returned to the same agent, providing additional evidence for both completion assessment and subsequent program revision.

Existing coding-agent systems obtain evidence through multimodal execution feedback (Fu et al., 2026), per-primitive traces (Lu et al., 2026), task-specific verification procedures (Xiao et al., 2026), or information-gathering actions embedded within task execution (Fu et al., 2026). AF instead synthesizes an attempt-specific diagnostic procedure after execution, separately from task evaluation.

The same AF prompt is used across tasks, while the diagnostic program is generated to investigate the unresolved questions of each attempt. By decoupling diagnosis from task progress, AF gives the agent freedom to select actions for their information value rather than their immediate contribution to task completion. Task outcomes are evaluated before diagnosis, and the environment resets before the next task attempt, so diagnostic actions contribute evidence without earning task credit.

Across six aerial manipulation tasks in two simulated environments, AF achieves 75% final task success, outperforming terminal (31.7%) and execution-trace (48.3%) feedback under the same agent configurations and execution limits. These results demonstrate that acquiring proper observations and interaction evidence improves program synthesis beyond existing execution-feedback baselines. We analyze task progress across refinement iterations, evaluate AF across models and interfaces, and examine how diagnostics inform program revisions.

Finally, we investigate the real-world transferability of our approach by deploying the synthesized programs on a physical aerial manipulator. Shared task-level APIs allow deploying these programs on our custom hardware without source code modifications, despite differing underlying sensing and control implementations. The resulting programs repeatedly complete tool pick-and-place and cabinet bottle-delivery tasks despite differences between the simulated and physical scenes and variations in initial robot position and heading across trials (Figure 1).

In summary, our work makes the following key contributions:

- We characterize two limitations of execution feedback for embodied coding agents: *missing outcome observations* and *unresolved failure causes*, which hinder completion assessment and program revision.
- We introduce **Active Feedback** (AF), a post-execution diagnostic stage in which the agent synthesizes an attempt-specific executable program to actively acquire evidence through informative viewpoints (**View**) and targeted physical interactions in simulation (**Probe**).
- We evaluate AF in aerial manipulation. Across six simulated tasks, AF improves final task success to 75%, compared with 31.7% for terminal feedback and 48.3% for execution-trace feedback, while also improving completion assessment. We further demonstrate repeated physical execution of simulation-synthesized task programs through shared robot APIs without task-program source-code modifications. To the best of our knowledge, this is the first study of *embodied coding agents for aerial manipulation*.

2 Related Work

Robot program synthesis and feedback-driven refinement. Code-as-policy and program-synthesis approaches represent robot behavior as executable programs that compose perception and control APIs (Liang et al., 2023; Singh et al., 2023; Chen et al., 2024; Mu et al., 2024; Oelerich et al., 2025; Mower et al., 2026). Recent work extends program generation through iterative refinement: CaP-X uses textual and visual execution feedback, ASPIRE leverages multimodal execution traces with reusable repairs and program search, ARCHITECT incorporates execution traces and human corrections into reusable skills, and ENPIRE constructs task-specific reset, safety, and verification interfaces (Fu et al., 2026; Lu et al., 2026; Chen et al., 2026a; Xiao et al., 2026). Related methods explain failures from execution experience, generate recovery behaviors, or acquire missing physical information (Liu et al., 2023; Schlesinger et al., 2024; Lin et al., 2026). Active Feedback (AF) instead addresses cases where existing execution evidence is insufficient, synthesizing an attempt-specific diagnostic program to actively acquire visual or physical evidence for completion assessment and subsequent program refinement.

Active and interactive perception. Active perception selects sensing actions to acquire task-relevant information (Bajcsy, 1988; Bajcsy et al., 2018), while interactive perception uses physical interaction to reveal information that is difficult to obtain from passive observation alone (Bohg et al., 2017). Recent robotic manipulation systems extend these ideas through task-driven viewpoint selection and coordinated perception–manipulation under occlusion (Weng et al., 2024; Wang et al., 2025; Xiong et al., 2025; Liu et al., 2026b;a). Language-model-based agents can likewise select epistemic actions to acquire multimodal information under partial observability (Zhao et al., 2023b). Whereas prior methods primarily acquire information to support ongoing perception or task

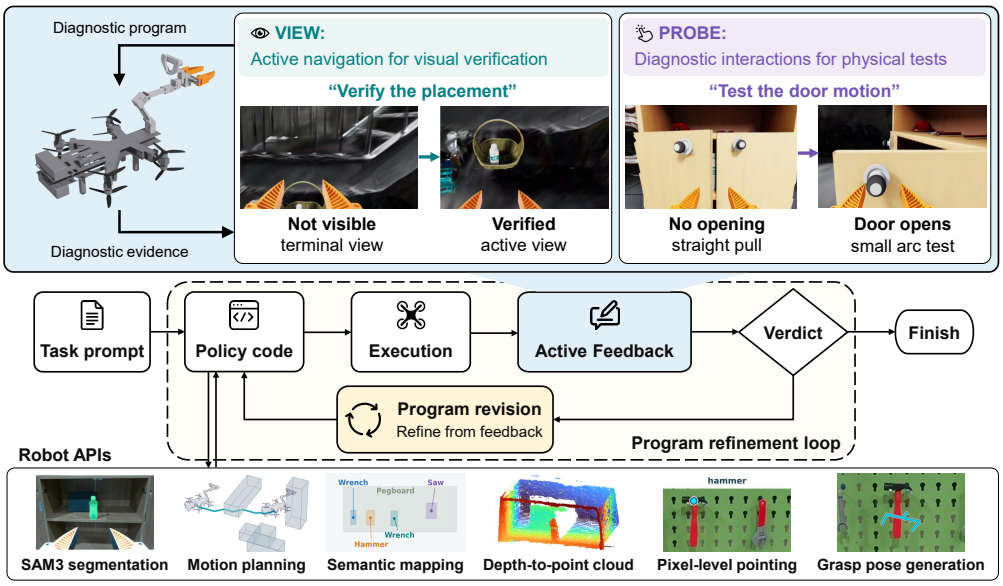


Figure 2: Overview of Fly-by-Code. **Bottom:** The program refinement loop. The coding agent translates a task prompt into executable policy code using shared robot APIs. Following execution, it enters the Active Feedback (AF) stage to acquire diagnostic evidence, iteratively refining the program until it decides to terminate the revision process. **Top:** During AF, the agent synthesizes a diagnostic program to resolve execution uncertainties. **View** actively navigates to informative viewpoints (e.g., inspecting a basket from above to verify placement). **Probe** conducts diagnostic physical tests (e.g., testing door articulation with an arc motion after a straight pull fails).

execution, AF synthesizes a separate diagnostic program after each attempt to resolve uncertainties for completion assessment and program refinement.

Learning- and language-guided aerial manipulation. Aerial manipulation brings together challenges in flight, perception, and physical interaction, with research spanning platform design, contact-aware control, whole-body motion planning, and perception-driven manipulation (Ollero et al., 2022; Bodie et al., 2021; Zhao et al., 2023a; Guo et al., 2024; Deng et al., 2025; Ubellacker et al., 2024; Bauer et al., 2025; Lee et al., 2026). Learning-based approaches learn manipulation policies from demonstrations or adapt pretrained policies to aerial platforms (He et al., 2025; Gupta et al., 2026; Tucker et al., 2026), while we focus on actively acquiring post-execution evidence to assess and revise policies expressed as executable programs. Among language-guided approaches, AERMANI-VLM constrains high-level decisions to predefined flight-safe skills to improve execution reliability (Mishra et al., 2025), while AeroGrab actively changes viewpoints to improve target observability under occlusion (Singh et al., 2026).

3 Active Feedback for Aerial Policy Synthesis

Active Feedback (AF) separates task-program execution from the acquisition of evidence needed for completion assessment and program revision. As illustrated in Figure 2, Fly-by-Code implements this separation by inserting an agent-generated diagnostic stage between task execution and the decision to finish or regenerate the program.

3.1 Motivation: Limitations of Passive Feedback

Existing embodied coding agents rely on passive execution feedback, such as console logs and terminal observations, to assess task completion and guide revisions. However, this reliance introduces two fundamental limitations in aerial manipulation. First, visual ambiguity masks true physical outcomes. Because an egocentric camera tightly couples its viewpoint to the robot's flight trajectory, critical end-states frequently fall outside the terminal field of view. Second, physical ambiguity obscures the root causes of failed interactions. A static terminal image may not distinguish whether an

interaction failed due to erroneous contact geometry or violated kinematic constraints. To reliably resolve these ambiguities, Fly-by-Code supplements passive observation logs with actively acquired evidence.

3.2 Synthesis of Diagnostic Programs

Fly-by-Code decouples task execution from outcome investigation. Following a task attempt, the agent reviews the standard execution feedback—including console logs, terminal observations, and semantic-map summaries—to identify specific visual or physical uncertainties left unresolved by this passive evidence. To resolve these gaps, the agent generates a dedicated diagnostic program. This program utilizes the same shared robot APIs (Appendix A.4, Table 3) as the task policy but serves a fundamentally different purpose: acquiring information rather than advancing the task. Because the environment records the task score prior to this stage, the agent safely performs diagnostic interactions without penalizing its evaluation. By synthesizing diagnostic code after task execution, the agent can target uncertainties specific to the current attempt rather than relying on pre-programmed verification steps (Appendix D.1).

3.3 Diagnostic Actions: View and Probe

The diagnostic program operationalizes its investigation through two targeted action primitives: View and Probe (Figure 2). Rather than following a fixed inspection sequence, the Fly-by-Code agent dynamically composes these primitives to gather the exact evidence required.

Active Viewpoint Selection (View). Visual ambiguities often arise from spatially distributed targets that fall outside the terminal camera view. To overcome these blind spots, the agent employs active viewpoint selection. Using the semantic map as a spatial reference, the diagnostic program navigates the aerial platform to additional viewpoints. For example, instead of relying on a static terminal frame, the agent can fly directly above a target to verify successful object placement.

Diagnostic Physical Interaction (Probe). Certain physical properties, such as contact responses, payload limits, or articulated constraints, remain invisible to purely visual inspection. The agent resolves these physical ambiguities by executing targeted physical interactions. For instance, the agent can push a misaligned box at varying contact points to measure the resulting translation and rotation. Because these interactions occur during the unscored diagnostic stage, the agent can explore physical responses without being penalized for exploratory failures. Furthermore, executing these probes directly from the post-execution state allows the agent to investigate long-horizon outcomes without replaying the entire task sequence.

3.4 Closing the Feedback Loop

The feedback loop concludes by integrating diagnostic findings into completion assessment and subsequent program revision. The diagnostic program returns selected observations, agent-written inspection notes, and console measurements to the agent, which evaluates them alongside the original execution logs to declare task completion (FINISH) or synthesize a revised policy (REGENERATE). Before the next task attempt, the environment resets its physical state, while the diagnostic information persists within the agent's context to inform the revised task program.

4 Experiments

We evaluate Active Feedback (AF) through three questions: (Q1) Does AF improve task success and completion assessment over passive execution feedback? (Q2) How does diagnostic evidence inform program refinement? (Q3) How does AF perform across models and agent interfaces? Finally, section 4.5 evaluates real world deployment of the simulation-synthesized programs.

