arXiv:submit/8341354 [cs.AI] 6 Nov 2026

Embodied First Exam

Kangrui Wang[1*§], Qineng Wang[1*§], Changhe Chen[1*§], Pingyue Zhang[1*§], Chi Gui[1*§], Feishi Wang[1*§], Canyu Chen[1*§], Wenli Xiao[2§], Xiaoyu Huang[3§], Tonghe Zhang[2§], Linjie Li[4§], Zihan Wang[1], Minyeong Kim[5], Yiyou Sun[3], Mingkai Deng[2], Jinyu Hou[2], Qi Zhu[1], Han Liu[1], Eric P. Xing[2], Huaxiu Yao[6], Zhiwen Fan[7], Yuejiang Liu[5], Dawn Song[3], Pieter Abbeel[3], Li Fei-Fei[5], Jiajun Wu[5], Yuke Zhu[8], Guanya Shi[2†], Ruohan Zhang[1†], Manling Li[1†]

[1]Northwestern University [2]Carnegie Mellon University [3]UC Berkeley [4]University of Washington
[5]Stanford University [6]UNC-Chapel Hill [7]Texas A&M University [8]UT Austin
*Equal contribution §Core contributor †Equal advising

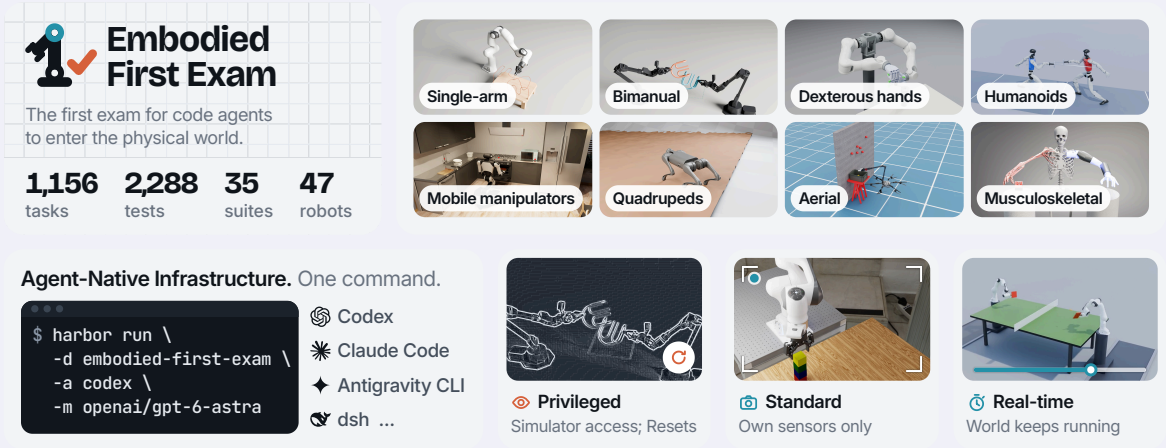


Figure 1 | Overview of the Embodied First Exam, a comprehensive evaluation framework for coding agents on robotics tasks. It is large and diverse, spanning 1,156 tasks, 35 suites and 47 robots (top), and easy to run and evaluate: one command runs any coding agent, and each attempt is graded in up to three evaluation settings (bottom).

Website: embodied-first-exam.ai
Code: github.com/mll-lab-nu/embodied-first-exam
Trajectory Data: huggingface.co/datasets/MLL-Lab/embodied-exam-data

Abstract

Early successes on agentic robotics point to a potential paradigm shift toward using coding agents to control robots. Existing robotics benchmarks built around visoumotor policy learning, leaving it open whether coding agents can reliably control diverse robots under physical-world constraints. We therefore introduce the **Embodied First Exam**, a unified, comprehensive, and agent-native evaluation framework where coding agents can choose their own control approach: direct control, code to control, or train a policy to control. We hope it serves as a first exam for coding agents before they enter the physical world, with 1,156 tasks in 35 suites on 47 robots, including 9 new suites that go beyond pick-and-place to spatial puzzles, dexterous manipulation, multi-robot interaction, dynamic whole-body control, and sim-to-sim engineering. Each task can be run in up to three evaluation settings. The `Standard` Setting mimics the real world: the agent receives only sensor observations and, by default, cannot reset the environment. Its `Realtime` variant keeps the environment running while the agent reasons. The `Privileged` Setting gives the agent access to the full simulator state and unlimited resets, allowing it to search for an oracle solution for training purposes. Success rates in this setting serve as an approximate upper bound. On the 391 core tasks with both `Privileged` and `Standard` settings, the strongest of seven agents, GPT-6 Astra in Codex, achieves a success rate of 30% in the `Standard` Setting, leaving much room for improvement. To see where agents fail, we label the stage at which each failed attempt goes wrong and compare failures across agents, settings, and stage categories.

1. Introduction

Robots acquire new skills either by manual programming or by learning from data, as in recent vision-language-action models [7, 9, 31, 61, 105, 116] and world action models [127, 153, 176]. Coding agents powered by large language models suggest a third way. They already complete challenging software engineering tasks in the digital world [58, 67, 97, 99]. What makes these agents promising for robotic engineering is their capabilities to turn instructions into executable code and revise their plans through automated experimentation [78, 135, 145]. Recent early success in agentic robots suggests such a potential paradigm shift, where agents directly control robots [41, 124, 185, 193], write robot programs [27, 42, 56, 96, 139, 161], or train policies [77, 160]. This motivates a systematic assessment of the capability boundary of coding agents:

What robotic tasks can general-purpose coding agents solve, under what physical-world constraints, and where do they fail?

Existing benchmarks do not yet fully address this question. The capability boundary cannot be established from a focused set of robotics tasks [14, 53, 70, 81, 101, 178], nor from automated programming tasks without imposing physical-world constraints [11, 58, 97, 111, 163, 192]. This requires defining which robotic capabilities to evaluate, measuring how physical-world constraints affect success, and supporting different models and harnesses through task interfaces.

