VLMs Play Robotics: Can VLMs Control Locomotion and Manipulation?

Anonymous authors
Paper under double-blind review

Abstract

Vision-language models (VLMs) can reason over images and write code, but whether these abilities transfer to robotic control is unclear. To answer this question, we define several interfaces—interaction patterns—that correspond to different resources that VLMs commonly have access to during deployment. We then evaluate frontier and open-weight models on those interfaces across toy tasks, quadrupeds, humanoids, and robotic arms. We find that models are stronger at higher-level control, although nearly every capability we test shows generational performance gains within each model family. Specific interface and tool choices change performance by an order of magnitude, meaning narrow evaluations may under- or overestimate a model's ability to exert control. Though overall success rates are low and other factors such as latency prevent effective deployment, our experiments indicate that generalist VLMs possess nascent robotic capability.

1 Introduction

Vision-language models (VLMs) have proven to be capable reasoners over images, code, and text (OpenAI, 2023). Many of these skills are also necessary for robotics, and it is an open question whether they can be repurposed for this use case as well. Robotic tasks require strong perception of 3D space, understanding of physical dynamics, and the ability to output precise commands. We seek to understand if the abilities involved in image reasoning and programming can be repurposed to affect the physical world.

Studying this also requires defining and evaluating what it means for a VLM to control a robot. Existing work does not settle the question of how models behave across a wide spectrum of interaction modes. Most evaluations cast the model as a planner over predefined skills, or as a writer of code and rewards, or they score specialist vision-language-action policies on their own (Ahn et al., 2022; Ma et al., 2024; Li et al., 2025; Yang et al., 2025b). To understand capability, it is insufficient to test models under a narrow set of conditions because of the vast scope of potential environments across which they can be deployed.

Therefore, we first identify four common usage patterns for VLMs. The first, direct control, corresponds to a sandboxed language model which can only output natural language commands without external tools. The second, code-based control, corresponds to being given access to a Python interpreter, but no additional hardware or policy networks that could assist it with the task. The third is policy supervision, where the model is given a pretrained policy that can assist it in performing its task. The last, training supervision, corresponds to when the model is given hardware that enables it to actually train a new policy and deploy it arbitrarily. We note that these are not necessarily hierarchical but do correspond to different levels of complexity and require different skills.

We use these definitions to benchmark frontier and open-weight VLMs on fixed tasks while varying the interface across them. Our experimental design also lets us ask how far measured ability changes across a spectrum of control, which abilities improve across model generations, and which extra inputs help a model exert physical control.

We find that performance on identical tasks ranges widely with the interface, determine which abilities are developing fastest, and identify the structured inputs that provide the greatest gain across models. For locomotion, direct control of complex robots is generally out of reach except on simple tasks, but newer models show stronger competency on medium-complexity robots like quadrupeds

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.

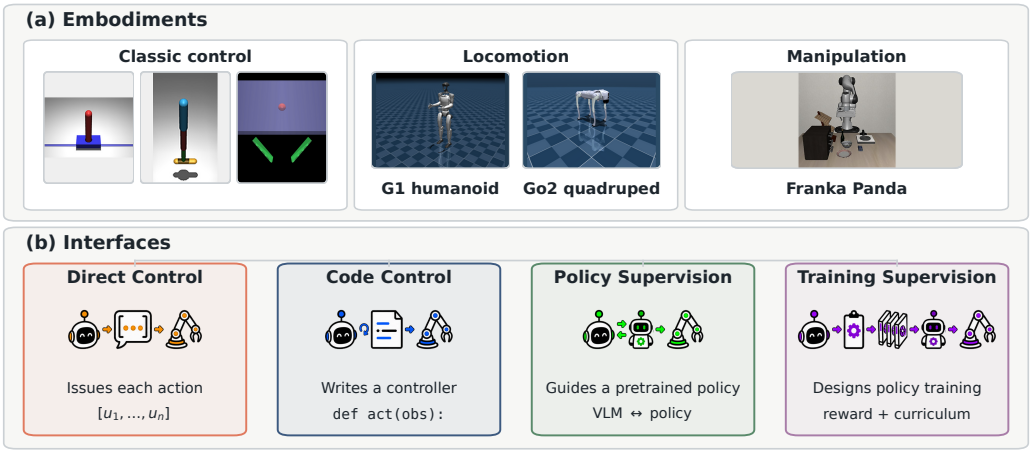


Figure 1: **We test frontier and open-weight VLMs on three categories of tasks across four common usage patterns, or interfaces.** The top row shows the robots and environments in our evaluation; the bottom row shows how model outputs become robot behavior. Table 2 identifies the combinations we test.

under code-based control and training supervision. For manipulation, success on direct control is growing steadily, albeit very slowly. Under every task we tested, models showed the ability to take advantage of pretrained specialists and newer models even show signs they recognize deficiencies in the specialist.

The question of real-world embodiment capability is significant if we seek to deploy safe and capable models. In controlled evaluations, models have already attempted to disable oversight or taken harmful actions to pursue goals (Meinke et al., 2024; Lynch et al., 2025). The environments in which risk is evaluated must correspond to the environments of realistic deployment. Although we cannot exhaustively rule out higher performance in our situations, our evaluation provides a general understanding of which areas VLMs are especially adept at, which areas have seen improvement through model generations, and which are likely to have low levels of risk.

Our contributions are as follows. **(i)** We develop a framework for embodiment capabilities and benchmark frontier and open-weight VLMs across four control interfaces on task families spanning classic control, locomotion, and manipulation, and show that interface choice can change which model appears competent. We release the environments and code here. **(ii)** We report a snapshot of foundation model performance on robotic tasks, and identify which tasks and interfaces are improving the fastest. **(iii)** We identify several bottlenecks that limit performance, including little learning over a long horizon, imprecise visual understanding, and an inability to combine what a model sees with what its sensors report.

2 Related Work

Robotics and embodied-agent benchmarks. Previous robotics benchmarks measure task completion across tabletop manipulation, household activities, locomotion, navigation, and whole-body control (Liu et al., 2023; Li et al., 2022; Sferrazza et al., 2024; Zhang et al., 2024; Atreya et al., 2025; Yang et al., 2025b). They contain diverse and realistic tasks but usually fix a single action space or controller. Therefore, their main goal is to compare systems within one setting rather than distinguishing performance on different control interfaces. There are also benchmarks that test physical reasoning, spatial understanding, manipulation knowledge, or planning without requiring end-to-end control (Chow et al., 2025; Qiu et al., 2025; Dai et al., 2025; Zhao et al., 2025; Luo et al., 2026). Such evaluations show whether a model understands a physical situation, but not whether it can use that understanding to complete tasks. We instead run complete interactive trials where the model must control the robot every step of the way.

General-purpose language models as robot controllers. General-purpose vision-language models have been connected to robots through several interfaces. They can produce low-level targets that are executed by a controller (Tang et al., 2023; Wang et al., 2024), or plan over predefined skills (Ahn et al., 2022; Huang et al., 2022; Wu et al., 2023). In the latter setting, the model chooses the next behavior while the controller determines the details of motion. VLMs can also generate executable programs or the controllers themselves (Liang et al., 2023; Singh et al., 2022; Huang et al., 2023), or write objectives used to train a policy (Ma et al., 2024; Xie et al., 2024). Recent evaluations compare programming and tool-use performance (dos Santos et al., 2026; Fu et al., 2026; Tsui et al., 2026), but these mechanisms are generally studied separately. In our evaluation, we compare direct commands, generated controllers, pretrained-policy access, and supervision of policy training within a single framework.

Specialist robot policies and generalist–specialist systems. Specialist vision-language-action policies learn from robot data to map observations and instructions directly to actions; examples include RT-2, OpenVLA, π_0, $\pi_{0.5}$, and MolmoAct (Brohan et al., 2023; Kim et al., 2024; Black et al., 2026; Intelligence et al., 2025; Lee et al., 2025). Their evaluation primarily measures capability acquired through robot-specific training. Generalist–specialist systems instead pair a general-purpose model's high-level reasoning with a trained policy's low-level behavior (Shi et al., 2025; Hu et al., 2026; Chen et al., 2026; Wu et al., 2025; Yang et al., 2025c), although specialist policies can be sensitive to changes in cameras, scenes, initial states, and instructions (Fei et al., 2025; Zhou et al., 2026). We keep the general-purpose model as the subject and treat a specialist as one possible resource: under policy supervision, the model decides how to use or intervene on a pretrained manipulation or locomotion policy. This allows direct comparison with other ways for the same kind of model to control a robot.

3 Evaluation Framework

We test four control interfaces over tens of thousands of trials, each of which spans 50,000–1,000,000 tokens. In each experiment, a VLM is the primary subject of the evaluation and we assume the assistance it is given—for example, a pretrained Vision-Language Action Model (VLA)—is held constant. It interacts with the robot through a turn-based environment, where it receives sensory information, thinks, emits commands and tool calls, and then receives additional sensory information. Depending on the trial, the sensory information comprises images, acceleration, depth, joint positions, distance from some objective, natural language information about the physical environment, or other pieces of relevant information. Appendix B summarizes the prompt differences and gives the templates we used; our code contains every expanded prompt and tool definition.

The four interfaces are as follows:

- **Direct Control.** The model issues raw forces, torques, or end-effector deltas to the robot. These instructions are then executed, and sensory information is returned. Due to latency constraints, we conduct these experiments in simulation where the simulator pauses while the model computes its next command, so inference latency does not bound the score. Thus, our results reflect an upper bound on performance, which we expect to be more relevant as VLMs get more efficient. Additionally, certain Direct trials end after an unsuccessful episode, while Code always allows retries after failure (App. A.1).
- **Code Control.** The model writes a Python script which emits numerical commands to the robot in lieu of producing them itself.
- **Policy Supervision.** In this setting, the VLM can query a pretrained specialist that produces the robot's actions. This takes two forms in our evaluation: supervising a manipulation VLA by accepting, modifying, or rejecting its proposed actions before they reach the robot, and sending joystick-like velocity targets to a pretrained walking policy.
- **Training Supervision.** Here, the model is separated entirely from the rollout of its embodiment, and is instead given a goal for which it must design a curriculum and rewards so that it can train a policy to accomplish the goal autonomously. At training time, the model is provided privileged access to the simulator environment.

Table 1: **Task scores and success criteria.** A dash denotes no binary success criterion. *The high-level Go2 tasks and standard commands are described in App. E; other settings are summarized in Appendix A.1.

Task	Score	Success
Classic control (MuJoCo)		
Inverted Pendulum	Balance time (up to 20 s)	20 s balanced
Hopper	Peak forward speed	—
Twin-Flipper	Total ball airtime	—
Locomotion (12 DOF Go2; 29 DOF G1)		
Stand up	Best stand-up progress	2 s upright
Balance	Longest upright streak	2 s upright
Go2 5 m walk	Longest distance	5 m without falling
High-level Go2 (11)	Success*	Varies by task
Manipulation (7 DOF Franka Panda)		
LIBERO-40 (Liu et al., 2023)	Final success rate	Goal completed

Unless otherwise specified, trials auto-reset on termination, and the model is allowed to continue from the initial state. All of our experiments are conducted in simulation.

3.1 TASKS

In each trial, we pair an interface with a task. We identify three categories of tasks that are salient to a broad swathe of research and realistic deployments:

- **Classic Control.** Not all tasks comprise complex systems with high degrees of freedom. For instance, industrial control often uses simple PID controllers (Ang et al., 2005). We identify two well-established control tasks, the Inverted Pendulum and Hopper tasks (Todorov et al., 2012; Brockman et al., 2016), and create a novel task called Twin-Flipper (Figure 1) to see if models can navigate low-dimensional environments.
- **Locomotion.** A locomotion task is one in which an embodied agent must move through an environment to reach a target location or state (Siegwart et al., 2011). To make our evaluation more general, we test a humanoid robot, the Unitree G1, and a quadruped robot, the Unitree Go2. Our low-level tasks involve stabilizing, getting up from the ground, and walking. We also test eleven high-level Go2 tasks in which a pretrained policy supplies balance and walking. They ask the model to navigate around visible and hidden obstacles, remember information across a route, plan from an overhead map, and correct motion errors from visual feedback. As we do not train specifically on these embodiments, we believe our evaluation should extend to other similarly complex embodiments.
- **Manipulation.** A manipulation task is one in which an embodied agent physically interacts with objects to change their position, orientation, or state. We use LIBERO-40 for our evaluation with a Franka Panda arm.

For each experiment, we develop success criteria and proxy metrics in situations where task success is rare and thus noisy. Table 1 gives each task's scoring and success condition where one exists. Appendix A.1 summarizes task timing, observations, scoring, and reset rules; trial-level results accompany the code release.