4.1 Experimental Setup

Simulation Environments and Robot. We evaluate Fly-by-Code in six tasks extending AM-Bench (Wang et al., 2026), implemented in Isaac Lab (Mittal et al., 2025). The simulated agent

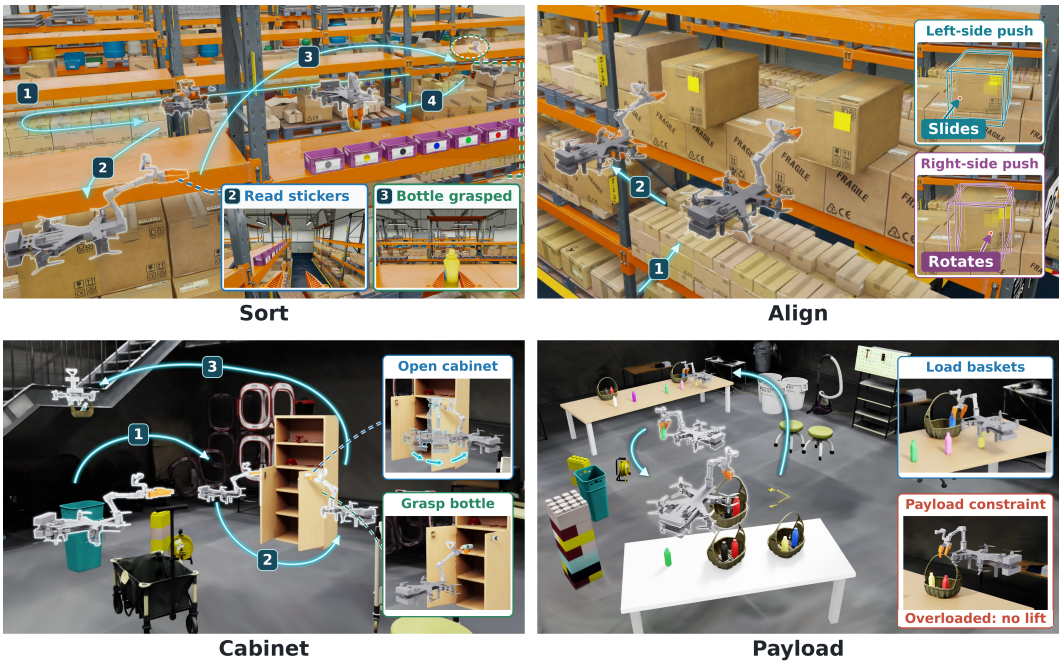


Figure 3: Representative aerial manipulation tasks. **Sort:** Identify the target bottle and place it into the color-matched bin. **Align:** Push protruding boxes into alignment with the rack. Due to offset centers of mass, varying contact points induce distinct translational or rotational motions. **Cabinet:** Open the cabinet doors, retrieve the bottle inside, and deliver it to a hanging basket. **Payload:** Transport filled bottles and baskets with minimal hall crossings under an undisclosed payload limit. The Sort and Align tasks operate in the warehouse simulation, whereas Cabinet and Payload use the laboratory digital twin. Overlaid robot poses and arrows illustrate sequential execution stages and motion trajectories.

controls an omnidirectional aerial manipulator (Appendix A.3). The tasks range from spatial object transport (1-Object and 2-Objects Pick-and-place, Sort) to contact-rich interactions involving articulated structures (Cabinet), unknown payload constraints (Payload), and contact-dependent pushing (Align). A shared execution backend manages collision-aware whole-body motion planning and tracks the generated trajectories under simulated flight dynamics (Appendix A.4).

Baselines and Agent Configuration. We build on CaP-X (Fu et al., 2026), adapting its program-generation and execution loop to aerial manipulation. All conditions share the same perception, control, and semantic-map APIs and a fixed skill library adapted from CaP-X. We use direct RGB feedback without CaP-Agent0's visual-differencing module or parallel ensemble. Unlike the continuous execution protocol in CaP-X, each revised program is executed from the initial environment state and must perform the full task. Within this common loop, we compare AF against two baselines: CaP-X, which provides execution logs and terminal RGB observations, and CaP-X+Trace, which additionally provides structured primitive-call records and execution keyframes. The agent receives these traces before deciding whether to finish or regenerate the task program. CaP-X+Trace adapts the trace-based feedback used in ASPIRE (Lu et al., 2026), without incorporating its skill library or evolutionary program search. The primary evaluation uses Claude Opus 5 (Anthropic, 2026a) through persistent sessions with effort set to `high`. Each trial permits one initial task-program execution and up to ten revisions.

4.2 Task Success and Completion Assessment

Overall Task Performance. Across six aerial manipulation tasks, AF achieves an overall success rate of 75%, significantly outperforming CaP-X (31.7%) and CaP-X+Trace (48.3%) (Figure 4). This advantage becomes increasingly pronounced in tasks requiring extended physical sequences.

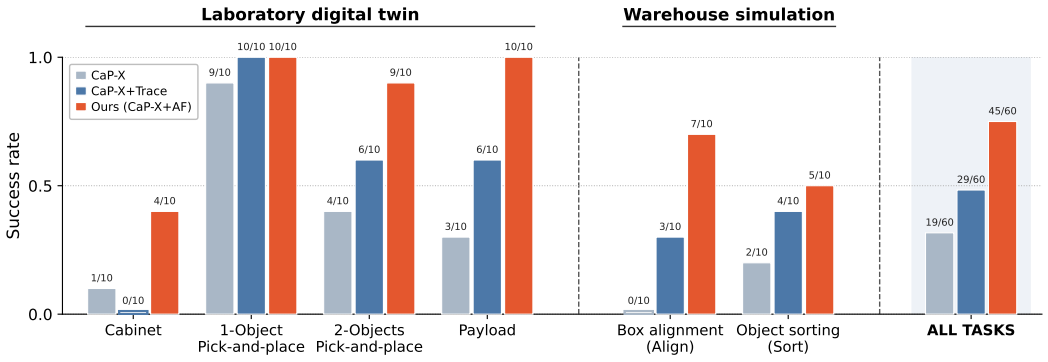


Figure 4: Final task success across six tasks. AF achieves the highest success rate overall and matches or exceeds both CaP-X and CaP-X+Trace on every task. Each trial permits one initial execution and up to ten policy revisions. Labels above the bars indicate successful trials out of ten total trials per task.

For example, in 2-Objects Pick-and-place, AF reaches 90% success compared to 40% and 60% for the baselines. Similarly, AF substantially improves performance on tasks governed by hidden physical constraints, such as Payload (100% vs. 30% and 60%) and Cabinet (40% vs. 10% and 0%). Furthermore, beyond final success rates, AF consistently accelerates iterative task progress, demonstrating that active diagnostic findings enable more effective policy refinement than passive traces (Figure 9 in Appendix C.1).

Reliable Completion Assessment. Crucially, AF drastically reduces false positive completion assessments. Across 60 trials, AF yields only five false positives, achieving a completion assessment precision of 90.0% (Table 1). In contrast, CaP-X and CaP-X+Trace produce 25 (43.2% precision) and 19 (60.4% precision) false positives, respectively. While baseline agents frequently declare success based on unverified execution logs—such as assuming an open gripper guarantees a successful placement—Fly-by-Code actively acquires visual and physical evidence to verify spatially distributed outcomes before making a final judgment.

Table 1: Completion assessment performance. Each triplet reports correct completion declarations (TP), incorrect completion declarations (FP), and trials reaching the step limit without a declaration (ND), with 10 trials per task and 60 overall. Precision is TP/(TP+FP). P&P denotes Pick-and-place.

	Task Outcomes (TP / FP / ND)						Overall	
Method	Cabinet	1-Obj P&P	2-Obj P&P	Payload	Align	Sort	TP/FP/ND	Precision
CaP-X	1/**1**/8	9/1/0	4/5/1	3/6/1	0/10/0	2/2/6	19/25/16	43.2%
CaP-X+Trace	0/3/7	**10/0**/0	6/3/1	6/3/1	3/7/0	4/3/3	29/19/12	60.4%
AF (Ours)	**4/1**/5	**10/0**/0	**9/1**/0	**10/0**/0	**7/3**/0	**5/0**/5	**45/5**/10	**90.0%**

4.3 How Does Active Feedback Support Policy Refinement?

To qualitatively analyze how Active Feedback (AF) informs the agent's program revisions, we demonstrate its diagnostic capabilities through three distinct execution examples. While baseline agents rely strictly on mid-execution physical feedback, Fly-by-Code explicitly executes diagnostic actions to investigate unresolved visual, physical, and task-level constraints.

First, **Visual Verification of Task Outcomes.** AF prevents speculative policy updates by acquiring targeted visual evidence. In the sequential 2-object Pick-and-place, baseline agents frequently drop a tool and mistakenly declare success simply because their gripper is empty. In contrast, Fly-by-Code actively navigates above the basket to visually verify the tool's actual placement (Figure 5(a), Appendix D.1). This active viewpoint selection prevents false positives and ensures the agent bases its subsequent actions on actual environmental states. Second, **Physical Probing for Control Refinement.** When visual inspection cannot resolve physical properties, Fly-by-Code employs targeted physical probing. In Align task, the agent explicitly pushes the target box to measure its resulting



Figure 5: Examples of Active Feedback (AF) for completion assessment and program refinement. **(a) View:** The robot moves above the basket to inspect the placement outcome. **(b) Probe:** A targeted push reveals the box's response to physical contact. **(c) Probe:** A diagnostic lift test of a 1.2 kg basket load is refused, indicating that the proposed load is infeasible (left). The next task program revises the object grouping and successfully carries a 1.1 kg load (right).

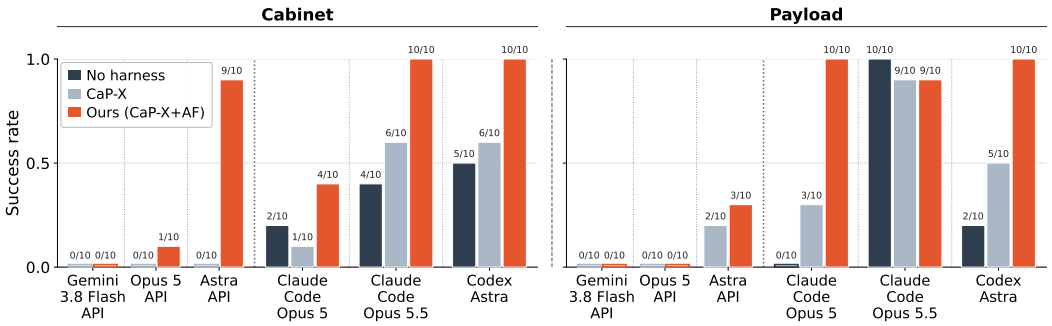


Figure 6: Final task success across models and agent interfaces on Cabinet and Payload. Labels above the bars indicate successful trials / total trials, with ten trials per task and condition.

rotational and translational responses (Figure 5 (b), Appendix D.2). Instead of passive trial-and-error, the agent uses these precise physical measurements to optimize contact points and forces for the next attempt. Furthermore, **Task Planning under Inferred Payload Constraints.** Fly-by-Code supports task-level planning by probing the robot's unmodeled physical limits. In the Payload task, the agent deliberately tests varying bottle combinations to determine liftable loads (Figure 5 (c), Appendix D.2). These diagnostic lift tests establish explicit payload bounds, informing transport strategies like optimal object groupings. Consequently, AF transforms uncertain embodiment constraints into actionable planning decisions.

4.4 Generality Across Models and Interfaces

Consistent Performance Gains. Active Feedback (AF) improves task success across diverse foundation models and agent interfaces (Figure 6) (Google DeepMind, 2026; Anthropic, 2026a;b; OpenAI, 2026). Across all evaluated task—backbone model pairs, AF matches or exceeds the success rate of the passive CaP-X baseline. On the Cabinet task, AF drastically raises success from 0% to 90% using the Astra API, and from 60% to 100% with both Claude Code Opus 5.5 and Codex Astra. Similarly, on the Payload task, AF elevates success from 30% to 100% with Claude Code Opus 5, and from 50% to 100% with Codex Astra. While inherent model limits constrain simpler setups (e.g., Gemini 3.8 Flash API remaining at 0% under both methods), these results show that AF's benefits extend across multiple model–interface configurations, although their magnitude varies.

Necessity of Structured Diagnostics. To isolate the impact of Fly-by-Code's structured diagnostic stage, we evaluate a "No harness" baseline that receives a superset of execution evidence—including full raw videos and console logs—and freely inspects it without predefined feedback packaging. Despite this unrestricted information access from all sensors, the No harness condition struggles on the Cabinet task, achieving 20% to 50% across coding-agent configurations, reaching at most half the success rate of AF. On the Payload task, No harness achieves 0% with Claude Code Opus 5 and 20% with Codex Astra, whereas AF achieves 100% with both. While unstructured data access

occasionally benefits specific model-task combinations (e.g., Payload with Claude Code Opus 5.5), the substantial overall performance gap demonstrates that synthesizing targeted diagnostic programs resolves execution uncertainties more effectively than unstructured log analysis.

4.5 Sim-to-Real Transfer and Physical Deployment

To demonstrate the practical robustness of Fly-by-Code, we deploy task programs synthesized with AF in simulation on a physical aerial manipulator. Physical aerial manipulation involves dynamic disturbances and visual conditions that differ from those in simulation. However, through a unified hardware-software interface, the physical robot executes the exact simulation-synthesized Python programs without source-code modification. Rather than replaying simulated trajectories, the deployed task programs use real-world observations through the shared robot APIs to guide execution.

Autonomous Long-Horizon Execution. We evaluate this zero-shot physical transfer on the 2-object Pick-and-place and the Cabinet tasks. The unmodified programs successfully coordinate sustained perception, flight, and contact-rich interactions in the real world. As detailed in Table 2, Fly-by-Code achieves a full-task success rate of 80% (4/5 trials) for 2-object Pick-and-place and 60% (3/5 trials) for cabinet task. Notably, these executions require approximately 8 to 9 minutes of sustained autonomous operation. Repeated task completion despite differences between simulated and physical scene configurations and variations in initial robot pose supports the transfer of task programs refined with AF in simulation (Appendix E.2, Tables 8 and 9).