To answer this question, we introduce the **Embodied First Exam (EFE)**, a unified evaluation framework that connects coding agents to simulated robots (Figure 1), following three principles.

First, include diverse tasks that go beyond pick-and-place. Spatial puzzles test reasoning about spatial constraints, while sim-to-sim engineering tests how agents revise controllers from execution feedback under unfamiliar dynamics. Dynamic tasks test responses to a changing world. Multi-robot interaction and whole-body control test coordination. EFE therefore contains 1,156 tasks in 35 suites on 47 robots. It combines 26 suites adapted from existing robotics benchmarks with 9 new suites covering these capabilities and safety.

Second, measure how success depends on physical-world constraints. Its tasks should follow the conditions of a real robot as closely as possible. Success with full simulator state and resets does not establish what an agent can achieve through partial observations, with no resets, or while the dynamic world keeps running. We therefore define up to three evaluation settings per task. `Standard` Setting follows real-robot conditions: the agent observes the environment only through the robot's sensors and cannot reset it by default. The simulation waits while the agent reasons, and its `Realtime` variant keeps the environment running while the agent makes decisions. `Privileged` Setting provides an upper bound and can be used to generate training data, where the agent has access to the full simulator state and unlimited resets, allowing it to search for an oracle trajectory by trial and error. We call each <task–setting> pair a *test*. Tests of the same task share the initial scene and success rule, allowing us to compare success as state access, reset limits, and timing change.

Third, provide unified infrastructure for different models, harnesses, and control approaches. With a single command, EFE runs different combinations of models and harnesses across diverse robotics tasks. Given a natural-language task instruction, the agent chooses its control approach (Figure 2). In *Direct control*, it directly outputs actions after receiving observations [41, 185, 193]. In *Code to control*, it outputs a planner, controller, or other program that generates the robot's actions [78]. In *Train a policy to control*, it trains a policy in simulation and uses it to control the robot [77, 90]. The framework enforces the access permitted by each

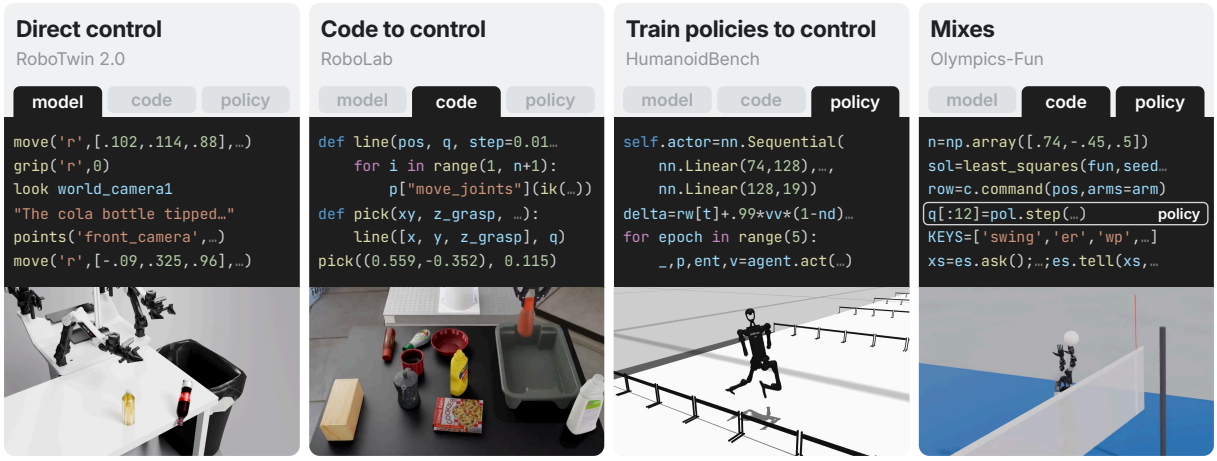


Figure 2 | Examples of the control approaches, each with an excerpt of the agent's commands or code. Dark tabs mark the approaches used (*model* is direct control), and the framed line is a policy.

evaluation setting, and a verifier outside the agent's control grades every attempt [190, 197].

We evaluate seven coding agents on a core set of 456 hard tasks. The strongest agent, GPT-6 Astra [110] in Codex, solves 77% of the `Privileged` tests but only 30% of the `Standard` tests of the same 391 tasks. None of the four weaker agents solves more than 6% of the `Standard` or `Realtime` tests (Section 3.1). The `Realtime` Setting is harder still: 65% of its tests remain unsolved by any agent, compared with 54% of the `Standard` tests of the same 54 tasks. To identify where failures occur within a task, we divide each task into stages. Each stage corresponds to a change in the world required to complete the task. For example, the task of putting a tomato into a closed fridge has three stages: the fridge is open, the tomato is inside, and the fridge is closed again. For each of 3,871 failed attempts, a separate coding agent labels the stage at which the failure occurred. Our main findings are as follows.

- **`Privileged` versus `Standard` (Section 3.2).** Resets help GPT-6 Astra more than the exact object state does, and with both, its success rate comes close to that in the `Privileged` Setting. Stages that need precision become much harder in the `Standard` Setting. At these stages, resets help, but the exact state alone does not.
- **Where agents fail (Section 3.3).** Failures on a task tend to happen at the same stage, usually the first stage of the task. This stage also tends to be the same in the `Privileged` and `Standard` settings. Stages that change a material or move the robot's body have the highest failure rates.
- **Differences between agents (Section 3.4).** Weaker agents are much more likely than stronger agents to fail before making the required contact. In the `Standard` Setting, the three strongest agents solve partly different sets of tests. Switching harnesses can change a model's success rate by about 10 percentage points.
- **How agents control the robot (Section 3.5).** In the `Standard` and `Realtime` settings of a soccer task, agents trained a policy when given a GPU and a training library, and only trained policies won. Agents mostly control the robot directly in the `Standard` Setting and write programs to control it in the `Privileged` and `Realtime` settings.