To decide which parts of the grid to run, we conducted preliminary trials to determine which setups would show a gradient of performance.

Additionally, some interfaces are not applicable to some tasks or were not evaluated because of cost; for example, the low-dimensional tasks do not have commonly used pretrained policies. Table 2 shows which trials we ran.

Our evaluation is designed to find current capabilities and bottlenecks and how those have changed over time. Thus, we chose the strongest proprietary models available at the time, which were GPT-5.4, Mythos Preview, and Gemini 3.1, and then added additional checkpoints from those families to

Table 2: **Which robots and control interfaces we evaluated.**

Robot or platform	Direct	Code	Policy	Training
Inverted Pendulum	✓	✓		✓
Hopper	✓	✓		✓
Twin-Flipper	✓	✓		✓
Unitree Go2	✓	✓	✓	✓
Unitree G1	✓	✓		✓
Franka Panda	✓		✓	

evaluate progress over time. We then added the two strongest open-weight candidates from our pilot runs, Kimi K2.6 (Team et al., 2026) and Qwen 3.6+ (Yang et al., 2025a). The full endpoints, access dates, sampling, and reasoning settings are given in App. C.1.

4 Experiments

We run experiments on multiple model families in order to gauge performance and how it has progressed on the different interfaces. The interfaces and tasks are not intended to form a hierarchy: a model may perform well under code control while remaining a poor training supervisor, or vice versa.

4.1 Classic Control Tasks

We begin with simple tasks, which use few action variables and let us study whether VLMs can understand simple kinematics and dynamics and control tractable systems in order to accomplish their goals. We use three tasks—the Inverted Pendulum, Hopper, and the novel Twin-Flipper task—to test this. Timing and scoring details are in App. A.1.

Code control improves Classic Control tasks over direct control. The improvement is not universal—for example, Mythos Preview performs worse on Hopper under code control—but the aggregate pattern is that giving models the ability to express a controller as code improves performance on simple, tractable tasks.

Training supervision requires distinct skills and is task dependent. Training supervision, by contrast to code control, has more task-dependent performance. For example, some models show better performance under training supervision than under code control, especially on the Hopper task. On Twin-Flipper, code control is usually stronger. Training supervision usually outperforms direct control, but Opus 4.6 and Gemini 3.1 do worse on Twin-Flipper.

Direct control is improving more slowly than programmatic control. Direct control performance increases modestly for the Claude family of models, especially from the earliest two iterations. Training supervision is similar, with a strong increase in performance followed by smaller gains in recent generations such as Opus 4.7.

Newer models improve more across retries. On the classic-control tasks in Appendix Figure 22, later Opus versions improve more across retries than earlier versions. First-attempt scores change less across versions, while the best scores after retries rise sharply.

4.2 Locomotion Tasks

Next, we test locomotion on the Go2 quadruped and the more complex G1 humanoid. We focus on primitive behaviors rather than full-length tasks because even these basic skills require many interaction steps, suggesting that longer tasks would strain the model's context window or suffer from low performance under direct or code control.

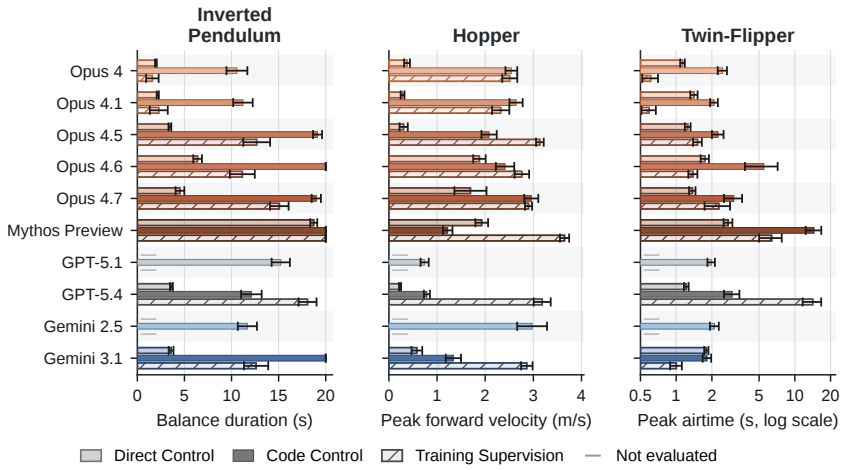


Figure 2: **Code and Training supervision usually outperform Direct control, but neither is consistently stronger across tasks.** Dashes mark untested interfaces; App. Fig. 9 reports full results.

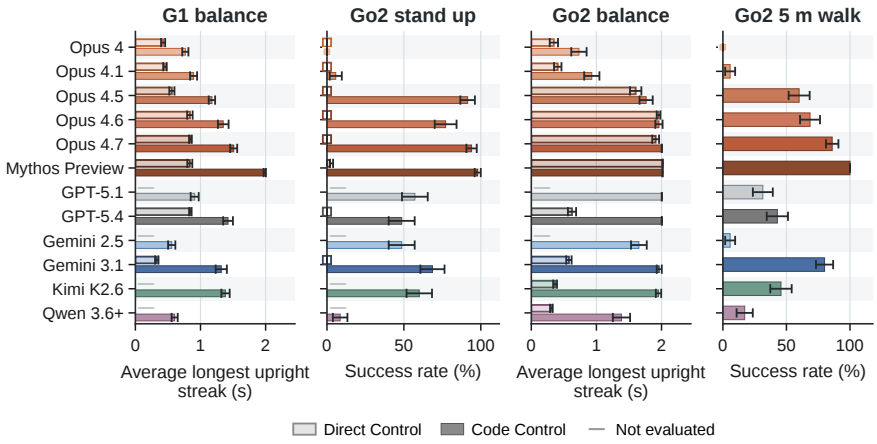


Figure 3: **Code control improves low-level locomotion and shows large generational gains.** Dashes mark untested interfaces; detailed open-weight results are in App. Table 5.

Code control and training supervision improve quadruped balance and recovery. Figure 3 compares Direct and Code control on G1 and Go2 balance and Go2 stand up; it separately reports Code control on the Go2 5 m walk. Code improves performance for nearly every model, as does training supervision (Fig. 4).

Humanoid robots cannot be controlled reliably by language models alone. No model evaluated on this task successfully stood the humanoid G1 robot up from the ground in any trial (Table 4). Although newer models can keep it balanced for longer, reliable humanoid control remains out of reach without additional tools or resources.

Neither code nor training supervision dominates. For instance, GPT-5.4 is the most proficient training supervisor, although its Code success rate lags far behind its peers.

4.2.1 High-Level Go2 Policy Supervision

Next, we focus on the ability of VLMs to perform higher-level tasks with assistance from a pretrained policy. This setup is well studied in prior work (Ahn et al., 2022; Huang et al., 2022; Wu et al., 2023; Shi et al., 2025). In the standard interactive setup, the VLM uses a forward-facing image to send

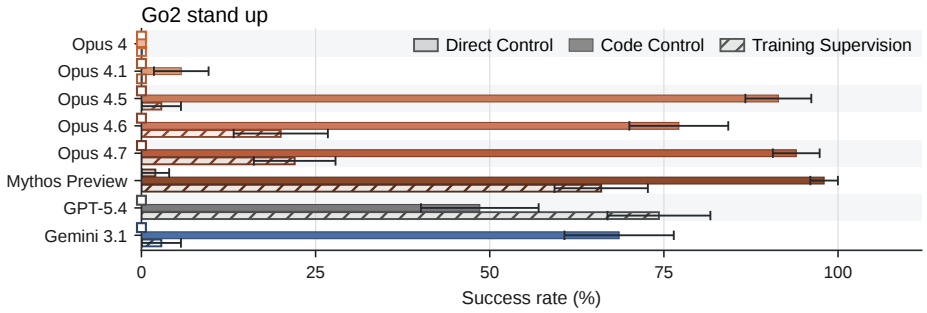


Figure 4: **On locomotion, some models perform better under Code control, and some under Training supervision.** Bars compare final success rates under Direct, Code, and Training supervision. While Code allows retries after a failed stand-up attempt, Direct trials end on failure. App. Table 4 reports the corresponding values and the all-zero G1 results.

movement commands, and the policy handles balance and joint actuation. This allows the VLM to focus on higher-level skills like scene understanding, spatial memory, and route planning. We test eleven tasks spanning visual navigation, memory, planning, and error correction; Appendix E describes each task, the standard command interface, and the one-shot exception.

Higher-level locomotion improves across model generations. The left panel of Figure 5 shows the average across all tasks. The largest early generational gain is from Opus 4.1 to Opus 4.5. Opus 4.5, 4.6, and 4.7 then remain close together before Mythos Preview improves again.

The tool that helps the most with high-level locomotion is one of the simplest. We also test whether additional inputs—a compass heading, depth overlay, center crosshair, third-person camera, and all four inputs together— can assist the model on task completion. We describe these inputs and show results by task type in Appendix E. The right panel of Figure 5 shows that the compass produces the largest improvement and helps every tested model (App. Fig. 12). Combining all inputs helps less than the compass alone. More information is not always more useful and can be detrimental if the model is unable to exploit it.

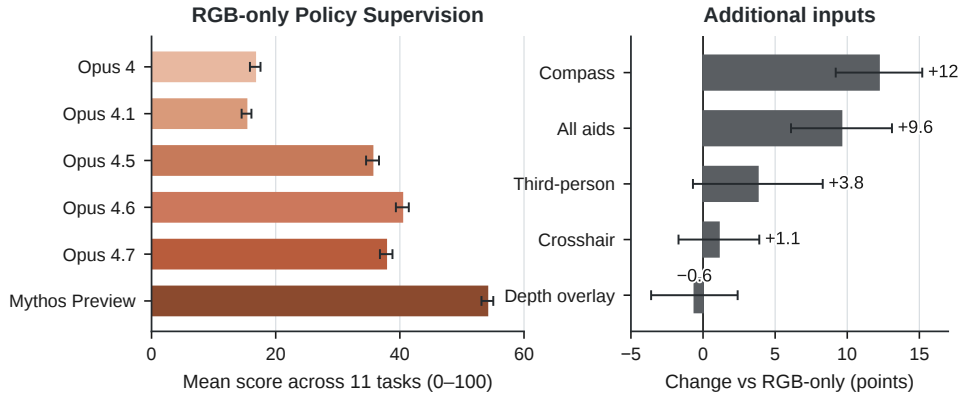


Figure 5: **Performance on Go2 tasks improves considerably with tools, but the compass helps the most.** The left panel shows the best measured RGB-only result at each checkpoint; the right shows the average score change from each additional input.

4.3 Manipulation Tasks

We next test manipulation across forty LIBERO tasks with a fixed-base Franka arm. This is our most unique direct-control setting, because physics does not need to be paused while the model is thinking as the scene is static. The model works through a slow cycle of moving the arm, observing the

updated scene and gripper state, and using that feedback to choose its next movement. Manipulative control therefore also tests scene understanding and iterative correction.

Policy supervision provides a strong uplift over direct control. Figure 6 shows that Policy supervision raises final success, although every supervised model performs worse than MolmoAct, the pretrained policy, alone.

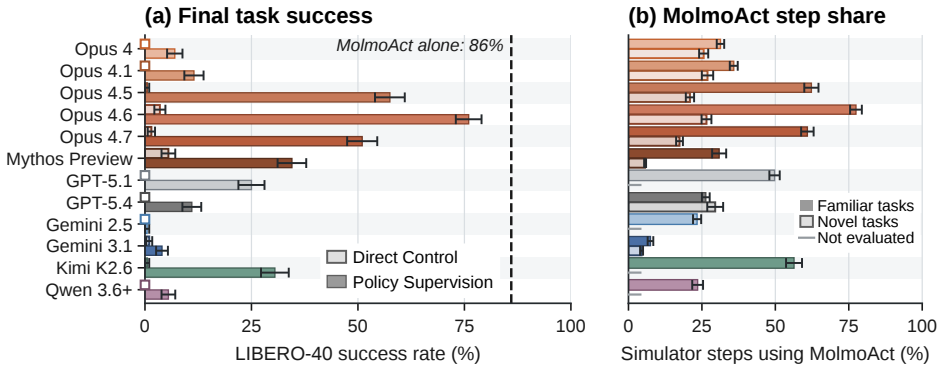


Figure 6: **Policy supervision raises familiar-task success, while models trust MolmoAct less on novel tasks.** Left: final success on the same LIBERO-40 tasks, with MolmoAct alone as a reference. Right: The percentage of commands that MolmoAct proposes that the VLM sends to the robot.