Table 2: Physical task outcomes. Denominators include all evaluated trials per task, even those not reaching the subtask.

Task	Outcome	Successful trials
Tool pick-and-place	Grasp hammer	5/5
	Place hammer in basket	4/5
	Grasp wrench	5/5
	Place wrench on shelf	5/5
	Full task	**4/5**
Water-bottle delivery	Open left door	5/5
	Open right door	4/5
	Grasp bottle	4/5
	Place bottle in basket	3/5
	Full task	**3/5**

5 Conclusion

We introduce Active Feedback (AF), a post-execution diagnostic stage in which the agent synthesizes a separate program to acquire informative viewpoints and conduct targeted physical tests in simulation. The resulting evidence informs both completion judgments and subsequent program refinement. We study this approach in embodied coding agents for aerial manipulation, where long-horizon task execution requires reliable completion assessment and informative feedback for program revision. Across six simulated tasks, AF improves final task success and completion assessment compared with terminal and execution-trace feedback. Through shared robot APIs, task programs refined with AF in simulation achieve zero-shot sim-to-real transfer to a physical aerial manipulator without task-program source-code modifications, repeatedly completing long-horizon tasks despite differences in scene configuration and variations in initial robot pose. The resulting workflow links targeted feedback acquisition to task-program refinement and full-task physical deployment in aerial manipulation.

6 Limitations

While Fly-by-Code successfully resolves visual and physical ambiguities through Active Feedback (AF), this explicit diagnostic stage inherently increases model inference and robot interaction times per trial. Furthermore, although post-execution inspection actively acquires targeted evidence, its effectiveness remains bounded by onboard perception errors and unreachable viewpoints. Simply performing a diagnostic check does not guarantee a correct task-completion judgment if the underlying sensory data is flawed. In Fly-by-Code, diagnostic interactions are performed in simulation, and the refined task programs achieve zero-shot sim-to-real transfer without task-program source-code modifications. However, this protocol does not provide an operational safety guarantee for unrestricted physical probing directly in the real world. Additionally, while the framework effectively handles spatial discrepancies across known tasks, our successful deployment relies heavily on underlying motion planning and flight controllers. We do not claim zero-shot generalization to entirely unseen physical environments or radically different embodiment morphologies.

Ethics Statement

Physical experiments take place in a controlled flight cage under operator supervision. The operator controls takeoff, the task-start signal, landing, and emergency intervention. Diagnostic physical probing is confined to simulation; simulated resets do not justify unsafe actions on hardware. The system is a research prototype, not a validated autonomous platform for operation around uninvolved people. All released artifacts, including figures, videos, and the project page, were audited before submission, and identifying signage, faces, file paths, metadata, and links were removed or anonymized.

Reproducibility Statement

The appendix specifies the feedback loop, interface contract, simulator and hardware configurations, task predicates, model snapshots, selection rules, perturbation distributions, and resource accounting. Task outcomes are scored before the diagnostic stage, and progress is computed from task-program execution records alone, so diagnostic-stage actions are never credited (Appendix B.2). Appendix D traces diagnostic acquisition to the resulting verdicts and program revisions for representative trials.

AI Use Statement

Generative AI tools helped organize author-provided concepts, review evaluation and reporting choices, edit prose, summarize supplied literature, identify candidate references, and format bibliography entries. This editorial assistance did not generate experimental measurements or substitute for robot trials. The embodied coding agents evaluated in the study are components of the research method and are described in Sections 3 and 4. The authors are responsible for verifying the manuscript's citations, claims, analyses, and results.

References

Anthropic. Introducing Claude Opus 5, 2026a. URL https://www.anthropic.com/news/claude-opus-5. Accessed September 26, 2026.

Anthropic. Introducing Claude Opus 5.5, 2026b. URL https://www.anthropic.com/claude-opus-5-5. Accessed September 26, 2026.

Ruzena Bajcsy. Active perception. *Proceedings of the IEEE*, 76(8):996–1005, 1988.

Ruzena Bajcsy, Yiannis Aloimonos, and John K. Tsotsos. Revisiting active perception. *Autonomous Robots*, 42(2):177–196, 2018.

Erik Bauer, Marc Blöchlinger, Pascal Strauch, Arman Raayatsanati, Curdin Cavelti, and Robert K. Katzschmann. An open-source soft robotic platform for autonomous aerial manipulation in the wild. In *Conference on Robot Learning (CoRL)*, volume 270 of *Proceedings of Machine Learning Research*, pp. 3094–3106, 2025.

Karen Bodie, Maximilian Brunner, Michael Pantic, Stefan Walser, Patrick Pfändler, Ueli Angst, Roland Siegwart, and Juan Nieto. Active interaction force control for contact-based inspection with a fully actuated aerial vehicle. *IEEE Transactions on Robotics*, 37(3):709–722, 2021.

Jeannette Bohg, Karol Hausman, Bharath Sankaran, Oliver Brock, Danica Kragic, Stefan Schaal, and Gaurav S. Sukhatme. Interactive perception: Leveraging action in perception and perception in action. *IEEE Transactions on Robotics*, 33(6):1273–1291, 2017.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Yash Katariya, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, et al. JAX: Composable transformations of Python+NumPy programs. Software, 2018. URL https://github.com/jax-ml/jax.

Daphne Chen, Archit Ritesh Jain, Eric Goossen, Emma Romig, Michael Murray, Nick Walker, and Maya Cakmak. A few words go a long way: Language guided robot policy synthesis. arXiv preprint arXiv:2607.23784, 2026a.

Junting Chen, Yao Mu, Qiaojun Yu, Tianming Wei, Silang Wu, Zhecheng Yuan, Zhixuan Liang, Chao Yang, Kaipeng Zhang, Wenqi Shao, et al. RoboScript: Code generation for free-form manipulation tasks across real and simulation. arXiv preprint arXiv:2402.14623, 2024.

Xingyu Chen, Fu-Jen Chu, Pierre Gleize, Kevin J Liang, Alexander Sax, Hao Tang, Weiyao Wang, Michelle Guo, Thibaut Hardin, Xiang Li, et al. SAM 3D: 3Dfy anything in images. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7220–7232, 2026b.

Weiliang Deng, Hongming Chen, Biyu Ye, Haoran Chen, Ziliang Li, and Ximin Lyu. Whole-body integrated motion planning for aerial manipulators. *IEEE Transactions on Robotics*, 41:6661–6679, 2025.

Daniel DeTone, Tianwei Shen, Fan Zhang, Lingni Ma, Julian Straub, Richard Newcombe, and Jakob Engel. Boxer: Robust lifting of open-world 2D bounding boxes to 3D. In *European Conference on Computer Vision (ECCV)*, 2026. Accepted; proceedings forthcoming.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, et al. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. In *International Conference on Machine Learning (ICML)*, 2026.

Farhad Goodarzi, Daewon Lee, and Taeyoung Lee. Geometric nonlinear PID control of a quadrotor UAV on SE(3). In *European Control Conference (ECC)*, pp. 3845–3850, 2013.

Google DeepMind. Gemini 3.8 Flash model card, 2026. URL `https://deepmind.google/models/model-cards/gemini-3-8-flash/`. Accessed September 26, 2026.

Xiaofeng Guo, Guanqi He, Jiahe Xu, Mohammadreza Mousaei, Junyi Geng, Sebastian Scherer, and Guanya Shi. Flying calligrapher: Contact-aware motion and force planning and control for aerial manipulation. *IEEE Robotics and Automation Letters*, 9(12):11194–11201, 2024.

Harsh Gupta, Xiaofeng Guo, Huy Ha, Chuer Pan, Muqing Cao, Dongjae Lee, Sebastian Scherer, Shuran Song, and Guanya Shi. UMI-on-Air: Embodiment-aware guidance for embodiment-agnostic visuomotor policies. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2026.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Guanqi He, Xiaofeng Guo, Luyi Tang, Yuanhang Zhang, Mohammadreza Mousaei, Jiahe Xu, Junyi Geng, Sebastian Scherer, and Guanya Shi. Flying hand: End-effector-centric framework for versatile aerial manipulation teleoperation and policy learning. In *Robotics: Science and Systems (RSS)*, 2025.

Mu Hu, Wei Yin, Chi Zhang, Zhipeng Cai, Xiaoxiao Long, Hao Chen, Kaixuan Wang, Gang Yu, Chunhua Shen, and Shaojie Shen. Metric3D v2: A versatile monocular geometric foundation model for zero-shot metric depth and surface normal estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):10579–10596, 2024.

Sunwoo Hwang, Dongjae Lee, Changhyeon Kim, and Hyoun Jin Kim. Autonomous heavy object pushing using a coaxial tiltrotor. *IEEE Transactions on Automation Science and Engineering*, 22: 4243–4256, 2025.

Sergio Izquierdo, Mohamed Sayed, Michael Firman, Guillermo Garcia-Hernando, Daniyar Turmukhambetov, Javier Civera, Oisin Mac Aodha, Gabriel Brostow, and Jamie Watson. MVSAnywhere: Zero-shot multi-view stereo. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11493–11504, 2025.

Chung Min Kim, Brent Yi, Hongsuk Choi, Yi Ma, Ken Goldberg, and Angjoo Kanazawa. PyRoki: A modular toolkit for robot kinematic optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1312–1319, 2025.

Dongjae Lee, Byeongjun Kim, and Hyoun Jin Kim. Autonomous aerial manipulation at arbitrary pose in SE(3) with robust control and whole-body planning. *The International Journal of Robotics Research*, 45(7):1021–1045, 2026.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500, 2023.

Chen-Yu Lin, Jing-Wen Chen, Hsueh-En Chang, Hung-An Chen, Sheng-Hsun Chang, Chi-Pin Huang, Fu-En Yang, Min-Hung Chen, Yi-Ting Chen, Yu-Chiang Frank Wang, et al. PhysCaP: Grounding code-as-policy agent with physics-informed exploration. arXiv preprint arXiv:2608.21031, 2026.

Mengzhen Liu, Enshen Zhou, Cheng Chi, Yi Han, Shanyu Rong, Liming Chen, Pengwei Wang, Zhongyuan Wang, and Shanghang Zhang. SaPaVe: Towards active perception and manipulation in vision-language action models for robotics. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 37164–37174, 2026a.

Zeyi Liu, Arpit Bahety, and Shuran Song. REFLECT: Summarizing robot experiences for failure explanation and correction. In *Conference on Robot Learning (CoRL)*, volume 229 of *Proceedings of Machine Learning Research*, pp. 3468–3484, 2023.

Zhenyang Liu, Yongchong Gu, Yikai Wang, Xiangyang Xue, and Yanwei Fu. ActiveVLA: Injecting active perception into vision-language-action models for precise 3D robotic manipulation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8141–8151, 2026b.

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

Matthias Minderer, Alexey Gritsenko, and Neil Houlsby. Scaling open-vocabulary object detection. In *Advances in Neural Information Processing Systems*, volume 36, pp. 72983–73007, 2023.

Sarthak Mishra, Rishabh Dev Yadav, Avirup Das, Saksham Gupta, Wei Pan, and Spandan Roy. AERMANI-VLM: Structured prompting and reasoning for aerial manipulation with vision language models. arXiv preprint arXiv:2511.01472, 2025.

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

Christopher E. Mower, Yuhui Wan, Hongzhan Yu, Antoine Grosnit, Jonas Gonzalez-Billandon, Matthieu Zimmer, Puze Liu, Daniel Palenicek, Davide Tateo, Jan Peters, et al. A robot operating system framework for using large language models in embodied AI. *Nature Machine Intelligence*, 8(3):313–325, 2026.

Yao Mu, Junting Chen, Qing-Long Zhang, Shoufa Chen, Qiaojun Yu, Chongjian Ge, Runjian Chen, Zhixuan Liang, Mengkang Hu, Chaofan Tao, et al. RoboCodeX: Multimodal code generation for robotic behavior synthesis. In *International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 36434–36454, 2024.

NVIDIA. Isaac Sim 6.0.0 documentation: Isaac Sim assets, 2026a. URL `https://docs.isaacsim.omniverse.nvidia.com/6.0.0/assets/usd_assets_overview.html`. Accessed September 26, 2026.

NVIDIA. Isaac Sim 5.1.0 documentation: Environment assets, 2026b. URL `https://docs.isaacsim.omniverse.nvidia.com/5.1.0/assets/usd_assets_environments.html`. Accessed September 26, 2026.