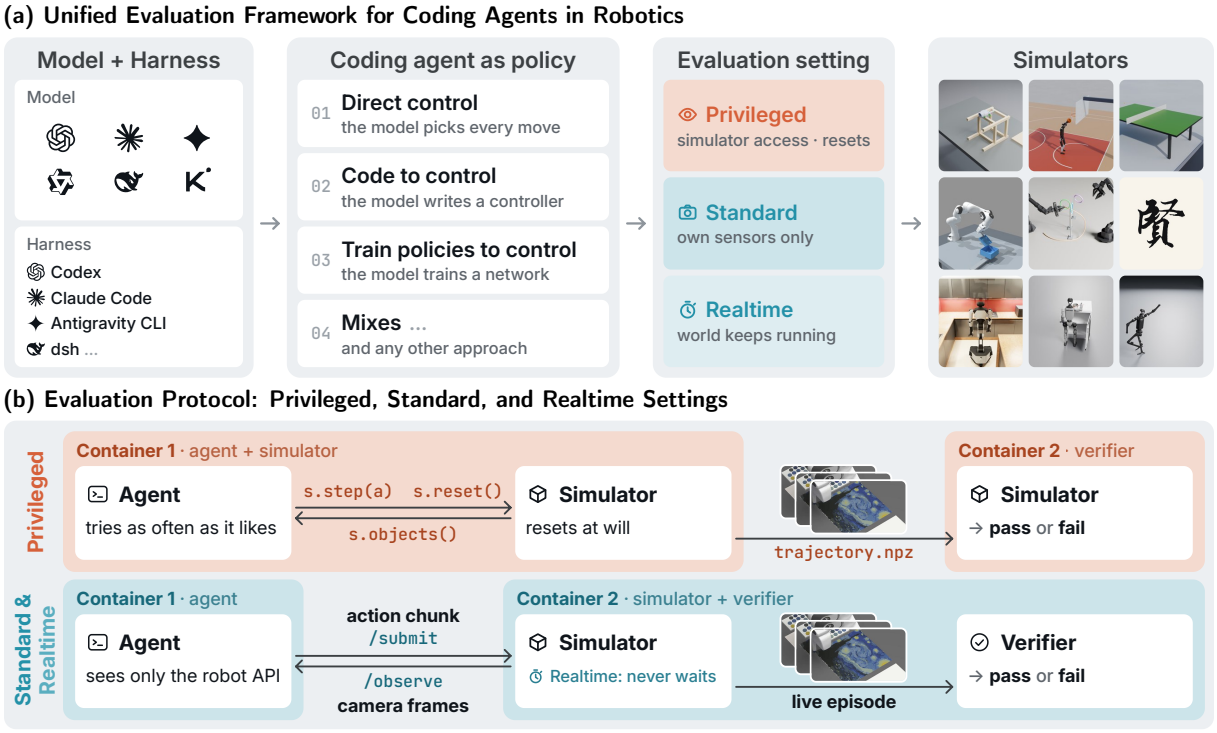


Figure 3 | The EFE framework and evaluation protocol. (a) Our unified framework supports different models, harnesses, control approaches, and simulators. (b) In `Privileged`, the agent and simulator share a container, and the verifier replays the submitted answer separately. In `Standard` and `Realtime`, the simulator and verifier are separated from the agent and grade the live episode.

2. The Embodied First Exam: A Unified Evaluation Framework for Coding Agents in Robotics

In this section, we describe how we evaluate coding agents on robotics tasks (Section 2.1), the resource constraints of each task (Section 2.2), and the tasks of the exam (Section 2.3).

2.1. Framework and Evaluation Protocol

Unlike previous robotics benchmarks, which test trained policies with a fixed input and output format, we evaluate coding agents that can decide on their own how to act. To evaluate them on robotics tasks, we want to leave the process open and constrain only the result: the agent may solve a task in any way, while we fix where it starts, what resources it may use, and how its result is graded. To achieve this, we package every robotics task in the Harbor format [97], which runs the task in containers, as shown in Figure 3a. The agent has its own workspace, where it can install packages, write and run programs and train models, and it chooses its own control approach. Each task starts from a fixed initial state; a *verifier* outside the agent's container checks the task's success rule on the agent's answer.

On top of this framework, we define a unified evaluation protocol with up to three settings per task (Figure 3b). We design `Privileged` to measure an upper bound on performance by giving the agent full access to the simulator for trial and error, while `Standard` and `Realtime` make the evaluation as close to a real robot as possible. For a given task, all settings share the same initial scene, success rule, and time limit, so their results are paired. They differ only in

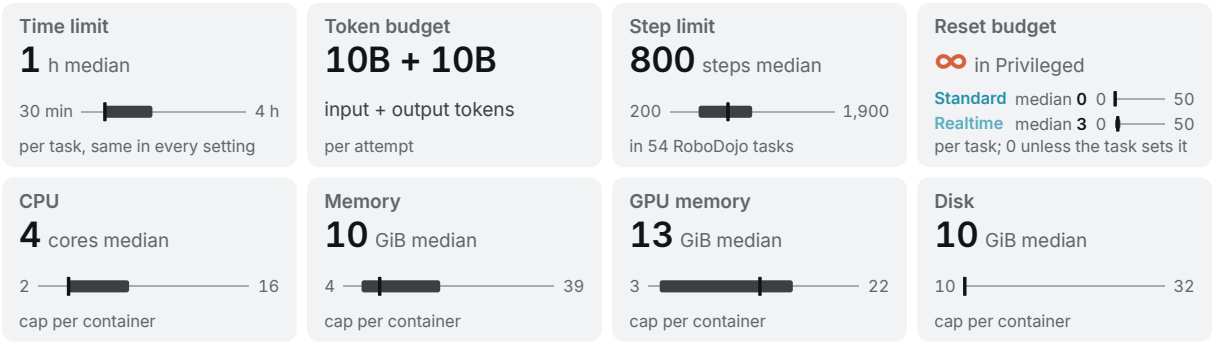


Figure 4 | Resource constraints across the exam. The large number and the tick give the median over tasks, the line spans the range, and the dark box spans the middle half of the tasks.

what the agent observes, whether it can reset, whether the world waits, and where the simulator and the verifier run, never in how the agent must act. We ablate these differences in Section 3.2.