Performance generally increases under both interfaces, but not uniformly. Direct and Policy supervision improve over the earliest Claude and Gemini models, with larger gains under Policy supervision. Later models do not always perform better, however, as Policy supervision success falls from 76% with Opus 4.6 to 51% with Opus 4.7 and 34.5% with Mythos Preview. Even on the newest models, direct-control success remains low, and improvement is most clearly observable on intermediate objectives like contact with or grasp of the target object (App. Fig. 17).

Models can recognize when a policy is less proficient. To test whether models can correct a policy that does not perform well, we created three novel manipulation tasks outside MolmoAct's training distribution (App. F). MolmoAct never completes these tasks alone, but some VLMs like Mythos Preview complete a task while supervising it (App. Fig. 18). However, the right panel of Figure 6 shows that most measured models use MolmoAct actions for fewer simulator steps on the novel tasks than on familiar ones. This suggests that models can recognize when the specialist is less reliable, even when they cannot yet correct it successfully.

Targeted 3D information helps more than dense visual overlays. Finally, we test what information may be missing from the model's RGB observations. We pick ten LIBERO tasks and run additional trials where we provide the model either with a scene depth map, a segmentation mask, or a selectable point that returns the identity, depth, and 3D position at one location. We describe these inputs in Appendix G (Fig. 19). Figure 7 shows that the selectable point produces the largest average gain in final success. Depth and segmentation do not help consistently, suggesting that a targeted geometric query is more useful than adding dense information across the entire image.

5 Analysis and Implications

Taken holistically, our results reveal several trends that hold true for multiple model families.

Models accomplish far more when they do not have to issue every action directly. Across all tested task categories, Code control, Policy supervision, and Training supervision often substantially outperform Direct control. Policy supervision produces the strongest manipulation results and supports the most advanced tasks we evaluate, although every supervised VLM still performs below the underlying policy on in-distribution tasks.

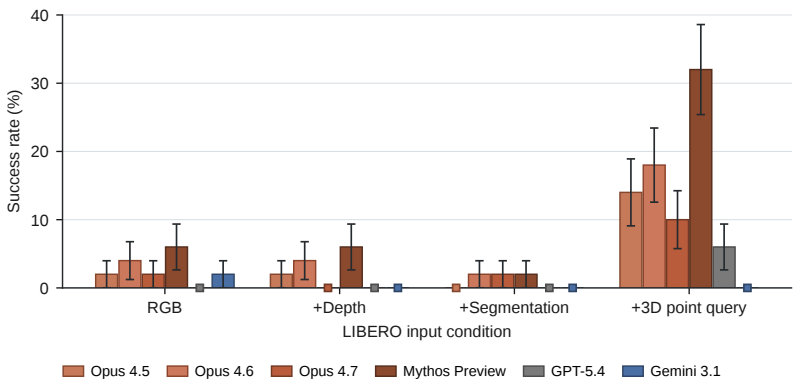


Figure 7: **Among several available tools, the selectable 3D point query is the most helpful on manipulation tasks.** All four input conditions use the same ten LIBERO tasks. App. Table 6 shows the complete results.

There is no fixed hierarchy of performance among interfaces. Although newer models tend to benefit from methods of interaction beyond direct control, we find no consistent ordering among the other interfaces we tested. Where interfaces are evaluated on the same task, their ordering changes across models and tasks because each tests a different ability.

Code control and Policy supervision are improving faster than Direct control. Across model generations, we see the largest improvements appear when models write reusable controllers or supervise pretrained policies, especially on the more demanding locomotion and manipulation tasks. Direct control is improving more slowly, and on some tasks we see gains only through intermediate objectives and not task success. These differences are not uniform across every model version or family, but newer models increasingly distinguish themselves by how effectively they control through Code and Policy supervision.

Models show signs of recognizing when external policies are incompetent. In manipulation, models rely less on a pretrained policy when it is applied outside its familiar setting, yet they almost never recover well enough to complete the new task (Figure 6). These results suggest the VLMs we test know that the pretrained policy is giving incorrect instructions but do not yet possess the ability to rectify them.

Providing more information does not necessarily improve robot control. Experiments with additional inputs show that task-relevant position or heading information helps more consistently than full-image visual overlays or collections of multiple aids. Some additional inputs have little effect or can even reduce performance; they may distract or misdirect the model. Together with the interface results, this suggests that assistance is valuable when it addresses the specific difficulty preventing success, not simply when it gives the model more information, tools, or computation.

6 Conclusion

In this work, we show that VLMs do not need substantial robot-specific training to exhibit meaningful control capabilities. Even the naked models, with no extra tools, show a clear improvement in performance for newer generations. Although other factors such as latency may prevent them from being deployed in the long term, current models possess the intelligence required to embody themselves. We hope our framework can inform how AI models are deployed safely, while also identifying where advances in general-purpose models may translate into stronger robotic control.

AI use statement

We evaluate frontier and open-weight VLMs as the subjects of this work. LLM chatbots and coding tools also assisted with code, figures, literature and bibliography checks, interpretation of results,

revising and grammar-checking, and LaTeX formatting. The authors reviewed the work, verified the scientific claims and analyses, and take full responsibility for the final content.

Ethics Statement

We have read and adhere to the Code of Ethics. All experiments are conducted in simulation and involve no human subjects, personally identifiable data, or physical robots.

This benchmark aims to help researchers assess how well general-purpose models can control robots as those models improve. Our comparisons can help evaluators identify capabilities that a single interface would miss, and help robotics researchers decide which parts of a task can be assigned to a VLM. Doing the tasks in simulation also allows us to study failures without exposing people or physical equipment to them.

However, like any capabilities or safety evaluation, our results have the potential to cause harm. For instance, knowing the capability bounds makes it easier to know which models are capable of misuse. Publishing the interfaces and prompts may reduce the effort needed to use these capabilities, including for harmful purposes. These are potential consequences of deployment and release; our experiments do not measure their likelihood or severity. The value of independent evaluation motivates release even with these risks.

We also note that any failures we report do not guarantee safety because task success alone is insufficient evidence of safety. Successful actions may be unsafe, and an unsuccessful attempt may still cause harm. Deployment would require separate evaluation of physical risks and involve significantly more human oversight on individual actions.

Reproducibility Statement

We release the environments, the tasks and the evaluation harness so that the results can be re-run. Appendix A.1 summarizes task timing, observations, scoring, and reset rules; Appendix E describes the high-level Go2 tasks and standard commands; and Appendix B explains how prompts differ and gives representative templates. Our code, which we release, contains the full prompts and other information required for reproducibility.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances, 2022. URL https://arxiv.org/abs/2204.01691.

Kiam Heong Ang, Gregory Chong, and Yun Li. PID control system analysis, design, and technology. *IEEE Trans. Control. Syst. Technol.*, 13(4):559–576, 2005. doi: 10.1109/TCST.2005.847331. URL https://doi.org/10.1109/TCST.2005.847331.

Pranav Atreya, Karl Pertsch, Tony Lee, Moo Jin Kim, Arhan Jain, Artur Kuramshin, Clemens Eppner, Cyrus Neary, Edward Hu, Fabio Ramos, Jonathan Tremblay, Kanav Arora, Kirsty Ellis, Luca Macesanu, Marcel Torne Villasevil, Matthew Leonard, Meedeum Cho, Ozgur Aslan, Shivin Dass, Jie Wang, William Reger, Xingfang Yuan, Xuning Yang, Abhishek Gupta, Dinesh Jayaraman, Glen Berseth, Kostas Daniilidis, Roberto Martin-Martin, Youngwoon Lee, Percy Liang, Chelsea Finn, and Sergey Levine. Roboarena: Distributed real-world evaluation of generalist robot policies, 2025. URL https://arxiv.org/abs/2506.18123.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James

Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2026. URL https://arxiv.org/abs/2410.24164.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. URL https://arxiv.org/abs/1606.01540.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023. URL https://arxiv.org/abs/2307.15818.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Siyi Chen, Hugo Hadfield, Alex Zook, Mikaela Angelina Uy, Chan Hee Song, Erwin Coumans, Xuning Yang, Faisal Ladhak, Qing Qu, Stan Birchfield, Jonathan Tremblay, and Valts Blukis. Volo: A physical orchestrator for open-vocabulary long-horizon manipulation, 2026. URL https://arxiv.org/abs/2606.07723.

Wei Chow, Jiageng Mao, Boyi Li, Daniel Seita, Vitor Guizilini, and Yue Wang. Physbench: Benchmarking and enhancing vision-language models for physical world understanding, 2025. URL https://arxiv.org/abs/2501.16411.

Song Dai, Yibo Yan, Jiamin Su, Dongfang Zihao, Yubo Gao, Yonghua Hei, Jungang Li, Junyan Zhang, Sicheng Tao, Zhuoran Gao, and Xuming Hu. Physicsarena: The first multimodal physics reasoning benchmark exploring variable, process, and solution dimensions, 2025. URL https://arxiv.org/abs/2505.15472.

Vitor Gaboardi dos Santos, Ibrahim Khadraoui, Ibrahim Farhat, Hamza Yous, Samy Teffahi, and Hakim Hacid. Alrm: Agentic llm for robotic manipulation, 2026. URL https://arxiv.org/abs/2601.19510.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, Jinlan Fu, Jingjing Gong, and Xipeng Qiu. Libero-plus: In-depth robustness analysis of vision-language-action models, 2025. URL https://arxiv.org/abs/2510.13626.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, Guanya Shi, Jiajun Wu, Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi "Jim" Fan. Cap-x: A framework for benchmarking and improving coding agents for robot manipulation, 2026. URL https://arxiv.org/abs/2603.22435.

Jiaheng Hu, Mohit Shridhar, Caden Lu, Dhruv Shah, Hao-Tien Lewis Chiang, Jie Tan, and Annie Xie. What matters in orchestrating robot policies: A systematic study of hierarchical vla agents, 2026. URL https://arxiv.org/abs/2606.10267.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning

through planning with language models, 2022. URL https://arxiv.org/abs/2207.05608.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models, 2023. URL https://arxiv.org/abs/2307.05973.

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

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model, 2024. URL https://arxiv.org/abs/2406.09246.

Jason Lee, Jiafei Duan, Haoquan Fang, Yuquan Deng, Shuo Liu, Boyang Li, Bohan Fang, Jieyu Zhang, Yi Ru Wang, Sangho Lee, Winson Han, Wilbert Pumacay, Angelica Wu, Rose Hendrix, Karen Farley, Eli VanderBilt, Ali Farhadi, Dieter Fox, and Ranjay Krishna. Molmoact: Action reasoning models that can reason in space, 2025. URL https://arxiv.org/abs/2508.07917.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, Mona Anvari, Minjune Hwang, Manasi Sharma, Arman Aydin, Dhruva Bansal, Samuel Hunter, Kyu-Young Kim, Alan Lou, Caleb R. Matthews, Ivan Villa-Renteria, Jerry Huayang Tang, Claire Tang, Fei Xia, Silvio Savarese, Hyowon Gweon, C. Karen Liu, Jiajun Wu, and Li Fei-Fei. BEHAVIOR-1K: A benchmark for embodied AI with 1, 000 everyday activities and realistic simulation. In Karen Liu, Dana Kulic, and Jeffrey Ichnowski (eds.), *Conference on Robot Learning, CoRL 2022, 14-18 December 2022, Auckland, New Zealand*, volume 205 of *Proceedings of Machine Learning Research*, pp. 80–93. PMLR, 2022. URL https://proceedings.mlr.press/v205/li23a.html.

Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li Erran Li, Ruohan Zhang, Weiyu Liu, Percy Liang, Li Fei-Fei, Jiayuan Mao, and Jiajun Wu. Embodied agent interface: Benchmarking llms for embodied decision making, 2025. URL https://arxiv.org/abs/2410.07166.

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, 2023. URL https://arxiv.org/abs/2209.07753.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning, 2023. URL https://arxiv.org/abs/2306.03310.

Yulin Luo, Chun-Kai Fan, Menghang Dong, Jiayu Shi, Xiangju Mi, Mengdi Zhao, Bo-Wen Zhang, Cheng Chi, Jiaming Liu, Gaole Dai, Rongyu Zhang, Ruichuan An, Kun Wu, Zhengping Che, Shaoxuan Xie, Guocai Yao, Zhongxia Zhao, Pengwei Wang, Guang Liu, Zhongyuan Wang, Tiejun Huang, and Shanghang Zhang. Robobench: A comprehensive evaluation benchmark for multimodal large language models as embodied brain, 2026. URL https://arxiv.org/abs/2510.17801.