Thies Oelerich, Christian Hartl-Nesic, and Andreas Kugi. Language-guided manipulator motion planning with bounded task space. In *Conference on Robot Learning (CoRL)*, volume 270 of *Proceedings of Machine Learning Research*, pp. 749–779, 2025.

Anibal Ollero, Marco Tognon, Alejandro Suarez, Dongjun Lee, and Antonio Franchi. Past, present, and future of aerial robotic manipulators. *IEEE Transactions on Robotics*, 38(1):626–645, 2022.

OpenAI. GPT-6 Astra: A new generation of intelligence, 2026. URL `https://openai.com/index/gpt-6-astra/`. Accessed September 26, 2026.

Lukas Radl, Felix Windisch, Andreas Kurz, Thomas Köhler, Michael Steiner, and Markus Steinberger. Confidence-based mesh extraction from 3D Gaussians. In *European Conference on Computer Vision (ECCV)*, volume 17039 of *Lecture Notes in Computer Science*, pp. 307–325, 2026.

Claire Schlesinger, Arjun Guha, and Joydeep Biswas. Creating and repairing robot programs in open-world domains. arXiv preprint arXiv:2410.18893, 2024.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating situated robot task plans using large language models. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530, 2023.

Shivansh Pratap Singh, Naveen Sudheer Nair, Samaksh Ujjawal, Sarthak Mishra, Soham Patil, Rishabh Dev Yadav, and Spandan Roy. AeroGrab: A unified framework for aerial grasping in cluttered environments. arXiv preprint arXiv:2603.15097, 2026.

Johnathan Tucker, Denis Liu, Aiden Swann, Allen Ren, Javier Yu, Jiankai Sun, Brandon Kim, Lachlain McGranahan, Quan Vuong, and Mac Schwager. π, but make it fly: Physics-guided transfer of VLA models to aerial manipulation. arXiv preprint arXiv:2603.25038, 2026.

Samuel Ubellacker, Aaron Ray, James M. Bern, Jared Strader, and Luca Carlone. High-speed aerial grasping using a soft drone with onboard perception. *npj Robotics*, 2, 2024. Article 5.

Guokang Wang, Hang Li, Shuyuan Zhang, Di Guo, Yanhong Liu, and Huaping Liu. Observe then act: Asynchronous active vision-action model for robotic manipulation. *IEEE Robotics and Automation Letters*, 10(4):3422–3429, 2025.

Yutong Wang, Dongjae Lee, Xiaofeng Guo, Yuanzhu Zhan, Yufei Jiang, Bavin Saravanan, Muqing Cao, Jia Xie, Chenyang Mao, Sebastian Scherer, et al. AM-Bench: A modular simulation suite and benchmark for aerial manipulation policy learning. In *Conference on Robot Learning (CoRL)*, 2026. Accepted; proceedings forthcoming.

Zehang Weng, Peng Zhou, Hang Yin, Alexander Kravberg, Anastasiia Varava, David Navarro-Alarcon, and Danica Kragic. Interactive perception for deformable object manipulation. *IEEE Robotics and Automation Letters*, 9(9):7763–7770, 2024.

Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, et al. ENPIRE: Agentic robot policy self-improvement in the real world. In *Conference on Robot Learning (CoRL)*, 2026. Accepted; proceedings forthcoming.

Haoyu Xiong, Xiaomeng Xu, Jimmy Wu, Yifan Hou, Jeannette Bohg, and Shuran Song. Vision in action: Learning active perception from human demonstrations. In *Conference on Robot Learning (CoRL)*, volume 305 of *Proceedings of Machine Learning Research*, pp. 5450–5463, 2025.

Rishabh Dev Yadav, Swati Dantu, Wei Pan, Sihao Sun, Spandan Roy, and Simone Baldi. Modular adaptive aerial manipulation under unknown dynamic coupling forces. *IEEE/ASME Transactions on Mechatronics*, 30(4):2688–2698, 2025.

Moju Zhao, Kei Okada, and Masayuki Inaba. Versatile articulated aerial robot DRAGON: Aerial manipulation and grasping by vectorable thrust control. *The International Journal of Robotics Research*, 42(4–5):214–248, 2023a.

Xufeng Zhao, Mengdi Li, Cornelius Weber, Muhammad Burhan Hafez, and Stefan Wermter. Chat with the environment: Interactive multimodal perception using large language models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3590–3596, 2023b.

A Implementation Details

A.1 Policy synthesis and active feedback

The coding agent initially receives a task instruction, an embodiment description, task-specific environment notes, robot API documentation, and an initial wrist-camera RGB image. Active Feedback (AF) diagnostic programs execute from the state left by the task program, without an intervening reset. Before each revised task program runs, the environment and semantic map are reset. Task outcomes are evaluated before AF, as described in Appendix B.2.

Evidence review. The AF prompt includes the executed task program, its console output, the terminal camera image, and the current semantic-map summary. It instructs the agent to assess the available evidence for each task requirement before generating diagnostic code:

> FIRST, REASON — in plain text, BEFORE any code: take the task PART BY PART and say, for each part, whether what you already have settles it. If you are not sure, it is not settled. [...] THEN ACTIVELY INSPECT what you could not confirm.

If the existing evidence confirms every requirement, the agent may instead return `NO_INSPECTION_NEEDED` with a brief explanation, skipping diagnostic execution.

Diagnostic interaction. The AF prompt explicitly directs the agent to acquire informative viewpoints and permits information-seeking physical interaction in simulation:

> Fly the robot to viewpoint(s) from which the outcome—and the likely cause of any failure—is clearly visible, and CAPTURE images there with get_observation(note="..."). [...] This code is NOT graded, and the environment is RESET to its initial state before the next attempt—nothing you do here persists. Because nothing persists, you are not limited to looking: when something you need to know is not visible from ANY viewpoint, you may physically ACT ON THE WORLD to reveal it, including changing whatever you believe is in the way. The reset undoes every such change, so the only thing that survives this step is what you LEARN. Do not use this step to perform the task; use it to find out what you cannot otherwise know.

Evidence reporting and reuse. The prompt instructs the agent to select useful views and explicitly record its findings:

> A bare get_observation() is not shown again. Give note= to the views the next step will actually need: the outcome of each part you judged not settled, and whatever explains a failure or changes the next program. [...] Your program's console output is carried verbatim into the next code-generation step.

Images selected through `get_observation(note=...)` are supplied with their notes and recorded end-effector positions at capture. Calls without `note` can support perception within the diagnostic program, but their images are not forwarded; measurements derived from them can be conveyed through console output. The selected images, notes, positions, diagnostic reasoning, and console output are supplied alongside the original task-execution feedback. In a single response, the agent either declares `FINISH` or returns `REGENERATE` with a complete revised task program.

A.2 Feedback conditions and generator configurations

The primary comparison uses the same Claude Code–Opus 5 configuration across terminal feedback, execution traces, AF. Terminal feedback contains the execution outputs and terminal observation; traces add records collected at primitive-call boundaries; AF adds observations and measurements acquired by a separately generated diagnostic program.

The cross-configuration study evaluates AF within each model–interface configuration before comparing configurations. Coding-agent and direct-API interfaces can differ in tool access, context management, and execution history, so cross-configuration differences are not interpreted as an isolated effect of the underlying language model.

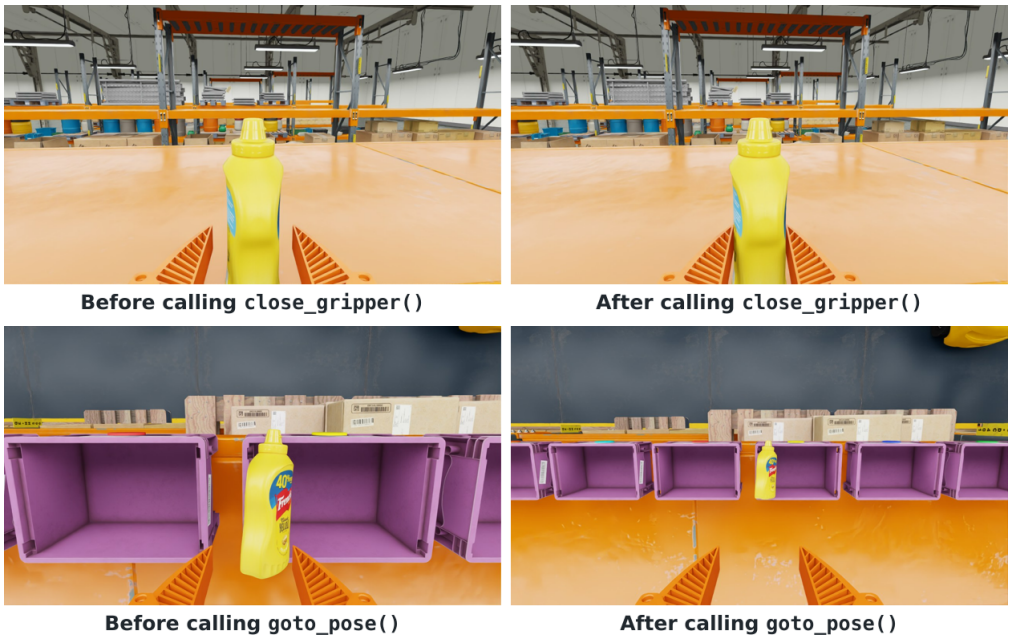


Figure 7: **Execution-trace feedback.** Images captured before and after a primitive call accompany its structured execution record. The observations follow the task execution rather than a dedicated verification trajectory.

Adaptation of CaP-X. Our implementation adapts CaP-X's robot APIs and fixed skill library to aerial manipulation, with the shared interfaces described in Appendix A.4. We hold the model and agent interface fixed across the primary feedback comparison and do not use parallel ensembling. CaP-Agent0 uses a visual-differencing module to summarize changes between successive observations in text. In spatially extended, long-horizon aerial tasks, however, egocentric observations may cover different regions and omit outcomes needed to establish completion, limiting what such comparisons can reveal. We therefore leave visual interpretation to the coding agent by providing RGB observations directly. The reset protocol in Appendix A.1 supports iterative refinement of a complete task program, with each revised program required to perform the entire task from the initial state.

Execution traces. Trace feedback pairs structured execution records with observations captured around primitive calls. The recorder follows ASPIRE's per-primitive tracing design (Lu et al., 2026), adapted to our robot APIs and CaP-X-based loop. Before each post-execution `FINISH`/`REGENERATE` decision, the same coding agent is directed to the trace files and instructed to read the structured records and inspect relevant keyframes using file tools. Session instructions explain the record format and provide interpretation heuristics for execution signals. The recorded viewpoints follow the task program rather than a separately generated diagnostic objective. Figure 7 illustrates this interface.

A.3 Simulation environments and robot configuration

We build on AM-Bench (Wang et al., 2026), a modular aerial-manipulation simulator that couples floating-base and manipulator dynamics with configurable low-level controllers. We integrate two task environments and a model of our custom-designed omnidirectional aerial manipulator, using the same robot configuration and control architecture as our physical platform. This setup allows synthesized programs to be evaluated under simulated flight dynamics and closed-loop tracking, rather than assuming perfect execution of commanded motions. Figure 8 presents the robot configuration and compares the simulated and physical laboratories.

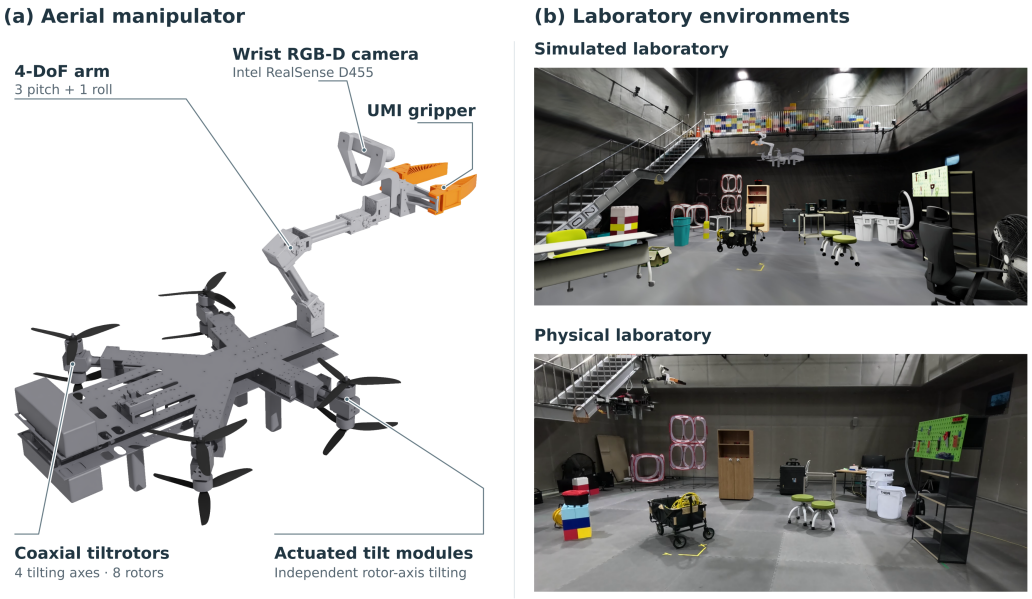


Figure 8: **Aerial manipulator and laboratory environment.** (a) Simulated model of our custom-designed platform, with its principal components annotated. (b) The laboratory digital twin and the physical laboratory, showing corresponding structures and task objects despite differences in their arrangement.