In `Privileged`, the agent and the simulator share the agent's container, so the agent reads the full state and resets as often as it likes. It submits a low-level robot trajectory, or a controller program in some tasks, and we replay it with the verifier in a separate container. In `Standard`, we run the simulator as a *simulator service* in a separate container. Over HTTP, the agent receives only the robot description (kinematics and joint limits), proprioceptive state (such as joint positions), and camera observations (such as calibrated RGB-D images), and sends action chunks. The simulation waits between requests, there are no resets by default, and the service grades the *live episode* it ran. `Realtime` is the same as `Standard`, except that the world keeps running at a fixed rate while the agent reasons. Section C details each setting.

2.2. Resource Constraints for Coding Agents

As shown in Figure 4, we set resource constraints for each task: a time limit and a token budget for the agent, a reset budget and, in some tasks, a step limit for the robot, and caps on CPU, memory, GPU memory, and disk. The time limit is the same in every setting of a task, and when it runs out we stop the agent and grade the result it has at that point. Resets are unlimited in `Privileged` and zero by default in `Standard` and `Realtime`. We set the compute caps per task from its measured needs and monitor usage throughout the run (Section C).

We observe that coding agents may choose different control approaches under different resource constraints and in different settings (Figure 2 and Section 3.5). For example, if we provide a GPU, agents can train a policy to control the robot (Figure 18).

2.3. Task Suites

The exam contains 1,156 tasks in 35 suites. A suite is a set of related tasks on one simulator. Each task states a goal, usually in natural language, such as untying a knot or turning the page of a book with a dexterous hand, and the agent must control the robot to reach that goal. We built 9 suites with 411 tasks for the exam and adapted the other 26 suites with 745 tasks from existing robotics benchmarks. Together, the tasks use 47 robots of eight embodiments (Figure 5). Table 2 in Section A lists the simulator, the robots and the number of tasks of every suite.

New suites. We built nine new suites because the adapted suites contain few tasks for several embodiments and capabilities. Most of their original benchmarks were designed to evaluate

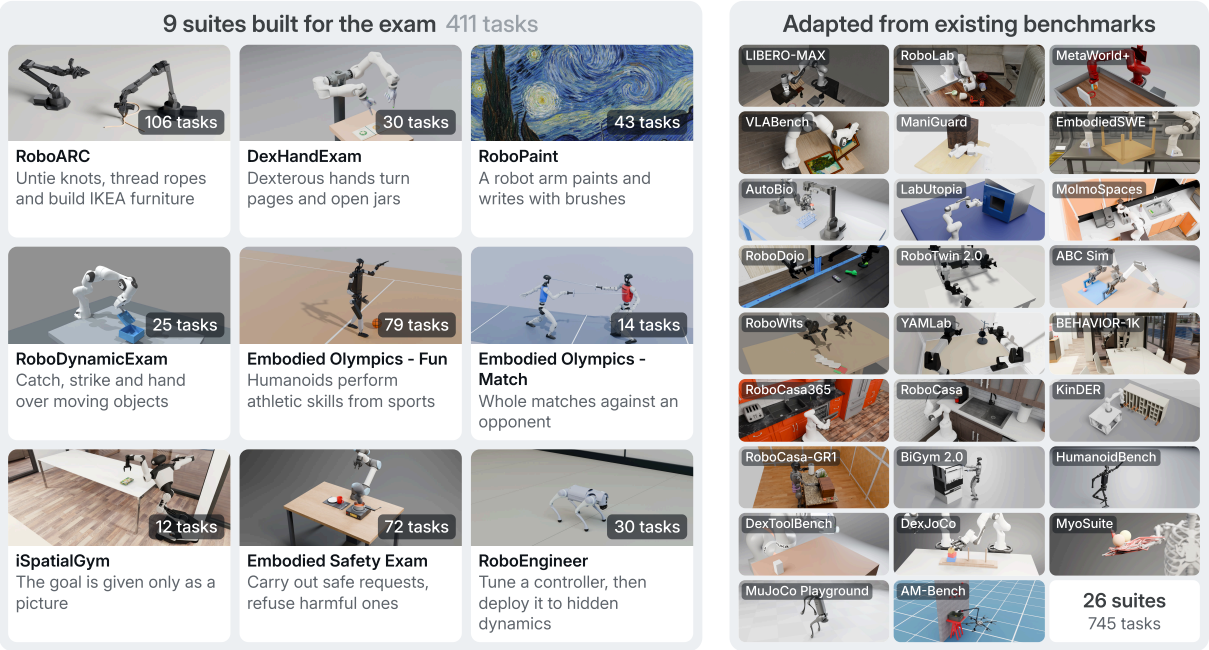


Figure 5 | The tasks of the exam. Left: the 9 suites built for the exam, each with a frame and its number of tasks, in the three groups of Section 2.3 by row. Right: a frame from each of the 26 adapted suites, ordered by their main robot embodiment, from single arms to aerial robots.

trained policies (Section 4), and 618 of their 745 tasks use single-arm, bimanual or mobile manipulators. Only 10 of these tasks are dynamic, that is, contain something other than the robot that moves on its own. The new suites form three groups, which are the rows of Figure 5, and Section A.1 describes each suite. RoboARC, DexHandExam and RoboPaint require contact-rich manipulation of deformable objects, multi-part assemblies and tools. RoboDynamicExam and the two Embodied Olympics suites involve moving objects and opponents, so the agent must act at the right time, and they contain most of the humanoid tasks of the exam. iSpatialGym, Embodied Safety Exam and RoboEngineer test abilities beyond executing a stated goal, since they give the goal only as an image, mix hazardous with benign requests, or require adapting a controller to hidden dynamics.