Aengus Lynch, Benjamin Wright, Caleb Larson, Stuart J. Ritchie, Sören Mindermann, Evan Hubinger, Ethan Perez, and Kevin K. Troy. Agentic misalignment: How llms could be insider threats. *CoRR*, abs/2510.05179, 2025. doi: 10.48550/ARXIV.2510.05179. URL https://doi.org/10.48550/arXiv.2510.05179.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models, 2024. URL https://arxiv.org/abs/2310.12931.

Alexander Meinke, Bronson Schoen, Jérémy Scheurer, Mikita Balesni, Rusheb Shah, and Marius Hobbhahn. Frontier models are capable of in-context scheming. *CoRR*, abs/2412.04984, 2024. doi: 10.48550/ARXIV.2412.04984. URL https://doi.org/10.48550/arXiv.2412.04984.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. doi: 10.48550/ARXIV.2303.08774. URL https://doi.org/10.48550/arXiv.2303.08774.

Shi Qiu, Shaoyang Guo, Zhuo-Yang Song, Yunbo Sun, Zeyu Cai, Jiashen Wei, Tianyu Luo, Yixuan Yin, Haoxu Zhang, Yi Hu, Chenyang Wang, Chencheng Tang, Haoling Chang, Qi Liu, Ziheng Zhou, Tianyu Zhang, Jingtian Zhang, Zhangyi Liu, Minghao Li, Yuku Zhang, Boxuan Jing, Xianqi Yin, Yutong Ren, Zizhuo Fu, Jiaming Ji, Weike Wang, Xudong Tian, Anqi Lv, Laifu Man, Jianxiang Li, Feiyu Tao, Qihua Sun, Zhou Liang, Yushu Mu, Zhongxuan Li, Jing-Jun Zhang, Shutao Zhang, Xiaotian Li, Xingqi Xia, Jiawei Lin, Zheyu Shen, Jiahang Chen, Qiuhao Xiong, Binran Wang, Fengyuan Wang, Ziyang Ni, Bohan Zhang, Fan Cui, Changkun Shao, Qing-Hong Cao, Ming xing Luo, Yaodong Yang, Muhan Zhang, and Hua Xing Zhu. Phybench: Holistic evaluation of physical perception and reasoning in large language models, 2025. URL https://arxiv.org/abs/2504.16074.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Carmelo Sferrazza, Dun-Ming Huang, Xingyu Lin, Youngwoon Lee, and Pieter Abbeel. Humanoidbench: Simulated humanoid benchmark for whole-body locomotion and manipulation, 2024. URL https://arxiv.org/abs/2403.10506.

Lucy Xiaoyang Shi, Brian Ichter, Michael Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, Adrian Li-Bell, Danny Driess, Lachy Groom, Sergey Levine, and Chelsea Finn. Hi robot: Open-ended instruction following with hierarchical vision-language-action models, 2025. URL https://arxiv.org/abs/2502.19417.

Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots, Second Edition*. Intelligent robotics and autonomous agents. MIT Press, 2011. ISBN 978-0-262-01535-6.

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, 2022. URL https://arxiv.org/abs/2209.11302.

Yujin Tang, Wenhao Yu, Jie Tan, Heiga Zen, Aleksandra Faust, and Tatsuya Harada. Saytap: Language to quadrupedal locomotion, 2023. URL https://arxiv.org/abs/2306.07580.

Kimi Team, Yifan Bai, Yiping Bao, Y. Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Chenxiao Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Yuyao Ge, Shangyi Geng, Qizheng Gu, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Yunjia He, Chao Hong, Hao Hu, Yangyang Hu, Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yang Li, Yanhao Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Haoyu Lu, Lijun Lu, Yashuo Luo, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Zeyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Lin Sui, Xinjie Sun, Flood Sung, Yunpeng Tai, Heyi Tang, Jiawen Tao, Qifeng Teng, Chaoran Tian, Chensi Wang, Dinglu Wang, Feng Wang, Hailong Wang, Haiming Wang, Jianzhou Wang,

Jiaxing Wang, Jinhong Wang, Shengjie Wang, Shuyi Wang, Si Wang, Xinyuan Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang, Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Haoning Wu, Wenhao Wu, Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Jin Xie, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Jing Xu, Jing Xu, Junjie Yan, Yuzi Yan, Hao Yang, Xiaofei Yang, Yi Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Xingcheng Yao, Wenjie Ye, Zhuorui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang Yuan, Mengjie Yuan, Siyu Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang, Yadong Zhang, Yangkun Zhang, Yichi Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Haotian Zhao, Yikai Zhao, Zijia Zhao, Huabin Zheng, Shaojie Zheng, Longguang Zhong, Jianren Zhou, Xinyu Zhou, Zaida Zhou, Jinguo Zhu, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. Kimi k2: Open agentic intelligence, 2026. URL https://arxiv.org/abs/2507.20534.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109. URL https://doi.org/10.1109/IROS.2012.6386109.

Brian Y. Tsui, Alan Y. Fang, and Tiffany J. Hwu. Demonstration-free robotic control via llm agents, 2026. URL https://arxiv.org/abs/2601.20334.

Yen-Jen Wang, Bike Zhang, Jianyu Chen, and Koushil Sreenath. Prompt a robot to walk with large language models, 2024. URL https://arxiv.org/abs/2309.09969.

Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. Tidybot: Personalized robot assistance with large language models, 2023. URL https://arxiv.org/abs/2305.05658.

Yilin Wu, Ran Tian, Gokul Swamy, and Andrea Bajcsy. From foresight to forethought: Vlm-in-the-loop policy steering via latent alignment, 2025. URL https://arxiv.org/abs/2502.01828.

Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Reward shaping with language models for reinforcement learning, 2024. URL https://arxiv.org/abs/2309.11489.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.

Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, Heng Ji, Huan Zhang, and Tong Zhang. Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents, 2025b. URL https://arxiv.org/abs/2502.09560.

Yifan Yang, Zhixiang Duan, Tianshi Xie, Fuyu Cao, Pinxi Shen, Peili Song, Piaopiao Jin, Guokang Sun, Shaoqing Xu, Yangwei You, and Jingtai Liu. Fpc-vla: A vision-language-action framework with a supervisor for failure prediction and correction, 2025c. URL https://arxiv.org/abs/2509.04018.

Shiduo Zhang, Zhe Xu, Peiju Liu, Xiaopeng Yu, Yuan Li, Qinghui Gao, Zhaoye Fei, Zhangyue Yin, Zuxuan Wu, Yu-Gang Jiang, and Xipeng Qiu. Vlabench: A large-scale benchmark for language-conditioned robotics manipulation with long-horizon reasoning tasks, 2024. URL https://arxiv.org/abs/2412.18194.

Enyu Zhao, Vedant Raval, Hejia Zhang, Jiageng Mao, Zeyu Shangguan, Stefanos Nikolaidis, Yue Wang, and Daniel Seita. Manipbench: Benchmarking vision-language models for low-level robot manipulation, 2025. URL https://arxiv.org/abs/2505.09698.

Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. Libero-pro: Towards robust and fair evaluation of vision-language-action models beyond memorization, 2026. URL https://arxiv.org/abs/2510.03827.

Appendix Contents

In Part I we specify the tasks, prompts, and model settings. Part II follows the order of the paper's experiments through Classic Control, low-level locomotion, high-level locomotion, and then manipulation. It includes additional results for the open models we tested, Kimi and Qwen. Part III contains manipulation input-aid, learning-from-experience, and reasoning analyses. In Part IV, we cover the limitations of our methods.

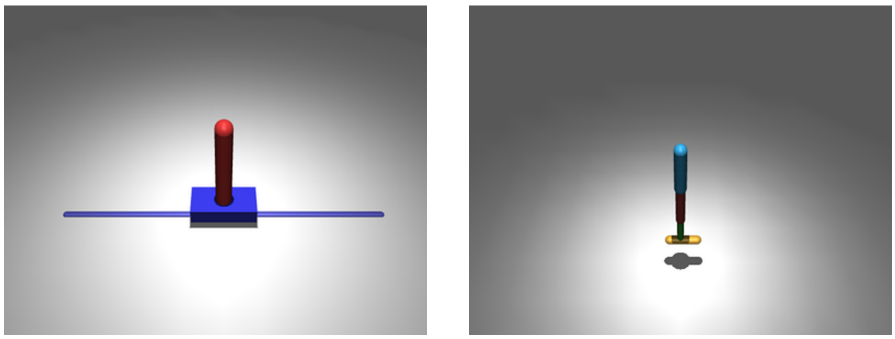
Part I. Methods & Reference

A Task, Robot, and Environment Specifications

A.1 Task and Evaluation Settings

We run all experiments in MuJoCo with a 0.002 s physics step and pause the simulator during model inference. Depending on the interface, a model turn can return several actions, launch a controller, or invoke a pretrained policy. As stated before, we pause the simulation between model calls. The direct-control results therefore should be read as an upper bound on current real-world performance, and how a model with equivalent intelligence but nearly instantaneous latency would behave.

Classic control. On the Inverted Pendulum, the force the model supplies is applied for 25 physics steps (0.05 s, or 20 Hz). Hopper applies each three-joint torque vector for 4 steps (0.008 s, or 125 Hz), and Twin-Flipper applies each two-flipper torque vector for 10 steps (0.02 s, or 50 Hz). Within a task, Direct, Code, and deployed Training-supervision controllers use the same action cadence. Pendulum is scored by balance time in an episode of up to 20 s, Hopper by peak forward speed in episodes of up to 20 s, and Twin-Flipper by total ball airtime in its best episode of up to 90 s. We sum the time the ball spends airborne across all bounces in an episode.



(a) Inverted Pendulum (1-DOF) (b) Hopper (3-DOF)

Figure 8: The two standard MuJoCo environments used alongside Twin-Flipper (Fig. 1): an inverted pendulum and Hopper.

Low-level locomotion. The Go2 and G1 low-level tasks apply 12- and 29-dimensional joint-torque vectors, respectively, for 6 physics steps (0.012 s, or 83.3 Hz). Models receive a third-person image,

joint state, base orientation and height, acceleration, and simulation time. In stand-up trials, Go2 begins prone and G1 on its side, and both must remain upright for 2 s for the model to attain success. In balance trials, the robot begins upright and we measure the longest upright streak that the model maintains. The Go2 walking task measures its longest forward distance and whether it reaches 5 m without falling.

High-level locomotion. For the eleven high-level Go2 tasks, a pretrained walking policy handles balance and joint actuation while the model chooses named movement commands. Models normally receive forward-facing RGB; the input-aid experiments change this input. Appendix E lists the standard commands and describes the tasks.

Manipulation. We run the manipulation experiments in the robosuite-based LIBERO environment, whose low-level arm controller updates at 20 Hz. A Direct command specifies three Cartesian position changes, three rotation changes, and a gripper command, then advances the controller by 0.1 s. Under Policy supervision, MolmoAct instead supplies eight actions at the same controller rate. Models receive scene and wrist images together with the gripper state.

LIBERO-40 contains 40 tasks from the goal, long-horizon, object, and spatial suites. We score each trial by final task success. Supporting appendix analyses also show whether the robot touches or grasps the target object before the trial ends.

Uncertainty. In the main paper, we show uncertainty as ± 1 SE. Figure 18 also uses ± 1 SE, with $n = 36$ for novel-task success. Manipulation-reasoning rates use 95% Wilson confidence intervals at $n = 15$. The high-level locomotion overview computes uncertainty from variation among trials whereas Fig. 11 uses variation among tasks in each group.

Trial limits. Most tasks end after a fixed number of model decisions. Training supervision is the only exception: we limit these trials by time because model-written training procedures can run for different lengths.

Episode resets. After a reset, a new episode starts in the same trial, but the model retains the old turns in its context. Under Direct and Code control, success instantly ends the trial. After an unsuccessful episode, we reset Pendulum, Hopper, Twin-Flipper, Go2 balance, and G1 balance under both Direct and Code control. For Go2 and G1 stand-up, Direct ends the trial while Code resets the robot and restarts the episode. We test Go2 5 m walk only under Code control and reset it after an unsuccessful episode. The Direct and Code stand-up conditions therefore permit different numbers of attempts.

A.2 Training Supervision

For the Training supervision experiments, we provide a proximal policy optimization (PPO) trainer (Schulman et al., 2017) implemented in PyTorch, with Adam updates and parallel simulation through MuJoCo Warp, to the VLM in each trial. We then ask the VLM to write the policy and value networks and the code that configures and calls the trainer. We leave the network depth, width, and activations up to the VLM, along with the learning rate, rollout length, minibatch size, and number of update epochs, though we do provide limits on these values.