Laboratory digital twin. We construct the laboratory digital twin from photographs and videos captured with an iPhone. Individual object assets are generated using SAM 3D Objects (Chen et al., 2026b) and scaled to match directly measured dimensions of the corresponding physical objects. For the laboratory background, we build on CoMe (Radl et al., 2026) and modify the 3D Gaussian rendering pipeline for integration with Isaac Sim. During reconstruction, surface normals predicted by Metric3D v2 (Hu et al., 2024) and depth maps predicted by MVSAnywhere (Izquierdo et al., 2025) provide geometric supervision. We assemble the reconstructed background and object assets into a task environment that preserves the principal laboratory structures and task objects, without requiring an exact match to the physical obstacle arrangement (Figure 8b).

Warehouse. We build the warehouse environment using NVIDIA Isaac Sim's *Full Warehouse* asset (NVIDIA, 2026b). Task objects are drawn from the assets provided with Isaac Sim, including a YCB mustard bottle, KLT totes, and cardboard boxes from the warehouse scene (NVIDIA, 2026a). We configure their placement and task-specific visual markings to construct the evaluation scenarios.

Aerial manipulator. The simulated robot follows the configuration of our physical platform: a coaxial quad-tiltrotor with eight rotors, four actuated tilt joints, a four-degree-of-freedom arm, and a parallel-jaw UMI gripper (Figure 8a). The arm comprises three pitch joints and one wrist-roll joint. A wrist-mounted RGB-D camera provides egocentric observations whose viewpoint changes with flight and arm motion. The robot operates under simulated flight dynamics through the shared execution backend described in the following subsection.

A.4 Shared robot APIs and execution backend

Task programs and AF diagnostic programs use the same robot API surface. Table 3 summarizes the observation, perception, motion, and manipulation interfaces available in the final aerial-manipulation experiments. Table 4 lists the accompanying geometric and numerical utilities. These tables describe the available functions; individual programs may use only a subset.

Table 3: **Robot APIs available to generated programs.**

API	Function
Observation and motion	
`get_observation`	Retrieve wrist RGB-D images, camera parameters, and robot state.
`get_ee_pose`	Read the end-effector pose and gripper opening.
`goto_pose`	Move the grasp center to a target position with a specified approach direction and roll.
`open_gripper`, `close_gripper`	Open or close the gripper.
Image-based perception	
`point_prompt_molmo`	Locate a language-specified target in image coordinates.
`segment_sam3_text_prompt`	Segment targets specified by a text prompt.
`segment_sam3_point_prompt`	Segment a target specified by an image point.
Semantic-map queries	
`find_objects`	Retrieve mapped objects matching a label.
`nearest_object`	Find the mapped object nearest a world-frame point.
`object_at_point`	Query the mapped object occupying a world-frame point.
`describe_semantic_map`	Summarize the currently mapped objects as text.
Grasp proposal and selection	
`plan_grasp`	Propose scored grasps from depth, intrinsics, and segmentation.
`plan_grasp_from_point_clouds`	Propose scored grasps from scene and target point clouds.
`select_horizontal_grasp`	Select a candidate with a near-horizontal approach.

Table 4: **Geometric and numerical utilities.**

Utility	Function
Depth and point-cloud processing	
`depth_to_point_cloud`	Deproject depth into camera-frame 3D points.
`mask_to_world_points`	Deproject masked depth into world-frame points.
`pixel_to_world_point`	Convert a pixel and depth into a world-frame point.
`get_oriented_bounding_box_from_3d_points`	Estimate an oriented bounding box from 3D points.
`filter_noise`	Remove point-cloud outliers.
`subsample_point_cloud`	Reduce the number of points in a point cloud.
Coordinate transformations and geometry	
`rotation_matrix_to_quaternion`	Convert a rotation matrix into a quaternion.
`decompose_transform`	Extract position and quaternion from a transform.
`transform_points`	Apply a homogeneous transform to 3D points.
`interpolate_segment`	Generate points along a straight line segment.
`normalize_vector`	Normalize a nonzero vector to unit length.

Composing perception and action. The API exposes separate operations rather than a single object-to-action routine. For example, a program can query the semantic map for candidate locations, move to an informative viewpoint, segment the target, recover its geometry from depth, and command an approach followed by gripper closure. Grasp-proposal functions return candidates rather than executing grasps.

Semantic-map APIs. We maintain an object-level 3D semantic map to provide spatial references for objects outside the current camera view. The perception pipeline uses OWLv2 (Minderer et al., 2023) for open-vocabulary 2D detection and Boxer (DeTone et al., 2026) for 3D box estimation and multi-view integration using RGB-D observations and camera poses. The detection vocabulary is fixed by the task configuration in the final experiments. Our mapping layer adds sensor-range weighting and distinguishes depth-supported estimates from RGB-only hypotheses, which are treated as directional cues. The map exposes observation-derived labels and geometry estimates through the queries in Table 3; it contains no ground-truth object poses. Because labels can be incorrect and locations can become erroneous, the agent is instructed to use map entries as navigation and reinspection cues and verify them through direct perception. The map is cleared together with the environment before the next task-program execution.

Collision-aware motion planning. `goto_pose` executes a motion only after a collision-free whole-body trajectory has been found and verified. The workspace is represented by a voxel occupancy map converted to a signed distance field, and the robot by 30 collision spheres on the airframe, arm, and gripper. Goal configurations for the airframe and the four arm joints are solved by nonlinear least-squares inverse kinematics in PyRoki (Kim et al., 2025), and a route between them is found by A* search (Hart et al., 1968) over airframe position, heading, and arm posture. The resulting path is smoothed with a JAX-based optimizer (Bradbury et al., 2018) and time-parameterized as a quintic Bézier trajectory under velocity and acceleration limits, then tracked by a reference governor that slows the reference when tracking error grows and aborts if it leaves the verified safety margin.

Shared interface, environment-specific execution. The same task-level function names and calling conventions are used in simulation and physical deployment. Observation calls obtain rendered sensor data and simulated state in simulation, and measured sensor data and estimated state on hardware. For `goto_pose`, the planner and reference governor above are shared, while the control stack that tracks the reference is specific to each environment. The call reports whether the target was reached, the motion was refused with a stated reason, or execution timed out. Optional arguments specify a final approach offset, settling behavior, and intended contact. This separation preserves the generated program's perception and action interface while allowing the underlying sensing, calibration, and control implementations to differ.

Use during active feedback. AF does not introduce separate `view` or `probe` primitives. Instead, a diagnostic program composes the same observation and action APIs to acquire additional views or conduct physical tests in simulation. Execution-trace conditions retain the same callable API surface and add recording around API calls. Thus, feedback conditions differ in how evidence is acquired and exposed to the coding agent, rather than in the robot functions available to the generated program.

B Tasks and Evaluation Protocol

B.1 Task instructions and setup

The primary evaluation contains six aerial-manipulation tasks. 1-Object Pick-and-place, 2-Objects Pick-and-place, Cabinet, and Payload use the laboratory digital twin; Sorting and Align use the warehouse simulation. Each box presents a task-instruction excerpt, with omissions marked by [...]. The accompanying setup descriptions provide experimental context and are not part of the quoted instructions.

Task 1: 1-Object Pick-and-place

Pick the hammer off the pegboard and put it in the basket hanging on the stairs.

Setup. The hammer is mounted among other tools on a pegboard. The destination basket is across the hall, outside the source view.

Task 2: 2-Objects Pick-and-place

Put the hammer in the basket on the stairs, and put the wrench on the upper shelf in the open section of the cabinet across the hall.

Setup. The two tools share the pegboard source but have spatially separated destinations.

Task 3: Cabinet

Take the water bottle from the cabinet across the hall and put it in the basket hanging on the stairs.

Setup. The bottle initially sits behind two closed, outward-opening cabinet doors, which must be opened sufficiently to allow access and extraction.

Task 4: Payload

Eight water bottles stand on the long brown table; they differ only in colour. Three of them are empty; the other five hold water, and how much they hold may differ. Move the five bottles that hold water and both wicker baskets onto the long white table across the hall, and leave the empty bottles on the brown table. The mission is not only to get them there but to do it with the fewest crossings of the hall: a crossing is any flight between the two tables' sides of the hall, in either direction, and every crossing counts. [. . .]

Setup. Bottle masses and the carrying limit are not disclosed in the initial instruction. The environment prevents simulated grasp slippage from causing transport failures within the payload limit. Motion commands are rejected when the carried load exceeds the payload limit. This setup more faithfully reflects the carrying behavior observed on our physical platform and is applied identically across all comparison conditions.

Task 5: Object sorting

Pick the mustard bottle off the rack shelf and put it in the matching tote. PURPLE KLT totes (plastic bins) stand in a row on a rack deck; each tote wears ONE round coloured sticker. The bottle MUST go into the tote whose sticker has the SAME colour as the object you pick up [. . .].

Setup. The bottle starts on a high rack shelf, while nine purple totes stand on another rack. Their colored stickers distinguish the target tote from the distractors.

Task 6: Box alignment

Some of the cartons in this bay carry a YELLOW STICKER, and any of them that sticks out past the shelf front into the aisle is in danger of falling. Push every such carton in so that its front face sits flush with, and parallel to, the front faces of the neighbouring cartons on the same shelf – as flush and as parallel as you can make it. [. . .]

Setup. Two target cartons protrude from the rack. Their centers of mass are offset from their geometric centers, a property neither visible from their appearance nor disclosed in the instruction. A push at the geometric center can therefore induce rotation.

B.2 TASK COMPLETION AND PROGRESS

Final task completion. A trial is counted as successful only if the final task-program execution satisfies the task-specific completion criteria, evaluated before Active Feedback (AF), and the agent declares `FINISH`. A false `FINISH` remains a failure.

For feedback condition a, task τ, and trial i, let $y_{a,\tau,i}$ denote final task success. We run ten trials for each of the six primary tasks under each condition and give each task equal weight:

$$\bar{C}_a = \frac{1}{|\mathcal{T}|} \sum_{\tau \in \mathcal{T}} \frac{1}{n_\tau} \sum_{i=1}^{n_\tau} y_{a,\tau,i}, \qquad n_\tau = 10. \tag{1}$$

Because trial counts are equal across tasks, this task-macro-average equals the pooled success rate.

Progress scores. We normalize each task's progress score by its rubric maximum: 11 for Cabinet, 5 for 1-Object Pick-and-place, 9 for 2-Objects Pick-and-place, 8 for Payload, 6 for Object sorting, and 2 for Box alignment. Progress is computed from task-policy execution records; physical progress obtained during AF is not credited. A correct `FINISH` declaration earns the declaration-related point only when the task's completion conditions are satisfied. An incorrect declaration earns no declaration-related point. Trials that terminate early retain their final progress score through turn 11.

Cabinet (maximum: 11). Each stage contributes one point. Reaching a later stage credits the preceding stages. The door-opening threshold is $55°$, determined by testing cabinet access in $1°$ increments; this was the minimum opening angle that allowed the drone's manipulator to enter the cabinet in our setup.

Points	Criterion
1	A cabinet door handle is observed.
1	A door handle is grasped.
1	The grasped door is moved.
1	One door is opened by at least $55°$.
1	The other door handle is grasped with one door open.
1	The other door is moved with one door open.
1	Both doors are opened by at least $55°$.
1	The bottle is grasped.
1	The bottle is extracted from the cabinet.
1	The bottle is placed inside the destination basket.
1	The agent declares `FINISH` with the bottle successfully placed inside the basket.

1-Object Pick-and-place (maximum: 5). The following items each contribute one point. Placement records successful settling at the destination (hanging basket), whereas retention checks whether the object remains there at the end of the attempt.

Points	Item	Criterion
1	Grasp	The hammer is held and lifted at least 2 cm from its mount.
1	Transport	The held hammer is brought within 0.5 m of the destination center.
1	Placement	The hammer is placed at the destination and remains there for at least 0.5 s.
1	Retention	The hammer remains at the destination at the end of the attempt.
1	`FINISH`	The agent declares completion while the hammer remains successfully placed.