Adapted suites. The other 26 suites come from existing benchmarks, and their tasks range from language-conditioned tabletop manipulation and long-horizon household activities to humanoid whole-body control, dexterous tool use, musculoskeletal control, aerial manipulation and laboratory automation (Section A.2). For each suite, we keep the scenes, robots and success rules of the original benchmark, in some suites with a stricter success rule. We save the initial state of each task as a frozen scene, write an instruction that states every convention that the agent cannot infer and connect the suite to the interfaces of the three settings. We also remove the expert policies and demonstration data of the original benchmark, so that the agent cannot copy a known solution from its container.

Core set. We report results on a core set of 456 hard tasks (901 tests), because tasks within a suite often require similar skills and current agents already solve many of the tasks. The core set excludes tasks that we judged easy and tasks that an initial screening agent, GPT-6 Luna, solved, and it keeps every task that the authors of a suite asked to keep. Because GPT-6 Luna belongs to the same model family as GPT-6 Astra, the screening may bias the core set against tasks that this family solves. Table 2 gives the number of core tasks in each suite. Each core task

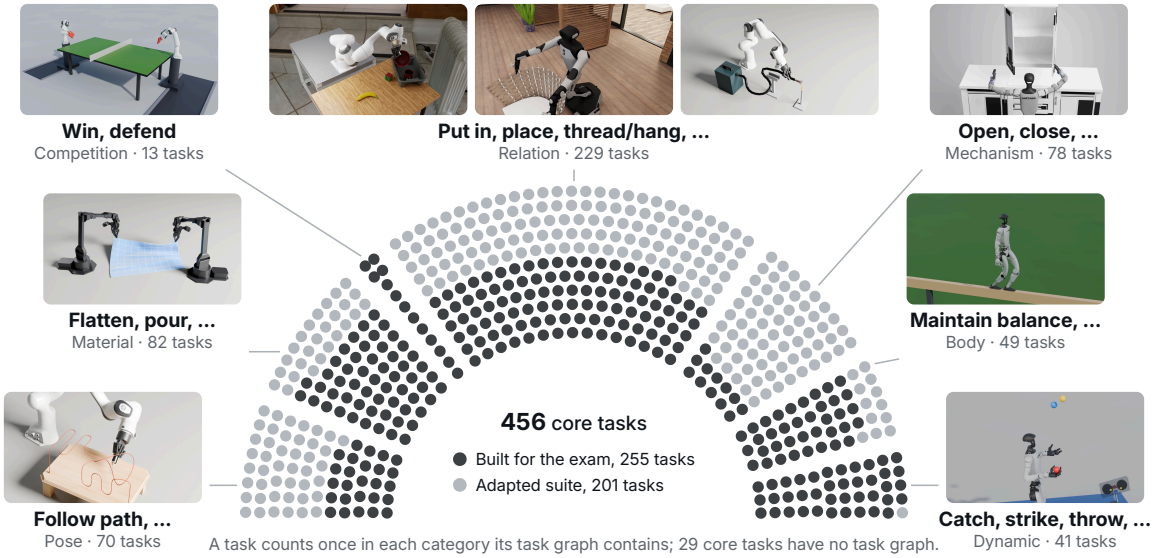


Figure 6 | The core tasks by the stage categories of their task graphs. Each frame shows an example task; its label lists the frames' stage types, then other common ones, and the category's name and number of tasks. Each seat is one task in one category, and the angle of a category is proportional to its number of tasks. Dark seats are tasks of the new suites.

also has a default setting, which the Core view of Section 3.1 uses. The default is `Standard` for 403 tasks and `Realtime` for 53 tasks in which timing is essential. The remaining 700 tasks stay in the exam and can be run in the same way. We split each core task into stages, such as opening a fridge (Section 3), and Figure 6 groups the core tasks by the categories of these stages (Table 7). Relation stages, such as placing or assembling objects, are the most common, and most tasks with Dynamic, Body or Competition stages come from the new suites (86 of 99).

3. Analysis

We evaluate seven coding agents on the core set (Figure 7). Section 3.1 gives their success rates, and Sections 3.2 to 3.5 report four findings. An agent is a model in a harness at high reasoning effort, and the harness is part of what we measure (Section 3.4). Each agent makes one attempt per test, and a test is solved when the verifier passes it (Section B). Success rates are the final results of the leaderboard, which have not changed since 2 November 2026 (Section D).

Stage-level failure labels. The verifier decides whether an attempt passed, but not where it went wrong. We therefore split each task into stages, such as opening the fridge, in a task graph (Figure 25 in Section H.1). Each stage has a type, such as *open*, in one of nine categories (Table 7). A failed attempt has at most one failure stage. If an irreversible event, such as a fall, occurs, it is the stage of this event, unless an earlier stage had already failed permanently. Otherwise, it is the stage of the earliest mistake that the agent did not correct. It also has up to two failure phenomena, such as a lost hold. An attempt reaches a stage if it meets the stage's condition or fails there. GPT-6.1 Sol in Codex labeled the 3,871 failed attempts that we analyze, and Claude Sonnet 5.5 in Claude Code reviewed the labels (Section H.1). We checked part of the labels by hand.