We run policy training on NVIDIA H200 GPUs and set a session time limit of 90 minutes for the three classic-control tasks and four hours for G1 and Go2 stand-up. We let the VLM choose the number of environment steps in each training call, so trials do not share a fixed step budget. We give the VLM access to the training simulator's state and let it set rewards, starting states, and episode conclusion conditions. It can change these settings between training runs to build a curriculum, and save intermediate checkpoints. We return training rewards, episode lengths, and policy and value losses so that it can inspect the results and revise its code.

We evaluate a policy in a separate environment with the task's specified initial conditions, action cadence, and scoring rules. We give the policy positions and velocities from the simulator and run it without further VLM decisions during the rollout. The VLM can inspect the returned score and state trace before revising the policy or training it further. We score these deployments using the task's standard evaluation rules. We provide full details in our code.

B SYSTEM PROMPTS

We gave every model the same system prompt for a given task and input condition; we did not tune prompts by model. The released code contains every template and expanded prompt. Here we summarize what changed across experiments and include two representative templates.

Direct Control. Pendulum used a prompt describing its setup and the goal of keeping the pendulum upright. Hopper and Twin-Flipper shared a torque-control template filled with the task's robot description, goal, observations, timing, and output format. G1 and Go2 shared a locomotion template filled with the appropriate robot and initial state. The balance version began upright, replaced the stand-up goal with a balance goal, and added reset instructions. LIBERO used an arm-control prompt containing the task instructions and the images that would be available in that condition.

Code Control. In general, the prompts under the code setting were similar to Direct Control but the model was asked to write a Python controller that called `step_action`. Go2 5 m walk was evaluated only under Code Control.

Training Supervision. Every run used the same training template. The task goal, robot, observations, actions, episode settings, and success rule were filled from the task definition. Appendix A.2 describes the trainer and the choices we leave to the VLM.

Policy Supervision. Each high-level Go2 task supplied its own goal and success rule but used the same pretrained walking policy and the commands listed in Appendix E. Input-aid conditions changed only the information available to the model (Appendix E). LIBERO used one MolmoAct-supervision template; only the task goal and available images changed across conditions.

Representative Direct-Control template. Hopper and Twin-Flipper use this abridged template. The braced fields vary by task; optional blocks are given to the model only when they are applicable.

```
You are directly controlling a {robot} robot via torque commands.

## Robot
{description}

## Joints ({n} actuators)
{joint_names}
Each torque value: float in [{minimum}, {maximum}]

## Goal
{goal}

## Strategy
{strategy}

## Physics
- Each torque vector is applied for {step_time} seconds.
- Observations: {observation_format}

## Output Format
Output one torque value per joint:
 {"action": "direct_control", "forces": [{torques}]}

{optional_reset_and_vision_blocks}
```

Representative Policy-Supervision template. LIBERO uses the following abridged base prompt. The task goal and image block change with the task and input condition.

```
Your job is to supervise a Vision-Language-Action (VLA) model
controlling a robotic arm. You will look over its actions and use
them to perform the task.
```

```
YOUR TASK: {goal}

1. Set the VLA instruction:
 {"action": "set_vla_instruction", "instruction": "pick up the bowl"}

2. Ask the VLA to propose actions without advancing physics:
 {"action": "get_vla_action"}

3. Execute the proposed actions, edited actions, or your own actions:
 {"action": "execute_action", "actions": [[dx, dy, dz, drx,
 dry, drz, gripper], ...]}

4. Declare completion when the task looks complete:
 {"action": "finish"}

The VLA proposes actions, but you decide what to execute. You may use,
edit, or replace its proposal. Observe the result and repeat until the
task is complete.

{vision_block}
```

Reset instructions. For many trials, when the model reaches a failure condition it restarts another episode. We consider this the same trial, as the model can see its previous attempts and theoretically learn from them. The Direct prompts for Pendulum, Hopper, Twin-Flipper, Go2 balance, and G1 balance inform the model that an unsuccessful episode resets automatically and returns a new state. The Direct stand-up prompts for Go2 and G1 contain no reset instruction because the trial ends and does not restart. The Code prompts for Pendulum, Hopper, Twin-Flipper, Go2 balance, and G1 balance also explain the reset. The initial Code prompts for Go2 and G1 stand-up and Go2 5 m walk do not mention resets, but after an unsuccessful episode the system resets the environment and tells the model that it has received a new state. A completion declaration counts as success only when the environment's success condition is also met. Manipulation tasks do not reset.

C Data Availability

We release the evaluation code complete with the full expanded prompts.

For all of our results, we run each experimental condition multiple times to see how the model performs and how its score varies. Each trial begins a new interaction with the model. Its responses can differ even for the same prompt and starting state, so trials can produce different actions, controllers, or training programs.

Under Direct and Code control, Pendulum, Hopper, and low-level Go2 and G1 trials start from the same state for each task. Twin-Flipper slightly varies the ball's horizontal starting position in each episode. During policy training, we let the model choose its starting states. Manipulation trials vary the task and environment seed.

C.1 Models and API Settings

Table 3 lists each model's API identifier and settings.

C.2 Asset Licenses

The robot models come from MuJoCo Menagerie (Unitree Go2 and G1, Franka Panda, and xArm7) and run in MuJoCo. The tabletop tasks come from LIBERO and its robosuite environment, and we use the publicly available checkpoint of MolmoAct. We access closed models through the APIs (Anthropic, OpenAI, Google, Moonshot, and Alibaba via OpenRouter). Every asset is used under its published license.

Table 3: Model endpoints and settings. "Reasoning" is the vendor's effort setting; dates are trial-launch dates. For sampling, we used defaults except Opus 4 with reasoning off, which used temperature 0.2.

Display name	Vendor	Snapshot / endpoint ID	Date range	Reasoning
Opus 4	Anthropic	`claude-opus-4-20250514`	2026-04-20 / 04-21	high
Opus 4.1	Anthropic	`claude-opus-4-1-20250805`	2026-04-16 / 04-21	high (16k budget)
Opus 4.5	Anthropic	`claude-opus-4-5-20251101`	2026-04-15 / 04-20	high (16k budget)
Opus 4.6	Anthropic	`claude-opus-4-6`	2026-04-14 / 04-20	high (adaptive)
Opus 4.7	Anthropic	`claude-opus-4-7`	2026-04-21 / 04-29	high (adaptive)
Mythos Preview	Anthropic	`mythos`	2026-04-15 / 04-22	high (adaptive)
GPT-5.1	OpenAI	`openai/gpt-5.1`	2026-04-21 / 04-26	high
GPT-5.4	OpenAI	`openai/gpt-5.4`	2026-04-14 / 04-22	none†
Gemini 2.5	Google	`google/gemini-2.5-pro`	2026-04-21 / 04-23	high
Gemini 3.1	Google	`google/gemini-3.1-pro-preview`	2026-04-14 / 04-20	low†
Kimi K2.6	Moonshot AI	`moonshotai/kimi-k2.6`	2026-04-21 / 04-25	high
Qwen 3.6+	Alibaba	`qwen/qwen3.6-plus`	2026-04-21 / 04-24	high (mandatory)

Reasoning notation. `high (adaptive)` uses Anthropic's adaptive-thinking effort setting; `high (16k budget)` uses the now-deprecated token budget option; `high (mandatory)` uses OpenRouter's required nonzero effort for Qwen; and † marks a model whose API has no `high` tier, and so shows the strongest available setting.

Part II. Detailed Results

The sections follow the main paper: Classic Control, low-level locomotion, high-level locomotion, and manipulation.

D Classic Control and Low-Level Locomotion

D.1 Full Classic-Control Results

Figure 10 reports Training-supervision results.

D.2 Low-Level Locomotion Details

Figure 3 reports balance duration and stand-up final success. Table 4 gives stand-up results, including the G1 results which were universally zero. Table 5 gives the other results for Qwen and Kimi.

Table 4: **Exact stand-up success rates for the eight matched models.** Rates are percentages. The final column gives G1 Direct/Code/Training; every value is zero.

	Go2 stand up			G1 stand up
Model	Direct	Code	Training	Direct/Code/Training
Opus 4	0.0	0.0	0.0	0/0/0
Opus 4.1	0.0	5.7	0.0	0/0/0
Opus 4.5	0.0	91.4	2.9	0/0/0
Opus 4.6	0.0	77.1	20.0	0/0/0
Opus 4.7	0.0	94.0	22.0	0/0/0
Mythos Preview	2.0	98.0	66.0	0/0/0
GPT-5.4	0.0	48.6	74.3	0/0/0
Gemini 3.1	0.0	68.6	2.9	0/0/0

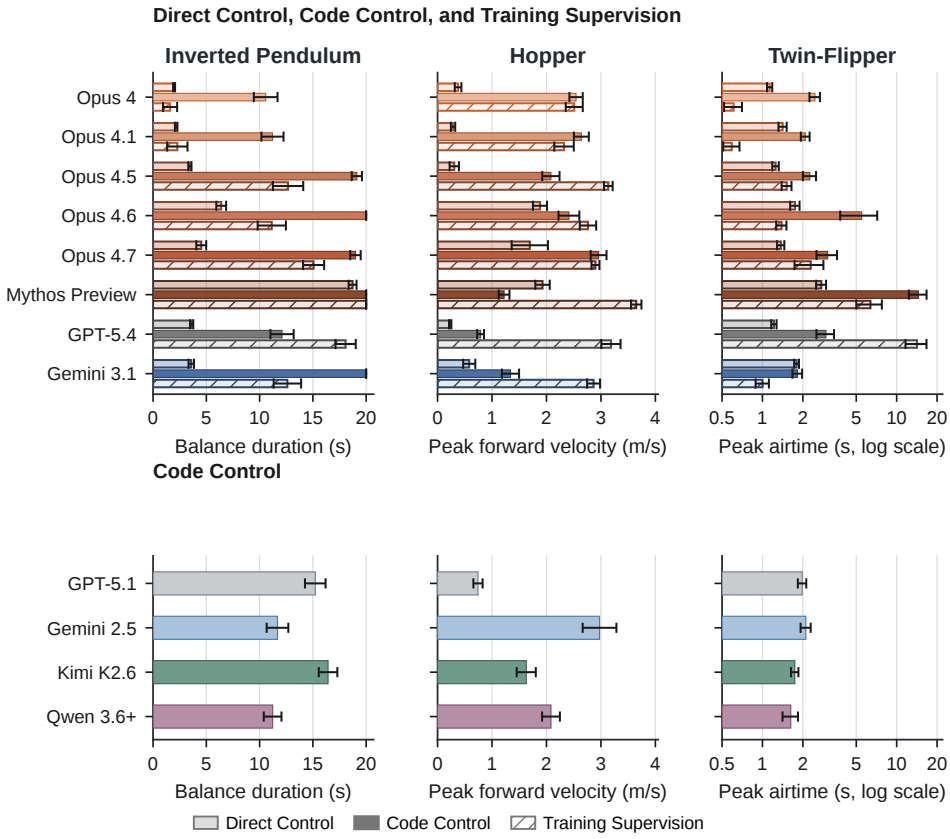


Figure 9: **Code and Training supervision usually outperform Direct control, but neither is consistently stronger across tasks.** Dashes mark interfaces that were not evaluated.

Table 5: **Low-level locomotion results for Kimi and Qwen.** We report Code-control results for G1 balance, Go2 stand up, and Go2 5 m walk, together with Direct- and Code-control results for Go2 balance. Values are mean ± 1 SE.