2-Objects Pick-and-place (maximum: 9). The hammer and wrench are scored separately: the hammer's destination is the hanging basket, and the wrench's destination is the designated shelf. The item scores are summed, with one additional point for a correct completion declaration.

Points	Item	Criterion
1	Hammer grasp	The hammer is held and lifted at least 2 cm from its mount.
1	Hammer transport	The held hammer is brought within 0.5 m of its destination center.
1	Hammer placement	The hammer is placed at its destination and remains there for at least 0.5 s.
1	Hammer retention	The hammer remains at its destination at the end of the attempt.
1	Wrench grasp	The wrench is held and lifted at least 2 cm from its mount.
1	Wrench transport	The held wrench is brought within 0.5 m of its destination center.
1	Wrench placement	The wrench is placed at its destination and remains there for at least 0.5 s.
1	Wrench retention	The wrench remains at its destination at the end of the attempt.
1	`FINISH`	The agent declares completion while both objects remain successfully placed.

Payload (maximum: 8). The task contains five filled bottles and three empty bottles. The rubric recognizes two transport pairs: red–black and yellow–blue. The green bottle is transported separately. Each item below contributes one point.

Points	Item	Criterion
1	Loading	A filled bottle is loaded into a basket.
1	Valid pair	A basket contains exactly one valid pair: red–black or yellow–blue.
1	Both pairs	Both valid pairs are loaded into their respective baskets.
1	First pair delivered	One valid pair is delivered in its basket to the destination table.
1	Both pairs delivered	Both valid pairs are delivered in their baskets to the destination table.
1	Green bottle delivered	The green bottle is carried separately to the destination table.
1	Empty bottles preserved	All three empty bottles remain at the source table.
1	`FINISH`	The agent declares completion with all five filled bottles and both baskets at the destination, all three empty bottles at the source, and no more than six crossings.

The empty-bottle criterion is already satisfied initially and is lost if the required source configuration is not preserved.

Object sorting (maximum: 6). The target tote is the one whose sticker matches the bottle's color. Each item below contributes one point.

Points	Item	Criterion
1	Grasp	The bottle is grasped.
1	Extraction	The bottle is moved out of the source shelf.
1	Transport	The held bottle is positioned above the target tote.
1	Release	Above the target tote, the gripper opening reaches at least 75 mm while the bottle is within 0.25 m vertically of the tote rim.
1	Retention	The task evaluator confirms successful placement in the target tote.
1	`FINISH`	The agent declares completion while the bottle is successfully placed in the target tote.

Box alignment (maximum: 2). The task requires aligning two protruding cartons marked with yellow stickers with the neighboring carton fronts. Let $e_{\max}$ be the largest absolute perpendicular distance from any evaluated front corner of either target carton to the reference plane. The precision component is

$$p = \mathrm{clip}\left(1 - \frac{e_{\max}}{15\,\mathrm{cm}},\, 0,\, 1\right). \tag{2}$$

The 15 cm normalization range follows the supplied scoring protocol and was selected using the CaP-X final-error median of approximately 15.1 cm. The two components below are added, and their sum is divided by 2 for the progress curves.

Points	Item	Criterion
0–1	Precision	The continuous precision score p defined above.
1	`FINISH`	The agent declares completion with $e_{\max} \leq 5\,\mathrm{cm}$.

For the reported Box alignment results, a true positive requires a `FINISH` declaration with $e_{\max} \leq 5\,\mathrm{cm}$.

For normalized score $r_{a,\tau,i,t}$ and final execution index $T_{a,\tau,i}$, the standing-score curve is

$$S_{a,\tau}(t) = \frac{1}{n_\tau} \sum_{i=1}^{n_\tau} r_{a,\tau,i,\min(t,T_{a,\tau,i})}. \tag{3}$$

Stopped trials retain their final score. The standing score can decrease after a revision, whereas a running maximum retains the best score reached by any preceding execution, regardless of whether the agent kept that program. These quantities answer different questions and are not used interchangeably.

B.3 Completion judgments and statistical analysis

Completion-judgment precision is the number of successful terminal `FINISH` declarations divided by the total number of terminal `FINISH` declarations. Trials ending in true `FINISH`, false `FINISH`, and exhaustion of the execution limit without `FINISH` are reported separately. These trial-level outcomes are distinct from error rates over all intermediate completion decisions.

Progress curves use ten trials per task and condition. Shaded bands are pointwise 95% percentile-bootstrap intervals obtained from 1,000 trial-level resamples: within each task and condition for task-specific curves, and over the pooled 60 trials within each condition for ALL TASKS. Repeated policy executions within a trial are not treated as independent samples, and matching trial indices across conditions are not assumed to define paired observations. Reported comparisons characterize the evaluated task suite.

B.4 Information access

The agent receives observations and execution outputs through the robot APIs and the feedback interface specified for each condition. Evaluator-only ground-truth states, task scores, and success labels are excluded from this feedback; grader logs are recorded separately from agent-visible execution output. The agent therefore makes completion decisions from the available execution evidence rather than receiving the evaluator's verdict.

C Additional Experimental Results

C.1 Task progress in the main evaluation

Figure 9 shows task progress across successive policy executions for the three feedback methods on all six tasks.

C.2 Task progress across models and interfaces

Figure 10 shows task progress across models and agent interfaces, with rows corresponding to tasks and columns to feedback methods.

C.3 Semantic map ablation

Role of semantic map support. We ablate semantic map support within Active Feedback (AF) on *2-Objects Pick-and-place* and *Object sorting*, which involve spatially separated task areas. We remove semantic map APIs, the semantic map summary included in the AF prompt, and semantic map-use instructions, while retaining the remaining perception and control APIs and the AF diagnostic stage. Task success drops from 9/10 to 4/10 on 2-Objects Pick-and-place and from 5/10 to

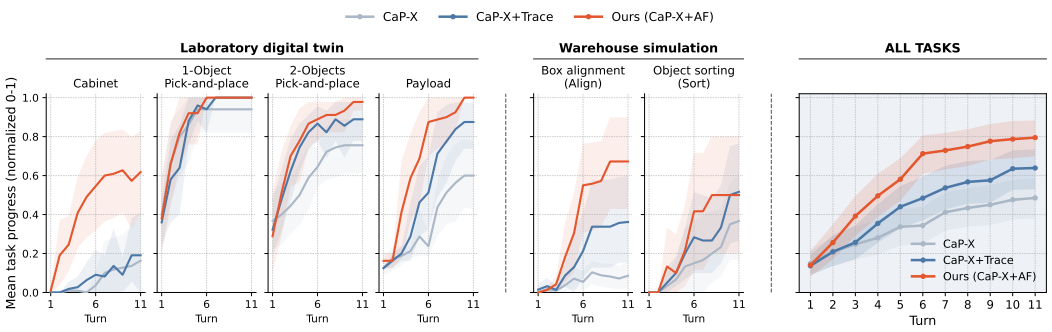


Figure 9: Task progress across policy executions. AF (Ours) achieves generally comparable or higher progress across individual tasks and shows a clear advantage in average progress across all six tasks. Turn 1 denotes the initial program execution, and trials that terminate early retain their final score through turn 11. Scores are normalized by the maximum score of each task-specific rubric (Appendix B.2).

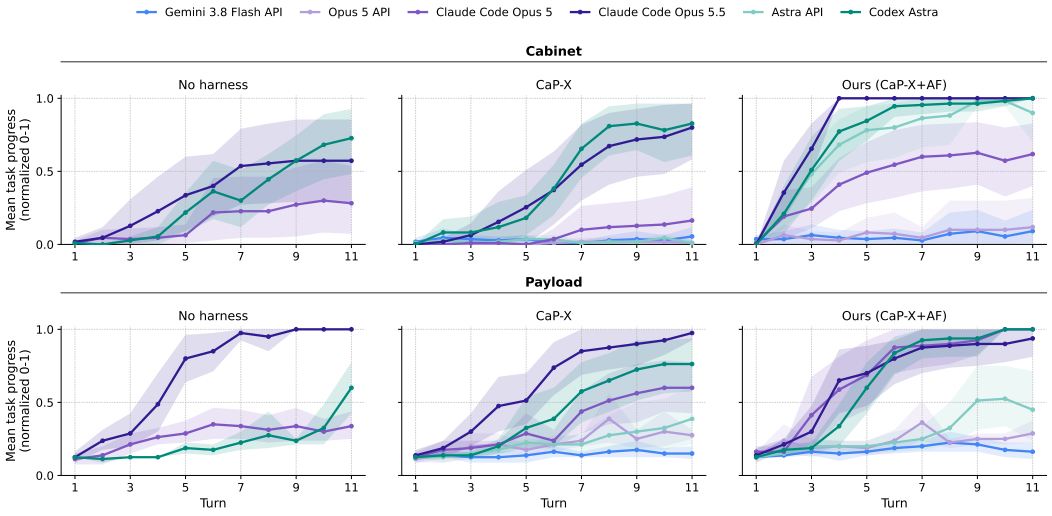


Figure 10: Task progress across API and coding-agent configurations. Each panel compares configurations under the same feedback method. Astra API achieves higher final average progress with AF than with CaP-X on both tasks, with a larger improvement on Cabinet. Normalization and early-termination handling follow Figure 9.

2/10 on Object sorting, while false `FINISH` declarations increase from 1 to 5 and from 0 to 3, respectively (Table 5).

Execution records suggest that semantic map supports spatial search during task execution. Aerial mobility enables observations from diverse viewpoints, but selecting informative viewpoints remains a separate decision. Program-invoked perception tools process selected views and queries, whereas the background mapper accumulates detections during flight, providing cues about objects outside the current view. On the first execution of 2-Objects Pick-and-place, both conditions reached a hammer grasp in 5/10 trials. Without the map, however, distant basket localization became less accurate and mean SAM3 calls per policy execution increased from 14 to 54, suggesting greater effort to reconstruct spatial information.

During AF, accumulated detections can also identify candidates for further inspection. All three false `FINISH` declarations in Object sorting without map support followed a final AF step that skipped observation and interpreted the target objects as absent. In one trial with map support, repeated SAM3 queries failed to detect the targets, but the map supplied target-tote candidates and the agent continued to `REGENERATE`. This suggests that map information can encourage continued

Table 5: Semantic map ablations within AF, with ten trials per task and condition. Success counts successful trials; FP counts false `FINISH` declarations. Cost is the mean estimated model cost per trial in USD.

AF variant	Success	FP	Cost ($)
2-Objects Pick-and-place			
Full AF	9/10	1	10.2
w/o semantic map	4/10	5	15.3
Sorting			
Full AF	5/10	0	29.3
w/o semantic map	2/10	3	37.6

search when individual perception queries provide no detections. Mean estimated model cost per trial increased from $10.2 to $15.3 on 2-Objects Pick-and-place and from $29.3 to $37.6 on Object sorting without map support. These costs reflect both execution count and termination decisions and should therefore be interpreted alongside success and false `FINISH` counts.

The map provides inspection targets rather than definitive task completion evidence. The agent is warned about inaccurate labels and locations and instructed to verify them through close-range observations. Across 11 analyzed executions involving wrench loss, the map never directly reported the fallen wrench's new location; in the eight executions where the loss was recognized, AF images and close-range perception results provided the evidence. Incorrect labels and stale locations sometimes confused the assessment. Together, these observations illustrate complementary roles: the map guides what to recheck and where, while AF checks the current state. Because map support is removed throughout the loop, this ablation does not isolate its contributions to task execution and diagnosis.

C.4 Resource use

Table 6 reports resource usage for the main evaluation (ten trials per task and method). Model usage includes initial generation, subsequent decision calls, and AF diagnostic calls where applicable. Per-success values equal total resource usage across all trials divided by the number of successes.

Table 6: Resource usage in the main evaluation. Turns denote policy executions; output tokens include thinking tokens and are reported in thousands (k). USD denotes estimated token-based LLM cost, not actual subscription charges. Per-success values include resources spent on failed trials; – denotes no observed success. Bold indicates the highest success count and lowest resource per success within each task block.