3.1. Overall Results

Model	Harness	Core (456 tests)	Privileged (391 tests)	Standard (456 tests)	Realtime (54 tests)
GPT-6 Astra	Codex	**32.9%** (150)	**77.5%** (303)	**34.2%** (156)	**31.5%** (17)
Claude Opus 5.5	Claude Code	**23.5%** (107)	**59.1%** (231)	**25.0%** (114)	**7.4%** (4)
Gemini 4 Argon	Antigravity CLI	**19.5%** (89)	**41.9%** (164)	**21.1%** (96)	**7.4%** (4)
Gemini 3.8 Flash	Antigravity CLI	5.9% (27)	24.6% (96)	5.9% (27)	1.9% (1)
Qwen3.8 Max	Claude Code	3.9% (18)	9.5% (37)	3.9% (18)	1.9% (1)
DeepSeek V4.1 Flash	dsh	2.6% (12)	17.6% (69)	2.6% (12)	1.9% (1)
GLM 5.3 Flash	Claude Code	2.4% (11)	9.2% (36)	2.4% (11)	0.0% (0)

Figure 7 | Success rate of each agent on the core set, with solved tests in parentheses. The Core view counts each core task once in its default setting.

Three agents each solve more than three times as many core tasks as any of the other four. In the Core view, GPT-6 Astra, Claude Opus 5.5 and Gemini 4 Argon solve 19.5% to 32.9% of the core tasks (Figure 7). We call them the three strongest agents and the other four the weaker agents. The `Realtime` Setting is harder than the `Standard` Setting. On the 54 tasks with both tests, GPT-6 Astra solves 23 `Standard` and 17 `Realtime` tests, Claude Opus 5.5 solves 10 and 4, and Gemini 4 Argon solves 11 and 4. 65% of these `Realtime` tests remain unsolved by any agent, compared with 54% of the `Standard` tests of the same tasks.

Together, the three strongest agents solve 48% of the `Standard` tests. Figure 8 shows the outcome of every attempt, with the tests ordered first by GPT-6 Astra's result and then by the number of agents that solve them. In the `Standard` Setting, GPT-6 Astra alone solves



Figure 8 | Every attempt on the core `Privileged` and `Standard` tests. Rows are agents in the order of Figure 7. Columns are tests: first those GPT-6 Astra solves, then those it fails, each ordered by how many agents solve it (the number above each bracket). Shades give the list-price cost of a solved attempt; pale cells are failures.

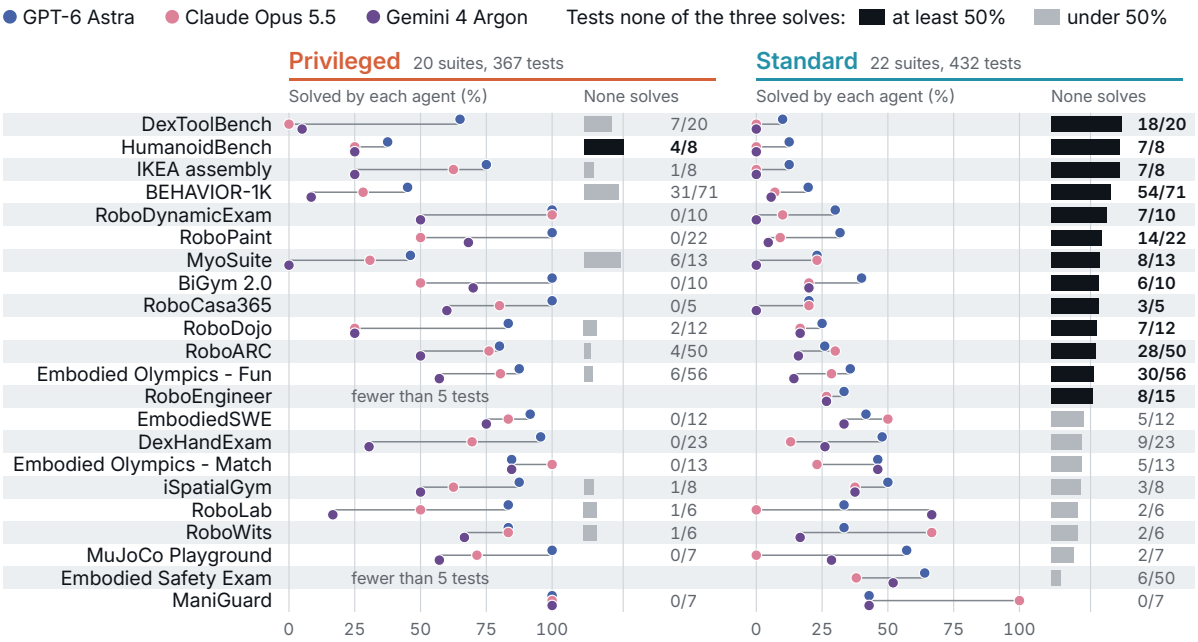


Figure 9 | Success rates by suite, for suites with at least five core tests in the setting. Dots give the success rate of each of the three strongest agents, and bars the share of tests that none of them solves. The IKEA assembly tasks of RoboARC count as their own suite.

156 of the 456 tests (34%). Adding Claude Opus 5.5 and Gemini 4 Argon raises the number of tests solved by at least one agent to 220 (48%). The four weaker agents raise it only to 222. In the `Privileged` Setting, GPT-6 Astra alone solves 303 of the 391 tests (77%), and the three strongest agents together solve 325 (83%). The four weaker agents add no tests. Across all seven agents, 234 `Standard` tests (51%) and 66 `Privileged` tests (17%) remain unsolved. Section 3.4 examines which tests each agent solves.

In most suites with at least five `Standard` tests, at least half of these tests are solved by none of the three strongest agents. This holds in 13 of these 22 suites. Of the 20 suites with at least five `Privileged` tests, it holds only in HumanoidBench (Figure 9). The pooled success rate of the three strongest agents is at least 70% in 10 `Privileged` suites and in no `Standard` suite.