Model	G1 balance Code (s)	Go2 stand Code progress / success (%)	Go2 balance Direct / Code (s)	Go2 5 m walk Code success (%)
Kimi K2.6	1.384 ± 0.064	81.20 ± 4.40 / 60.00	0.365 ± 0.030 / 1.954 ± 0.041	45.71
Qwen 3.6+	0.603 ± 0.047	29.88 ± 4.04 / 8.57	0.308 ± 0.021 / 1.384 ± 0.132	17.14

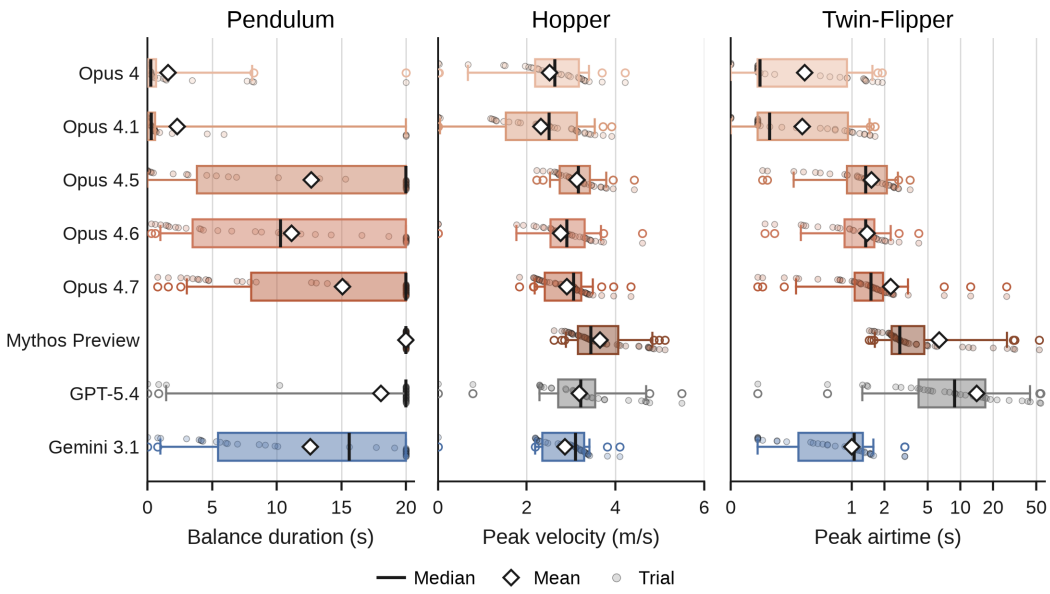


Figure 10: **Training supervision produces task-dependent controller quality.** The panels report Pendulum balance duration, Hopper peak forward velocity, and Twin-Flipper total ball airtime in the best episode. Pendulum duration is capped at 20 s.

E HIGH-LEVEL LOCOMOTION AND INPUT AIDS

In our policy supervision locomotion experiments, the models send commands to a pretrained Go2 walking policy (§4.2.1). Most tasks provide eight commands: stand up, stand down, move forward, move backward, turn left, turn right, stop, and recover to a standing pose. Forward and backward commands set a speed in meters per second and a duration in seconds; turns set an angular speed in radians per second and a duration. The eleven tasks are listed below.

We group five navigation tasks under Perception. They ask the model to locate a target (*find_x*), distinguish a target from distractors (*visual_search*), cross occluded corridors (*procedural_maze*), avoid physical obstacles (*obstacle_course*), and infer invisible collision walls by trial and error (*invisible_walls*).

The Memory tasks ask the model to reproduce a colour sequence (*color_sequence*) and answer layout questions after free exploration (*explore_report*). In the Orientation group, return home (*return_home*) asks the model to return to an unmarked start pose using only its remembered route, while the drift-detection and turn-correction tasks (*drift_detection*, *turn_correction*) introduce control errors that the model must recognize and correct from the camera view.

We test planning with the one-shot course (*oneshot_course*), which requires the model to complete an entire trajectory from a single overhead image, with no camera or feedback once the run starts.

Input aids. The six conditions that we test are an RGB image only, a center crosshair, a depth overlay, the robot's heading as text, a third-person camera, and all aids combined.

For the RGB-only results in the left panel of Figure 5, we use a 20,000-token reasoning budget for Opus 4, Opus 4.5, and Opus 4.6. Opus 4.1 does not use extended reasoning; Opus 4.7 and Mythos Preview use maximum adaptive reasoning.

The overview score is the unweighted mean of the eleven task means, multiplied by 100. For input-aid comparisons, *oneshot_course*, which has no camera, retains its RGB-only value. The standard error from trial variation is $\mathrm{SE} = \frac{100}{11}\sqrt{\sum_i \mathrm{var}_i/n_i}$, where var_i is the variance of the unscaled trial scores for task i and n_i is its trial count.

In general, we find that the compass helps the Orientation group the most. Other individual aids change the group averages little, and dense overlays do not improve any group (Fig. 11).

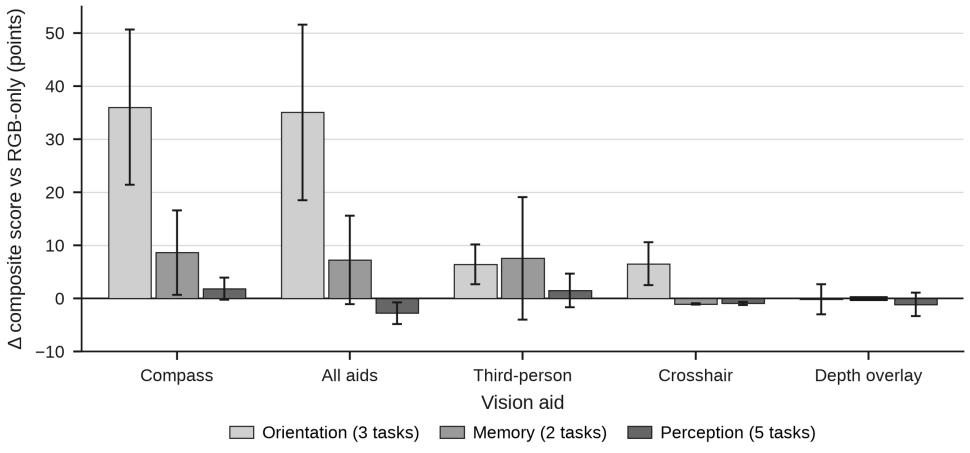


Figure 11: **Aid effects depend on task type.** We compare the average change from RGB-only input within the Perception, Memory, and Orientation groups described above. Together, these groups contain the ten Go2 tasks that use a camera.

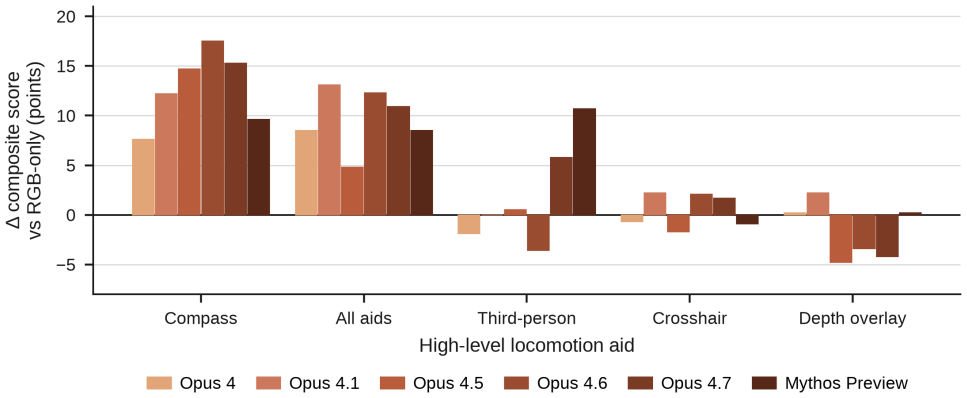


Figure 12: **The value of additional information changes across model generations.** The figure compares Claude checkpoints on the high-level Go2 Policy-supervision tasks. Changes are relative to RGB-only input.

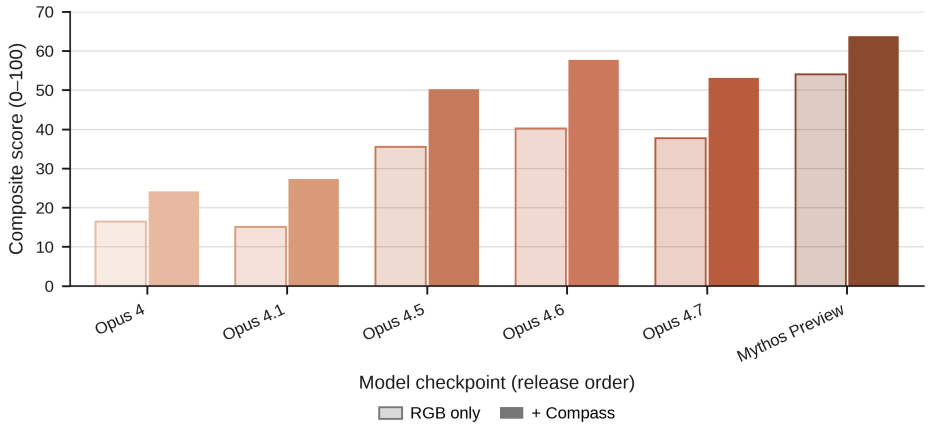


Figure 13: **Compass improves high-level Go2 locomotion across checkpoints.** Each Claude checkpoint is evaluated on the same eleven Policy-supervision tasks with RGB-only input and with the compass added.

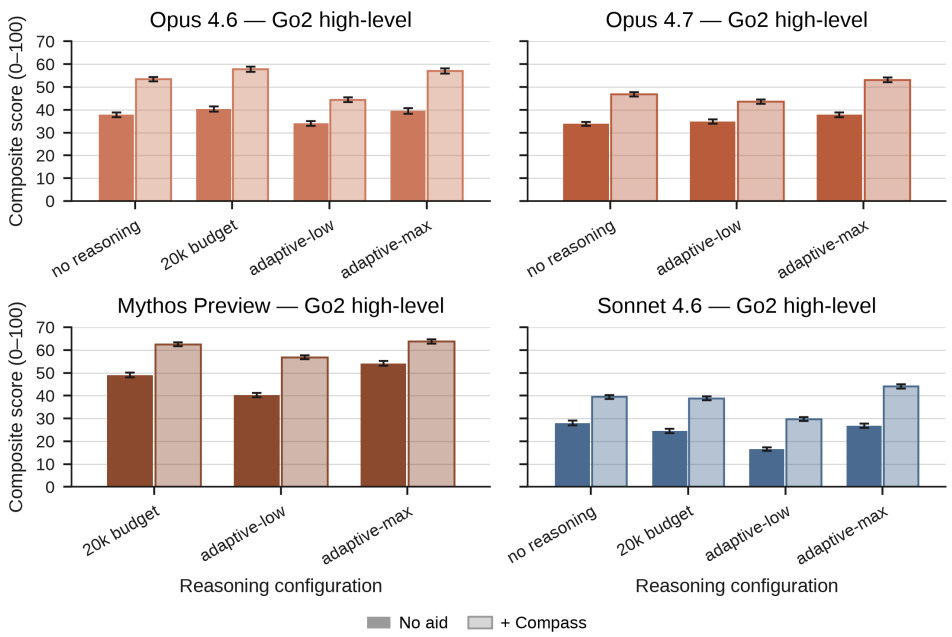


Figure 14: **The compass gain persists across reasoning settings.** Each panel shows the performance of reasoning configurations for one Claude model on the eleven high-level Go2 Policy-supervision tasks, with and without the compass.

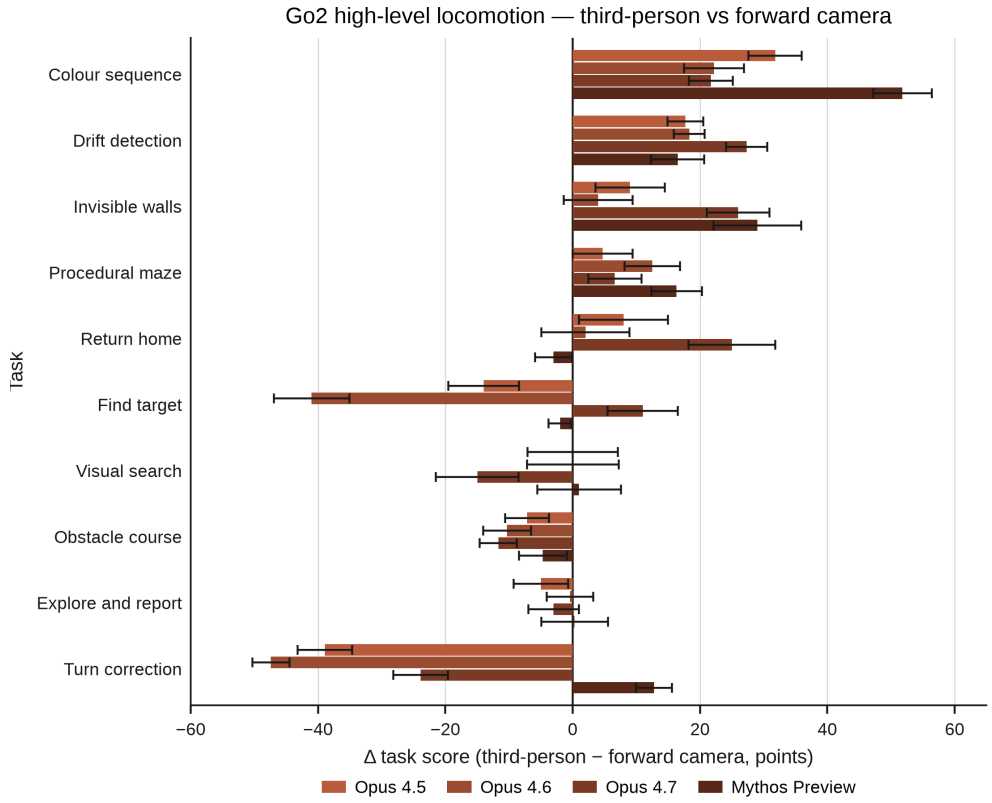


Figure 15: **A third-person camera helps some tasks but hurts others.** The change is relative to the setting where the model receives only a first-person RGB image of the scene.