			Per trial			Per success		
Task	Method	Success	Turns	Output (k)	USD	Turns	Output (k)	USD
All tasks	CaP-X	19/60	7.2	142	9.6	22.7	447	30.4
	CaP-X+Trace	29/60	7.0	129	9.4	14.6	267	**19.5**
	AF (Ours)	**45/60**	6.0	155	17.1	**8.1**	**206**	22.8
Cabinet	CaP-X	1/10	10.9	156	10.8	109.0	1562	107.7
	CaP-X+Trace	0/10	10.9	174	13.7	–	–	–
	AF (Ours)	**4/10**	8.5	189	24.6	**21.2**	**473**	**61.5**
1-Object Pick-and-place	CaP-X	9/10	2.4	27	1.8	**2.7**	**30**	**2.0**
	CaP-X+Trace	**10/10**	3.6	53	3.4	3.6	53	3.4
	AF (Ours)	**10/10**	2.9	68	5.4	2.9	68	5.4
2-Objects Pick-and-place	CaP-X	4/10	5.5	104	6.1	13.8	259	15.4
	CaP-X+Trace	6/10	4.8	105	7.1	8.0	175	11.8
	AF (Ours)	**9/10**	4.7	111	10.2	**5.2**	**123**	**11.3**
Payload	CaP-X	3/10	8.5	250	20.2	28.3	834	67.4
	CaP-X+Trace	6/10	7.2	112	10.7	12.0	186	17.9
	AF (Ours)	**10/10**	5.5	136	16.5	**5.5**	**136**	**16.5**
Align (Box alignment)	CaP-X	0/10	5.6	121	6.4	–	–	–
	CaP-X+Trace	3/10	6.6	160	8.8	22.0	534	29.4
	AF (Ours)	**7/10**	6.3	203	16.7	**9.0**	**290**	**23.8**
Sort (Object sorting)	CaP-X	2/10	10.3	191	12.5	51.5	957	62.3
	CaP-X+Trace	4/10	9.1	171	12.8	22.8	**427**	**31.9**
	AF (Ours)	**5/10**	8.4	221	29.3	**16.8**	442	58.6

D Diagnostic Programs and Case Studies

The case studies examine the evidence available after task execution and how it informs completion assessment and program revision. They illustrate observation gaps in execution feedback and how AF acquires additional evidence to investigate unresolved outcomes.

D.1 Evidence for completion assessment

We present two separate trials from object pick-and-place tasks. The first illustrates a false completion declaration under CaP-X+Trace, where the inspected keyframes did not reveal whether the object had reached its destination. The second shows how Active Feedback (AF) acquired new viewpoints to identify unsuccessful placements despite the task program's success reports.

Missing outcome observations in execution traces. In a CaP-X+Trace trial, the task program attempted to place a hammer in a basket by moving to an estimated position above it, opening the gripper, and performing a visual check after moving upward. The following excerpts show the recorded execution signals and the observations available for completion assessment.

1. Task-program output: approach and release without placement confirmation

```
over basket: True
released; gripper: 1.0
[...]
hammer visible after release: None
```

The program set `over basket` to `True` when a motion to an estimated release position succeeded; this flag did not verify placement. After opening the gripper and moving upward, the program obtained no hammer detection. Neither the open gripper nor the absence of a detection established that the hammer had landed inside the basket. The agent declared FINISH, although the evaluator recorded that the hammer-placement requirement was unmet.

The agent inspected release-related trace keyframes, including the two shown below, before declaring `FINISH`. These views showed the open gripper and only part of the basket rim at the lower image boundary, leaving placement inside the basket unverified. This case illustrates an observation gap in execution-trace feedback: recording approach and release does not necessarily provide evidence of the resulting placement.

2. Inspected trace keyframes: placement remains unverified



(a) After gripper opening (step 200). The gripper is open, but the basket interior is outside the recorded view.



(b) After upward motion (step 203). Only part of the basket rim remains visible; the hammer's placement is not shown.

These selected keyframes show the observation gap, not direct visual evidence of task failure. Failure was established separately by the evaluator. Panel descriptions are author-written explanations, not agent-generated image notes.

Acquiring additional outcome observations with AF. The task program reported both tool placements as successful. The agent questioned whether the available evidence confirmed either outcome and generated a diagnostic program to inspect the destinations and nearby surfaces. The additional views revealed both tools near floor level, outside their intended destinations, leading the agent to select `REGENERATE`.

The excerpts below follow this sequence; quotations and code retain the recorded wording, with omissions marked by [...] or `# [...]`. Image notes were written by the agent before capture and express inspection questions, not observed conclusions.

1. Task-program output: reported placement success

```
hammer done: True
[...]
wrench done: True
```

2. Agent's recorded AF analysis: unresolved outcomes

"The previous program reports both parts done, but neither is confirmed."

Hammer. [...] "That shows the hammer left the hand, not that it landed inside the basket. It could have dropped beside the basket or onto the stairs. I need a view looking down into the basket."

Wrench. [...] "I need to look at the upper shelf more closely from above, then at the lower shelf and the floor in front of the cabinet, and write down where the wrench actually is."

The resulting diagnostic program acquired five views: the upper shelf, lower shelf, cabinet floor, basket from above, and area around the basket. It used map queries to locate the cabinet and basket, then image segmentation and depth to inspect the tools. The helper below connects a requested viewpoint to an agent-written note returned with the image.

3. Generated diagnostic code: move, observe, and attach a question

```
def look(target, offsets, note):
    target = np.asarray(target, float)
    for off in offsets:
        vp = target + np.asarray(off, float)
        if go(vp, target − vp):
            return get_observation(note=note)
    # [...]

fl = np.array([3.55, cc[1], 0.05])
o = look(fl, [[−0.8, 0, 1.0], [−1.0, 0.3, 1.1]],
         "Floor in front of the cabinet's open face: did the wrench fall to the floor?")
report(o, ["wrench"])
```

Four of the five acquired views are shown below, grouped by task requirement rather than capture order. They show the intended destinations and nearby floor areas, together with the agent-written inspection notes. The lower-shelf view is omitted from the panel. The accompanying measurements provide evidence of the tools' locations.

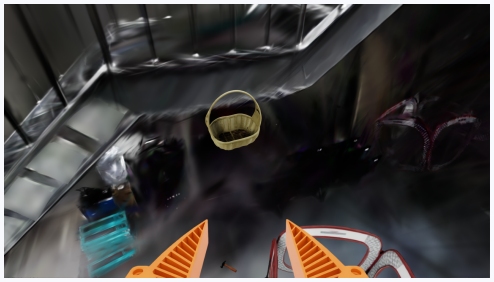
4. Returned visual evidence and the agent's image notes



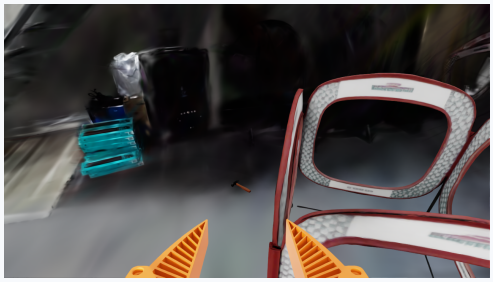
Upper shelf. "Upper open shelf of the cabinet from above-front: is the black-handled wrench from the pegboard on it?"



Cabinet floor. "Floor in front of the cabinet's open face: did the wrench fall to the floor?"



Basket interior. "Basket on the stairs from above: is the hammer inside it?"



Area around the basket. "Area below/around the basket and stairs: did the hammer fall out instead?"

5. Diagnostic console output: estimated tool locations

```
[wrench] score 0.91 [...] med [ 3.567  0.953 −0.009] [...]
[...]
[hammer] score 0.92 [...] med [ 2.463  3.979 −0.009] [...]
```

The images and median world-frame coordinates from segmented depth placed both tools near floor level, outside their required destinations. These were agent-visible perception estimates, not ground-truth evaluator outputs. The evidence established unsuccessful placement, without determining when or why either tool fell.

6. Decision and an excerpt from the next generated program

Agent decision: `REGENERATE`

```
def release_and_hold():
    open_gripper()
    p = pos()
    goto_pose(p, settle=True)        # hold still ~3 s so the object drops free
    get_observation()
    open_gripper()
```

The next program added this release-and-hold routine, revised the hammer release pose, and tried deeper wrench placements. These changes show how the observed failures informed revision; their effectiveness was still to be tested. The immediately following execution failed, and the trial succeeded after further refinement on the fourth task execution. Thus, this case supports AF's role in

obtaining evidence against completion and directing revision, while separating a useful diagnosis from an immediately successful repair.

D.2 Evidence for policy revision

We present two cases in which AF-generated physical experiments inform policy revision. In Box alignment, the agent tests and compares push actions to revise contact selection. In Payload, it tests a proposed basket load to revise the transport plan.

Box alignment: testing alternative contact points. After repeated pushes continued to rotate a carton, the agent used AF to test alternative contact locations and sideways slides. Excerpts retain the recorded wording, with omissions marked by [. . .] or `# [...]`.

1. Agent's recorded AF analysis: which push reduces rotation?

"My working guess is sideways friction drag." [. . .]

"I need controlled experiments to learn how much each action twists the carton: centre push, high-edge push, low-edge push, and a push followed by a small sideways slide while still in contact."

The program tested these actions sequentially, returning to the same viewpoint after each probe to measure the response from RGB-D observations.

2. Generated diagnostic code: compare contact actions

```
open_gripper()
# [...]
exps = [("centre", 0.0, 0.0), ("hi−edge", +1, 0.0),
        ("lo−edge", −1, 0.0), ("centre+drag+y", 0.0, +0.015),
        ("centre+drag−y", 0.0, −0.015), ("hi−edge", +1, 0.0)]
for name, where, drag in exps:
    # [...]
    r = go(G, approach=N, z_approach=0.15, allow_contact=True, settle=True)
    # [...]
    if drag != 0.0 and r:
        G2 = G + np.array([0.0, drag, 0.0])
        go(G2, approach=N, allow_contact=True, settle=True)
        # [...]
    # [...]
    f2, s2 = measure_B("after " + name)
```

The first high-y edge push produced less rotation per unit of advance than the centre and low-y edge pushes. Neither sideways slide reversed the rotation.

3. Returned diagnostic views and the agent's image notes



Before probing. "carton B untouched, head-on, before the push experiments"



After probing. "carton B after the six push experiments (centre, hi-edge, lo-edge, drag +y, drag -y, hi-edge)"

4. Decision and subsequent policy revision

Agent decision: `REGENERATE`

The next program initialized a linear push-response model from the diagnostic observations and used it to select the lateral contact position.

```
twist_data = [(0.0, −1.0, 1.0), (0.8, −0.2, 1.0), (−0.8, −1.7, 0.5)]
def twist_model():
    # [...]
    beta, alpha = np.polyfit(u, r, 1, w=np.sqrt(w))
    # [...]

# [...]
        w = y2 − y1; yc = 0.5 ∗ (y1 + y2)
        # [...]
        alpha, beta = twist_model()
        # [...]
        u = float(np.clip((rate_des − alpha) / beta, −0.82, 0.82))
        yp = yc + u ∗ w / 2.0
```

In the next task execution, the robot pushed on the high-y side, toward which the carton's center of mass was shifted, and precisely aligned the boxes.

Payload: testing a load capacity. The agent used AF to test whether the robot could lift a proposed basket load. The result informed a revised transport plan; a later AF round checked the final placements before the agent declared completion. Excerpts retain the recorded wording, with omissions marked by [. . .] or `# [...]`.

1. Agent's recorded AF analysis: a plan depends on an unknown limit

"The packing I planned needs baskets of 1.2 kg (red + yellow) and 1.0 kg (blue + black) with green carried alone. That only works if the limit is at least 1.2 kg." [. . .]

"I'll load red and yellow into the $-y$ basket and test-lift it ($\approx$1.2 kg)." [. . .]

The preceding execution had measured the bottle masses but stopped at an array-comparison error. Beyond fixing that error, the agent needed to check its proposed packing. It generated code to load the red and yellow bottles, grasp the basket handle, attempt a lift, and return the interface's response with a diagnostic image.

2. Generated diagnostic code: test the load and report the response

```
r, txt = captured(lambda: go(ap + np.array([0, 0, 0.2]), settle=True))
lim = "limit exceeded" in txt
P(f"LIFT TEST {tag}: result={r} kg={parse_kg(txt)} limit_exceeded={lim}")
get_observation(note=f"lift test {tag}: basket lifted? limit_exceeded={lim}")
```

3. Diagnostic response and the agent's image note

[drone thrust sensor] limit exceeded: the drone cannot lift what it is holding. Nothing moved.
LIFT TEST basket+red+yellow ~1.2kg: result=refused kg=None limit_exceeded=True



Agent-written note:
"lift test basket+red+yellow ∼1.2kg: basket lifted? limit_exceeded=True"

The refusal showed that the proposed load was infeasible; it did not identify the exact carrying limit. A subsequent intended 1.5 kg test failed to grasp the handle, so it supplied no additional capacity bound. The agent selected `REGENERATE`.