3.2. What Causes the Drop from the `Privileged` to the `Standard` Setting?

Key Takeaways: `Privileged` versus `Standard`

- **Resets help GPT-6 Astra more than the exact object state does.** Resets raise its success rate from 33% to 52%, the state to 47%.
- **Stages that need precision become much harder in the `Standard` Setting.** Their failure rate rises by 23 to 52 percentage points, that of coarse stages by 3 to 23.
- **At stages that need precision, resets help GPT-6 Astra, but the exact state alone does not.** Of its failures there, a second run solves 19%, a run with the state 20% and a run with resets 36%.

Figure 10 pairs the `Privileged` and `Standard` tests of the 391 core tasks that have both. Of the 303 tasks that GPT-6 Astra solves in the `Privileged` Setting, it fails 189 in the `Standard` Setting. It solves only 5 tasks in the `Standard` Setting alone. Its success rate drops in each of the 20 suites with at least five such tasks. Two of the features that the `Standard` Setting lacks are

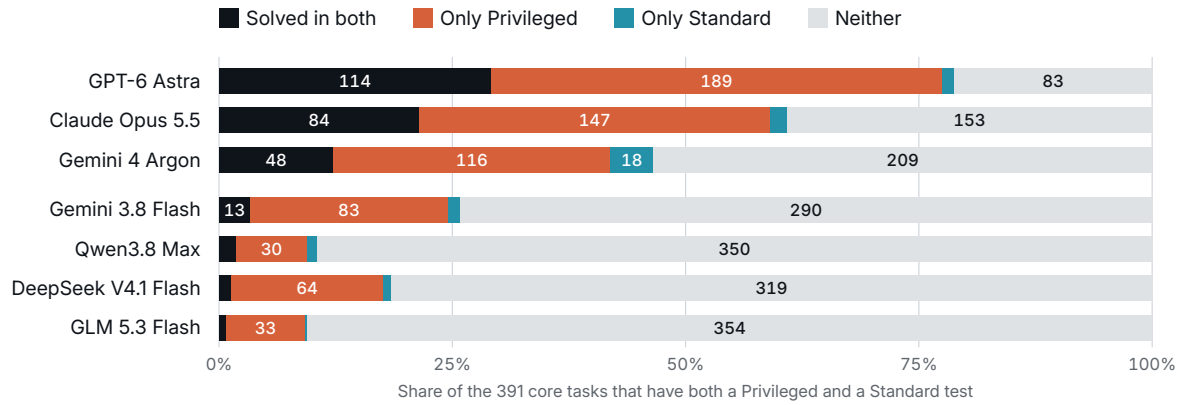


Figure 10 | The 391 core tasks that have both a `Privileged` and a `Standard` test, split for each agent by the settings in which it solves them.

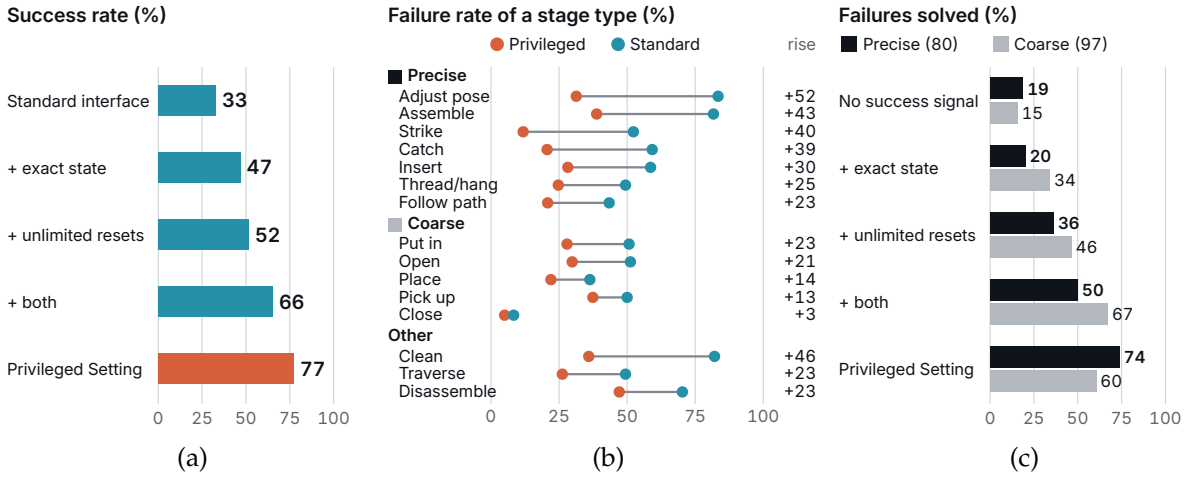


Figure 11 | (a) Success rate of GPT-6 Astra in Codex as features of the `Privileged` Setting are added to the `Standard` interface. (b) Failure rate of each stage type, all agents. (c) Share of GPT-6 Astra's failures at precise and coarse stages that each run solves.

the exact state of every object and unlimited resets.

Adding resets closes more of this gap than adding the exact state. In controlled runs of GPT-6 Astra in Codex, we add one or both of two features to the `Standard` interface: a query for the exact state of every object, and unlimited resets with saved states (Figure 11a, Section G). On the 456 core tests, the success rate rises from 33% to 47% with the state, to 52% with resets and to 66% with both. On the 391 `Privileged` tests, it is 77%. With resets, 58% of the successes come in an episode that starts from a saved state, so the agent retries a hard stage without repeating the whole task (Section E.1).