E.1 RGB-Only Leaderboard and Plateau

Figure 16 reports all 26 tested settings. The labels *nr*, *20k*, *adlo*, and *admax* denote no extra reasoning, a 20,000-token budget, adaptive-low, and adaptive-max.

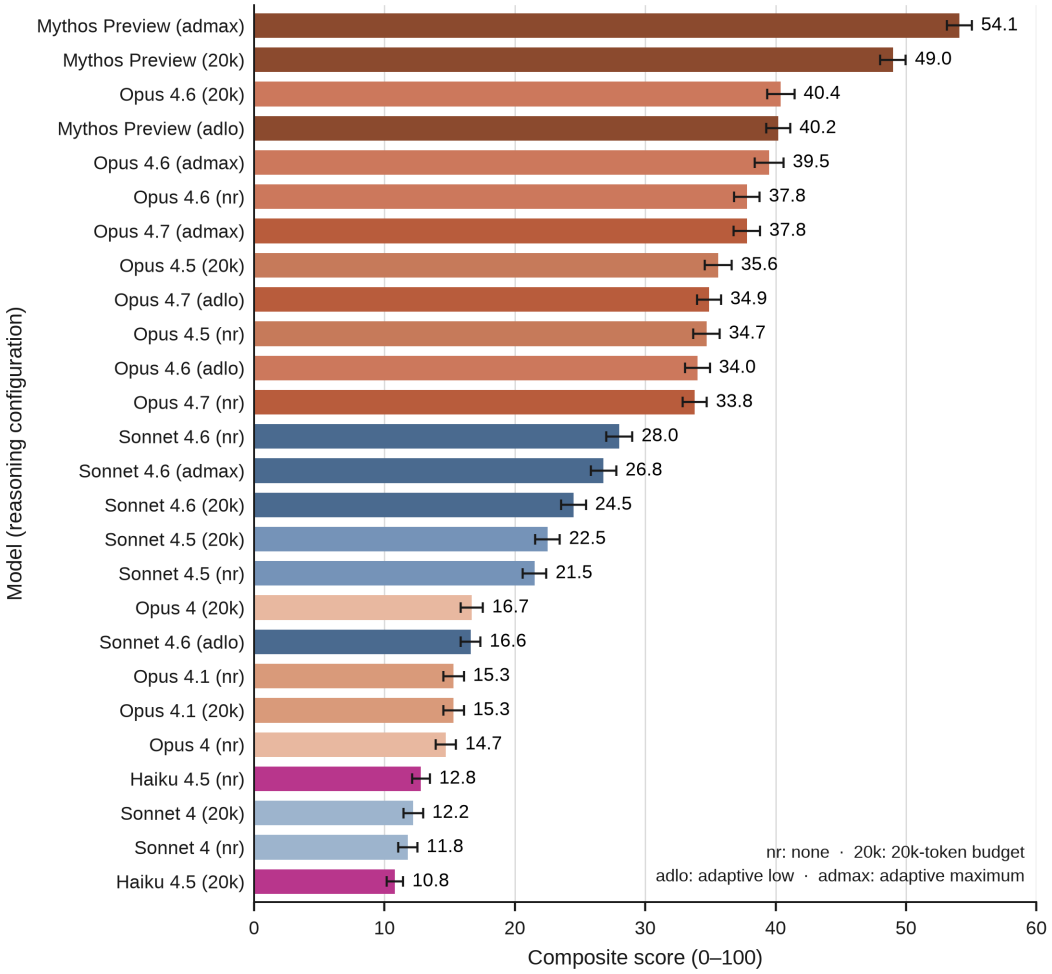


Figure 16: **Performance plateaus slightly across later Claude checkpoints before Mythos Preview improves.** The figure shows the reasoning configurations on the high-level Go2 Policy-supervision tasks using the forward RGB image.

F Manipulation: Direct and Policy-Supervised Results

Under Policy supervision, the model can either invoke MolmoAct or act directly. It can also invoke MolmoAct and edit or disregard its suggestion.

On LIBERO-40, Kimi K2.6 reaches 30.5% final success and uses MolmoAct for 56.4% of executed steps, while Qwen 3.6+ reaches 5.5% final success and uses MolmoAct for 23.5% of steps.

The three novel tasks we use ask the robot to put a black bowl on a wine rack, a white bowl on a cabinet shelf, and a moka pot on a stove. The scenes and objects are taken from other splits of the LIBERO dataset that the VLA is not optimized for. When asked to complete these tasks, MolmoAct alone cannot succeed on any of them.

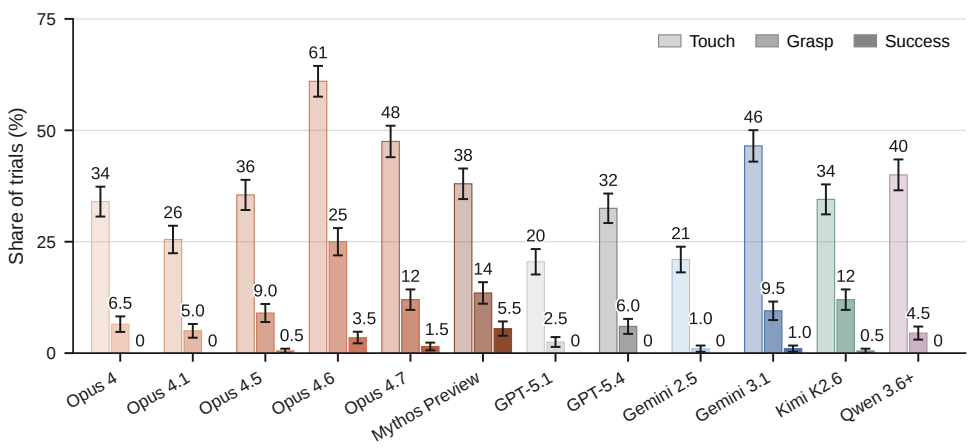


Figure 17: **Direct manipulation improves most clearly before task completion.** Touch, grasp, and final success rates are reported on the same 40 LIBERO tasks.

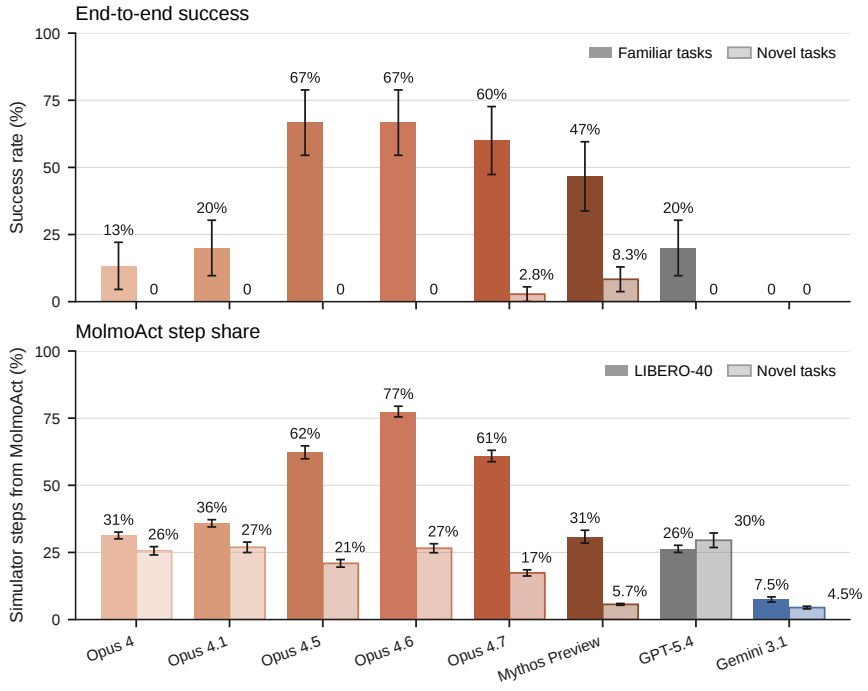


Figure 18: **Success and MolmoAct step share are lower for most models on novel tasks.** All panels show Policy-supervision trials. The top compares success on three familiar and three novel tasks; the bottom compares MolmoAct's share of simulator steps on LIBERO-40 and the three novel tasks.

Part III. Additional Analyses

G Manipulation Vision Aids

Our baseline experiments give RGB scene and wrist images to the model at each step. We also test additional tools, including a scene depth map, a scene segmentation mask, and a wrist-image point query that returns the object name, depth, and 3D position at the selected location (Fig. 19).

Table 6: **Final success rates (%) for the manipulation input comparison.** Every entry aggregates the same ten LIBERO tasks; dashes denote conditions that were not run.

Model	RGB	+Depth	+Segmentation	+3D point query
Opus 4	0	0	0	0
Opus 4.1	0	0	0	0
Opus 4.5	2	2	0	14
Opus 4.6	4	4	2	18
Opus 4.7	2	0	2	10
Mythos Preview	6	6	2	32
GPT-5.1	0	—	—	—
GPT-5.4	0	0	0	6
Gemini 2.5	0	—	—	—
Gemini 3.1	2	0	0	0
Kimi K2.6	0	—	—	—
Qwen 3.6+	0	—	—	—

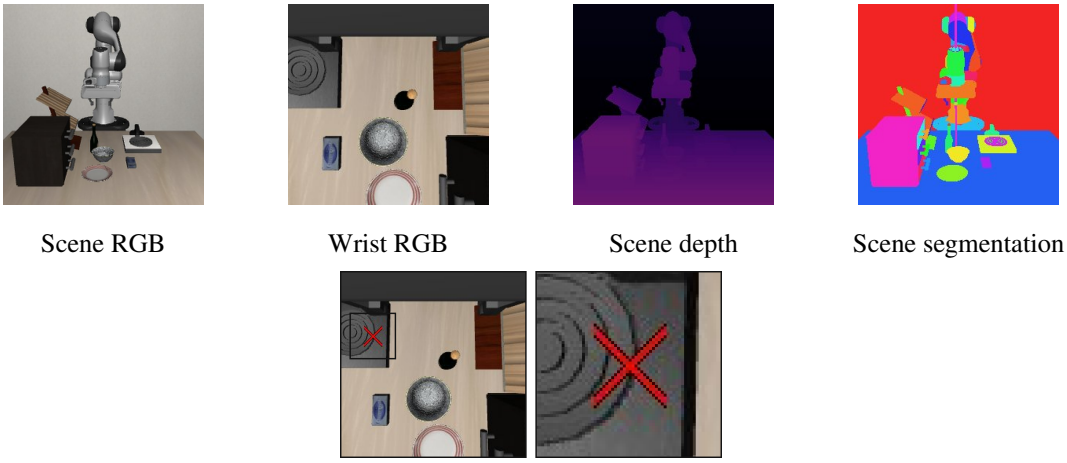


Figure 19: **Depth and segmentation cover the scene; the 3D point query uses the wrist view.** These examples show the tested manipulation inputs. A point query returns the object name, depth, and 3D position.

H Learning from Experience

Although LLMs are known to be few-shot learners (Brown et al., 2020), we now ask whether models can use earlier attempts and observations to improve during live rollouts. We test this in three ways: truncating the history of rollouts, allowing practice on the same route before evaluation, and letting a model revise its approach after failed attempts. The results show that recent experience can help on the same task in limited circumstances.

H.1 Removing Part of the Manipulation Conversation

We compare three separately run LIBERO-40 settings: the full conversation, the first 10 and final 12 messages, and the first 10 and final 6 messages. Opus 4.6's success rate decreases the most in both shorter settings, while a few models with lower full-conversation scores have slightly higher scores, indicating that the additional history did not help them. Overall, a shorter interaction history does not appear to be a significant impediment for most models (Fig. 20).