4. Next generated program: revise the planning limit

```
# payload: 0.9 kg bottle alone lifts, basket+1.1 kg of bottles (1.2 kg) is refused -> start from
    1.1 kg,
# and lower it automatically if the thrust sensor refuses a lift.
PAYLOAD_LIMIT = [1.1]
BASKET_KG = 0.1
# [...]
best = plan(pool, R, PAYLOAD_LIMIT[0])
```

The next program replaced its previous 1.5 kg assumption with a conservative 1.1 kg planning limit. It paired red with black and blue with yellow, and carried green separately. Both basket lifts succeeded, as recorded below. The revision also fixed the array-comparison error.

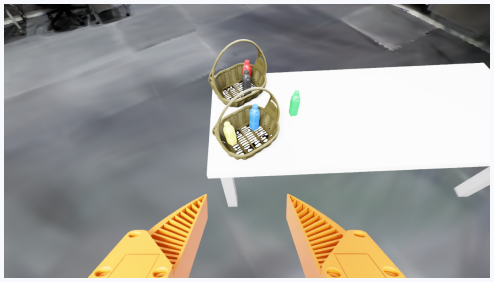
5. Next task execution: both planned basket loads lift successfully

basket0 lift: reached kg=1.1 limit_exceeded=False load~1.10
[...]
basket1 lift: reached kg=1.1 limit_exceeded=False load~1.10

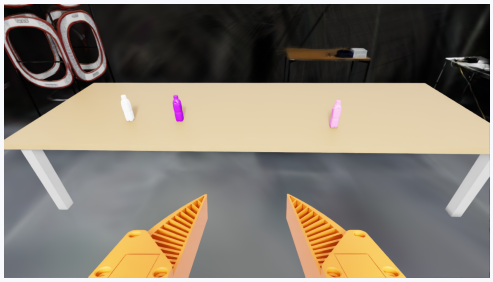
6. Next AF round: verify the distributed completion conditions

"The run finished without errors, but I haven't confirmed the final state yet." [...]

"I haven't seen both baskets full on the white table in one view." [...] "I haven't checked that only pink, purple and white remain there." [...]



Destination note: "white table overview: both baskets (red+black, blue+yellow) and green bottle present?"



Source note: "brown table: only the 3 empty bottles (pink, purple, white) should remain"

AF obtained two destination views and one source view, with image-based detections supporting five bottles and two baskets at the destination and three empty bottles at the source. The agent then declared `FINISH`; the supplied evaluation report records this execution as successful. The sequence illustrates how AF can first test a planning constraint and later check completion, rather than treating a successful lift or an error-free program return as sufficient evidence for the full task.

E Physical System and Deployment

E.1 Platform and Execution System

Aerial platform. Our custom-designed omnidirectional aerial manipulator comprises a coaxial quad-tiltrotor with eight rotors, four actuated tilt joints, a four-degree-of-freedom arm, and a parallel-jaw UMI gripper. The arm has three pitch joints and one wrist-roll joint. A wrist-mounted RGB-D camera provides the egocentric visual observations used by the task program. External cameras record the experiments for visualization and outcome assessment; their images are not supplied to the program. Figure 8 illustrates the robot configuration, and Table 7 summarizes the physical system.

E.2 Physical Experiments and Outcome Assessment

Starting poses and scene differences. The robot's position and heading at policy start, after take-off, vary across trials. Table 9 shows these starting poses for every evaluated trial. The physical scene contains the corresponding task objects but does not exactly reproduce their simulated placement or the surrounding obstacle arrangement. Table 8 compares task-object positions in a common laboratory coordinate frame. The digital twin and the motion-capture system share this frame, including its origin.

Successful cabinet water-bottle delivery task. Figure 11 shows a successful real-world execution of the cabinet task, complementing the tool pick-and-place execution in Figure 1. Figure 12 compares the same task in simulation and on the physical robot from the wrist camera.

Trial-specific outcomes. Table 10 lists the stage outcomes of every trial together with a short account of what happened.

Table 7: **Physical platform specifications.**

Item	Specification
Airframe	
Configuration	Omnidirectional coaxial quad-tiltrotor: four tilting arms, each carrying a coaxial pair of rotors (eight rotors, four tilt joints).
Dimensions	Airframe 0.62×0.69 m, height 0.20 m including rotors.
Mass	8.30 kg in flight configuration (battery, onboard computer, camera, and gripper included).
Motors	$8\times$ T-Motor VELOX V2812, 1155 KV (6S).
Propellers	$8\times$ T-Motor C7.5$\times$4.6 three-blade cinematic propellers (7.5 in).
Electronic speed controllers	$2\times$ HobbyWing XRotor FPV G2 4-in-1 ESC (65 A).
Battery	Two 6S1P semi-solid-state lithium-polymer packs (HYBRID PT-B18000-SS-6S), 22.2 V, 18,000 mAh.
Tilt actuators	$4\times$ ROBOTIS DYNAMIXEL XM430-W210-T (one per arm).
Manipulator and gripper	
Arm	Four degrees of freedom (three pitch joints and one wrist-roll joint), actuated by $2\times$ DYNAMIXEL XM540-W270-R and $2\times$ DYNAMIXEL XM430-W210-R.
Gripper	Parallel-jaw UMI gripper driven by one DYNAMIXEL XM430-W210-R. Symmetric left and right finger links are coupled through a mimic joint so that both fingers move evenly; maximum opening 80 mm.
Sensing and state estimation	
Wrist camera	Intel RealSense D455 RGB-D camera (active stereo depth, global-shutter RGB), operated at 1280×720 for both color and depth at 30 Hz with depth aligned to color.
Onboard IMU	Holybro Pixhawk 6C flight controller with its built-in IMUs.
External localization	OptiTrack motion capture (Motive) providing rigid-body pose at 100 Hz.
State estimation	Extended Kalman filter fusing the onboard IMU with the external pose measurements.
Computation and control	
Onboard computer	ASUS NUC 15 Pro with an Intel Core 7 processor and 32 GB memory.
Flight controller	Geometric nonlinear PID controller on SE(3) (Goodarzi et al., 2013), running position and attitude control in a single 333 Hz (3 ms) loop.
Off-board perception	Workstation with an NVIDIA RTX A5000 GPU serving the perception and semantic-mapping models (Appendix A.4).
Communication	ROS 2 Humble (DDS) over Wi-Fi between the onboard computer, the ground laptop, and the perception workstation.

Table 8: **Scene layout in the laboratory digital twin and the physical laboratory.** Coordinates are in the shared world frame (metres). Simulation values are ground-truth poses from the scene description. Physical positions are motion-capture measurements at the closest corresponding point on each object, so the comparison is approximate rather than exact.

Item	Simulation (x, y, z)	Physical (x, y, z)
Target objects		
Hammer	$(0.78, -3.00, 1.68)$	$(0.84, -2.91, 1.41)$
Wrench	$(0.96, -3.01, 1.41)$	$(1.02, -2.91, 1.38)$
Water bottle	$(3.79, 0.98, 0.85)$	$(3.29, 0.40, 0.83)$
Task fixtures		
Pegboard front face (y)	-3.05	-2.95
Hanging basket (center)	$(2.57, 4.01, 1.86)$	$(2.22, 3.37, 2.01)$
Cabinet (center)	$(3.91, 0.97, 0.86)$	$(3.45, 0.40, 0.91)$
Upper shelf top (z)	1.42	1.40
Door knob (left)	$(3.68, 1.07, 0.98)$	$(3.22, 0.49, 1.04)$
Door knob (right)	$(3.68, 0.87, 0.98)$	$(3.22, 0.31, 1.03)$

Table 9: **Starting poses of the hardware trials.** Airframe position and heading at policy start after takeoff, in the shared world frame.

Task	Trial	Position (x, y, z) [m]	Heading [°]
Tool pick-and-place	1	$(-0.66, 0.00, 1.86)$	$+1.8$
	2	$(-0.90, -0.01, 1.86)$	-0.3
	3	$(-0.97, -0.04, 1.86)$	-0.1
	4	$(-0.87, -0.04, 1.86)$	-0.6
	5	$(-0.71, 0.33, 1.86)$	$+41.3$
Cabinet delivery	1	$(-1.19, -0.39, 1.86)$	$+20.4$
	2	$(-0.91, -0.26, 1.86)$	$+20.2$
	3	$(-1.26, -0.24, 1.86)$	$+22.0$
	4	$(-1.56, 0.08, 1.86)$	$+90.7$
	5	$(-1.08, -0.06, 1.84)$	$+34.6$

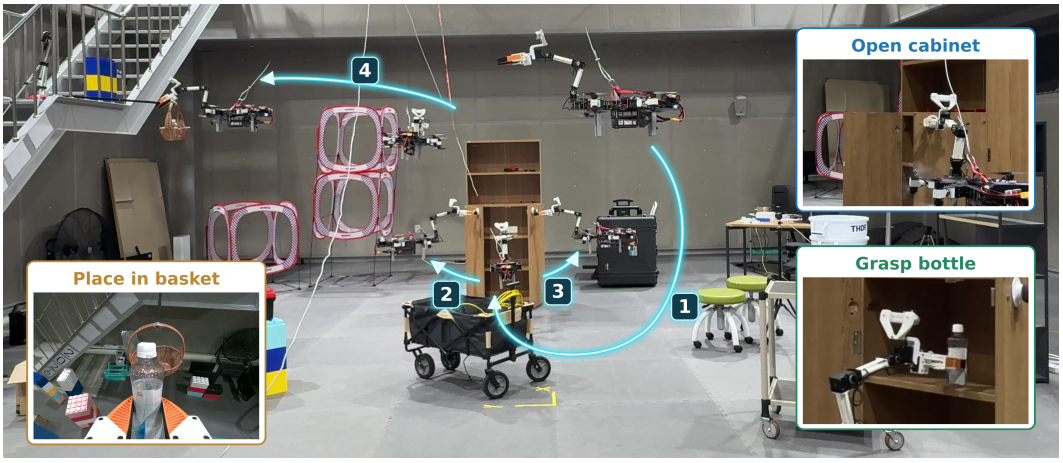


Figure 11: **Real-world cabinet water-bottle delivery.** External views of one successful trial: opening the left door, opening the right door, grasping the bottle, and placing it in the hanging basket.

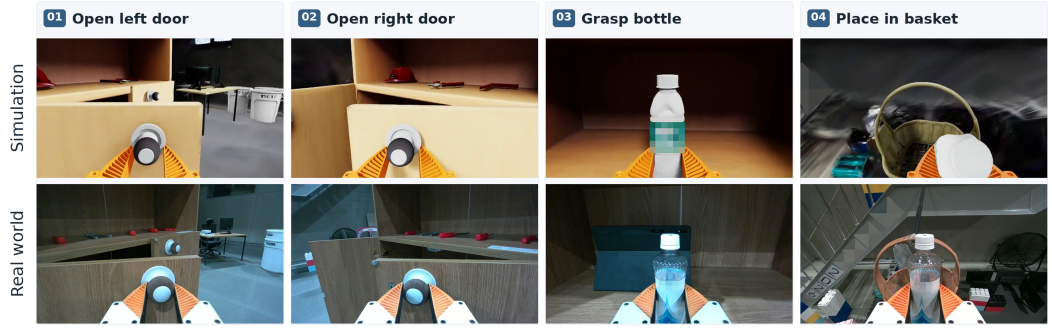


Figure 12: **Cabinet water-bottle delivery in simulation and the real world.** Wrist-camera views show opening the left door, opening the right door, grasping the bottle, and approaching the basket before release. The top and bottom rows show simulation and physical execution, respectively.

Table 10: **Trial-level records of the real-world trials.** ✓: completed, ×: failed.

Trial	S1	S2	S3	S4	What happened
Tool pick-and-place (S1 grasp hammer, S2 place in basket, S3 grasp wrench, S4 place on upper shelf)					
1	✓	✓	✓	✓	–
2	✓	✓	✓	✓	–
3	✓	×	✓	✓	A perception error in the detected basket position shifted the release point, and the hammer struck the basket rim on release and fell outside.
4	✓	✓	✓	✓	–
5	✓	✓	✓	✓	–
Cabinet water-bottle delivery (S1 open left door, S2 open right door, S3 grasp bottle, S4 place in basket)					
1	✓	✓	✓	✓	–
2	✓	✓	✓	✓	–
3	✓	×	×	×	After releasing the left knob, the gripper became stuck on the handle and the robot could not move to the next waypoint; all subsequent motion calls were refused, and the trial failed.
4	✓	✓	✓	×	A control error caused the gripper to grasp the lower part of the bottle, so the bottle occluded the wrist camera. The basket could therefore not be re-measured at close range, and the program failed to place the bottle inside it.
5	✓	✓	✓	✓	–