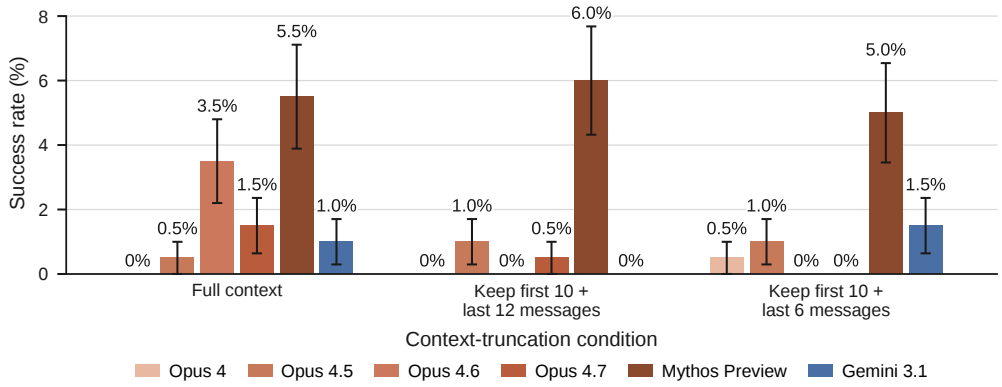


Figure 20: **Removing the middle of the conversation most clearly hurts Opus 4.6.** The figure compares LIBERO-40 Direct Control success with full conversation history and with the two shorter histories.

H.2 Practice on a Repeated Route

The one-shot course is one of the high-level Go2 tasks in which the model works with a pretrained walking policy. On this task, the model sees an overhead map and submits its full sequence of movement and turning commands before the robot begins moving. It receives no camera feedback while the robot follows the route, so for a one-shot attempt it cannot correct a mistake after the attempt begins, unlike other tasks which allow for self-correction. To test whether experience helps, we allow zero to five practice attempts on the same hallway before the scored attempt. Practice greatly improves success on this repeated route (Fig. 21).

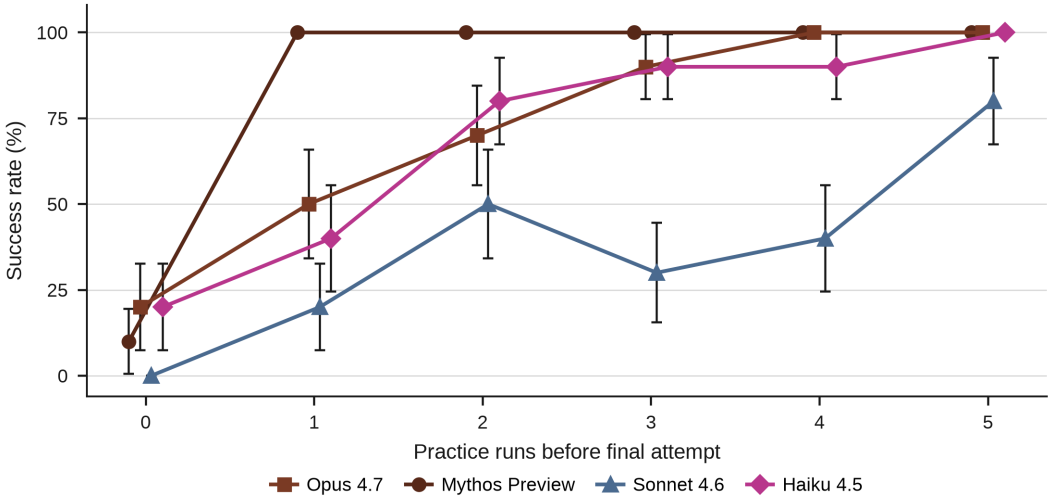


Figure 21: **Practice improves success on the same L-shaped hallway.** Each point shows success on the final attempt after zero to five practice runs on that route. The model submits the full command sequence before the robot moves and cannot change it after the attempt begins.

H.3 Improvement Across Retries

Across successive Opus versions, the best score across retries improves more than the score on the first attempt (Fig. 22).

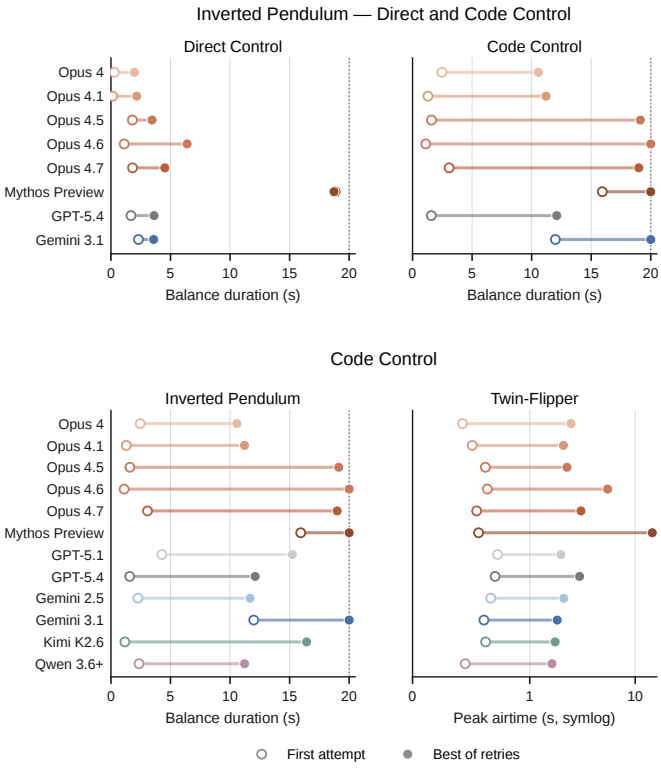


Figure 22: **Later Opus versions improve more after retries, while first-attempt scores change less.** The figure compares Direct and Code Control on Inverted Pendulum and Code Control on Twin-Flipper.

I Extended Reasoning

I.1 Effect of Extended Reasoning on Classic Control and Locomotion

With extended reasoning, most models have lower Hopper peak forward velocity; Opus 4.1 and GPT-5.4 gain less than 0.2 m/s. Pendulum balance duration changes little (Fig. 23). On the two Go2 tasks, only GPT-5.4's success rate changes substantially (Fig. 24). Based on this, we do not think reasoning is a substantial bottleneck.

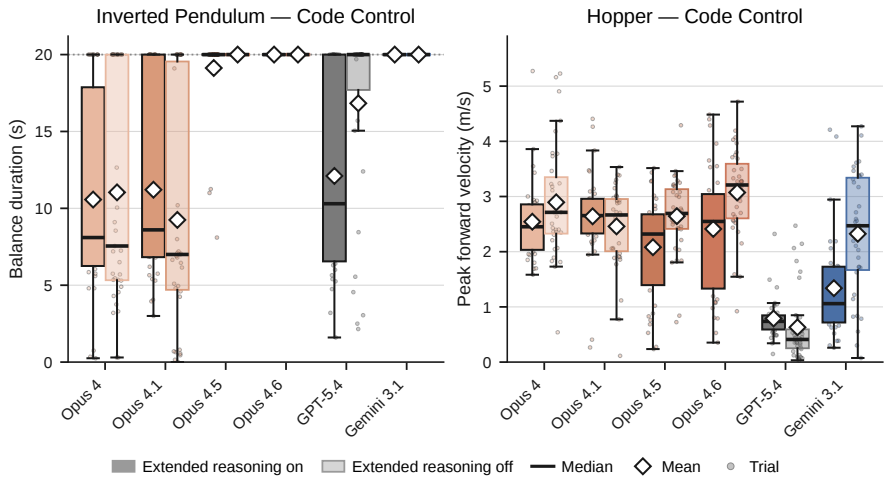


Figure 23: **Extended reasoning does not consistently improve classic-control code.** The panels report Pendulum balance duration and Hopper peak forward velocity with and without extended reasoning.

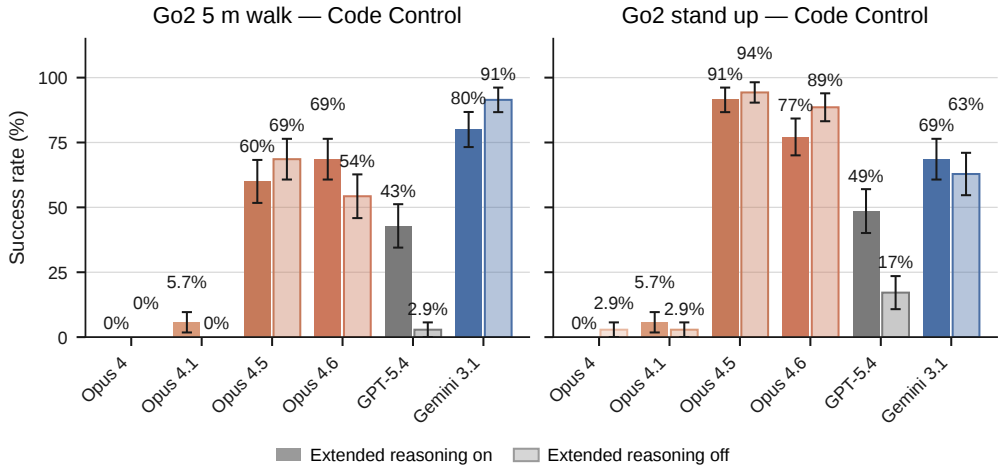


Figure 24: **Extended reasoning changes Go2 success for only a few model–task pairs.** The panels report Go2 5 m walk and Go2 stand up under Code control.

I.2 Effect of Extended Reasoning on Manipulation

Under Direct Control, extended reasoning changes touch and grasp rates unevenly and produces little change in final success (Fig. 25).

Under policy supervision, increased reasoning improves GPT-5.4's touch and grasp rates, whereas Opus 4.1 and Gemini 3.1 grasp less. Opus 4.5 and Opus 4.6 remain largely unchanged in final success (Fig. 26).

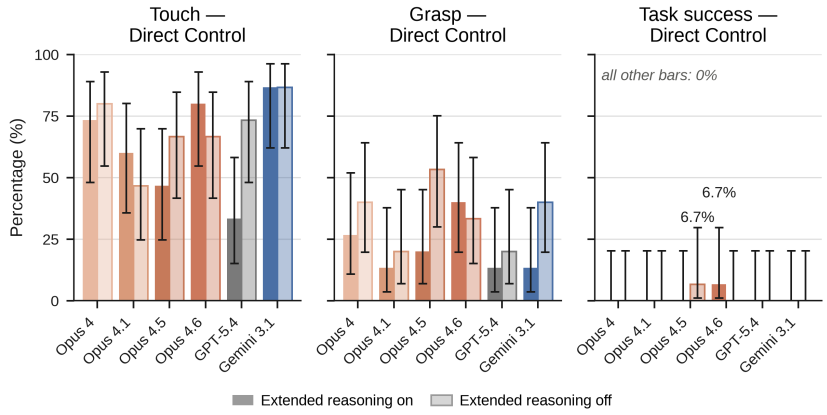


Figure 25: **Extended reasoning usually fails to improve direct manipulation.** Touch, grasp, and final success are compared on the same three familiar LIBERO tasks.

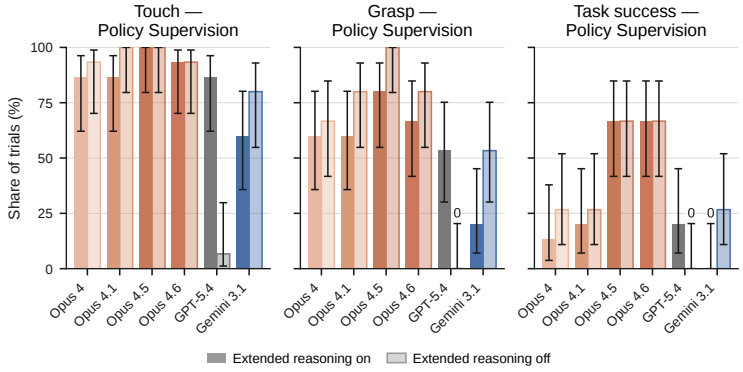


Figure 26: **Extended reasoning has mixed effects under policy supervision.** Touch, grasp, and success rates on three familiar LIBERO tasks are compared with and without extended reasoning.

Part IV. Discussion

J Limitations

Our results describe performance on the tasks, embodiments, and interfaces we evaluate. Because task difficulty and evaluation settings differ across embodiments, our results do not establish which kinds of robots VLMs control best. The frontier also depends on the prompts, observations, tools, pretrained policies, and budgets provided. However, we use the same prompt for each task and input condition without tuning it to individual models; other prompts or resources may produce different results. The models may have encountered public benchmarks, simulator code, or reference controllers during pretraining, which would indicate that these results may not generalize to novel embodiments. Reset opportunities also differ in some comparisons. For example, Code can retry stand-up within a trial, whereas Direct cannot.

All experiments run in simulation, and the simulator pauses during model inference. We therefore do not measure whether models can respond quickly enough to events around a physical robot. Differences in sensing, contact dynamics, and actuation may also change performance on real-world hardware. Additionally, task scores do not measure the frequency or severity of harmful actions, only ability to perform the tasks we test. The reproduction of our results may be impacted by changing access to proprietary model versions as they become deprecated.