Stages that need precision fail much more often in the `Standard` Setting. From the stage labels of all seven agents, we compute the failure rate of each stage type in each setting: the number of stages of the type at which attempts fail, divided by the number that attempts reach. We compare the settings within each suite (Figure 11b). A stage type is precise when it needs a tight fit, a target pose or path, or contact with a moving object, and coarse when it needs only a region, a container, a grasp or the state of a mechanism (Section H.5). Among the types compared in at least two suites, the failure rate rises by 23 to 52 percentage points at precise

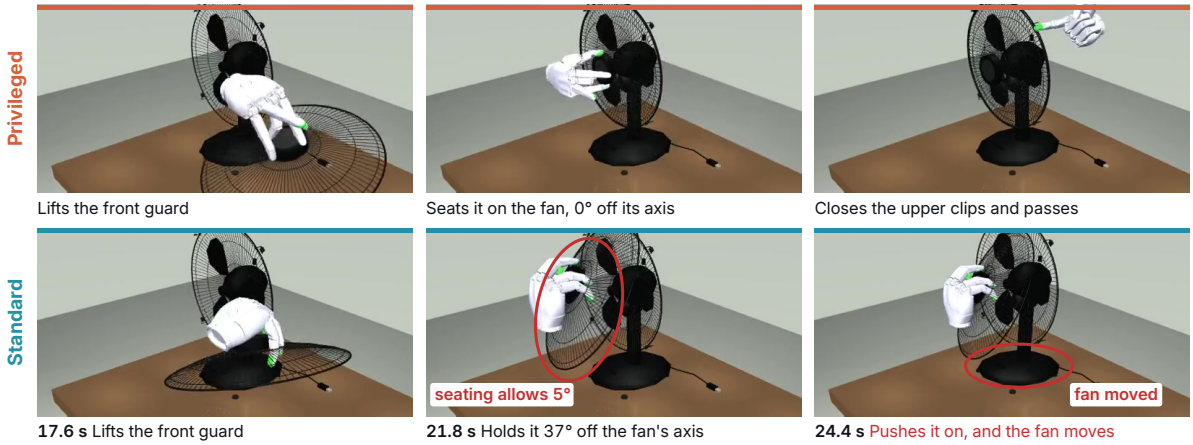


Figure 12 | Claude Opus 5.5 installing the front guard of a fan (DexHandExam) in the `Privileged` and the `Standard` Setting. Red marks where the `Standard` attempt goes wrong. Times are seconds on the replay of the `Standard` episode.

types and by 3 to 23 points at coarse types. Assembling, for example, rises from 39% to 82%, and placing from 22% to 36%. If we count only stages that the same agent reaches on the same task in both settings, the difference between the two groups is smaller: precise stages rise by 21 points and coarse stages by 17 (Section E.1). Figure 12 shows a precise stage that fails in the `Standard` Setting. There, Claude Opus 5.5 holds the front guard of a fan 37 degrees off the fan's axis, where seating allows 5 degrees, and pushing it further moves the whole fan. Section H.6 gives the full attempt and two more paired examples (Figure 33).

At precise stages, resets help GPT-6 Astra, but the exact state alone does not. We take the `Standard` and `Realtime` tests that GPT-6 Astra failed in the leaderboard run and group them by the type of the stage at which they failed (Figure 11c). Some of these failures would succeed in any second attempt. To estimate how many, a fourth run adds neither feature and only hides the success signal. Of the 80 failures at precise stages, this run solves 19%, the run with the state 20% and the run with resets 36%, with 28 of its 29 successes coming after a reset or a restored state. Compared with the fourth run, the state thus solves about one more of the 80 failures, and resets about 14 more. At coarse stages, by contrast, the state alone solves 34% of the 97 failures, against 15% for the second attempt (Section E.1).

3.3. At Which Stage of a Task Do Agents Fail?

Key Takeaways: Where Agents Fail

- **Failures on a task tend to happen at the same stage.** For the three strongest agents, 57% of failure pairs share a stage, against 31% by chance.
- **This stage also tends to be the same in the `Privileged` and `Standard` settings.** Leaving out first-stage failures, 55% of tasks match, against 9% by chance.
- **Stages that change a material or move the robot's body have the highest failure rates.** Once reached, they fail at 64% and 45%.

Failures on a task tend to happen at the same stage. The concentration of failure stages measures this (Figure 13a). For each test, it is the share of pairs of failed attempts, by different agents, that failed at the same stage. We use the attempts with a located failure stage and average this share over tests. The baseline draws each failure stage at random, with the probability of

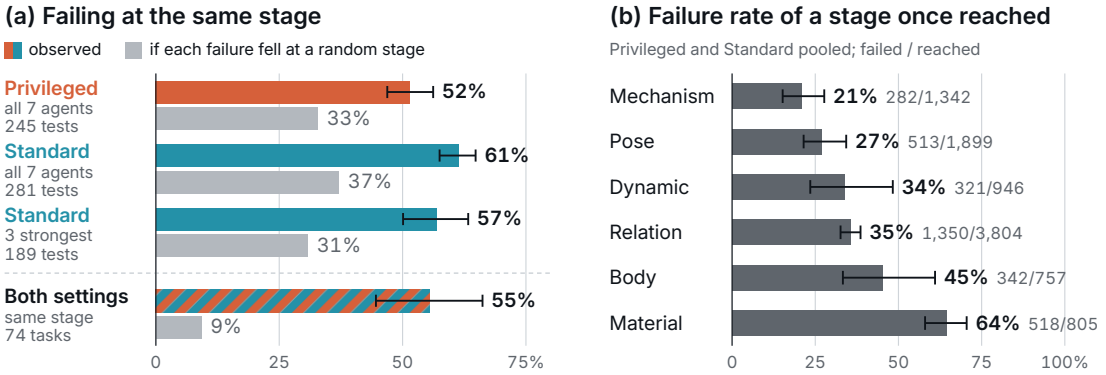


Figure 13 | (a) Top: concentration of failure stages, mean over tests. Bottom: share of tasks with the same most common failure stage in both settings, without the tasks where it is the first stage in both. Gray bars: random failure stages; black lines: 95% intervals. (b) Failure rate of a stage once reached, by category, both settings pooled (Table 18).

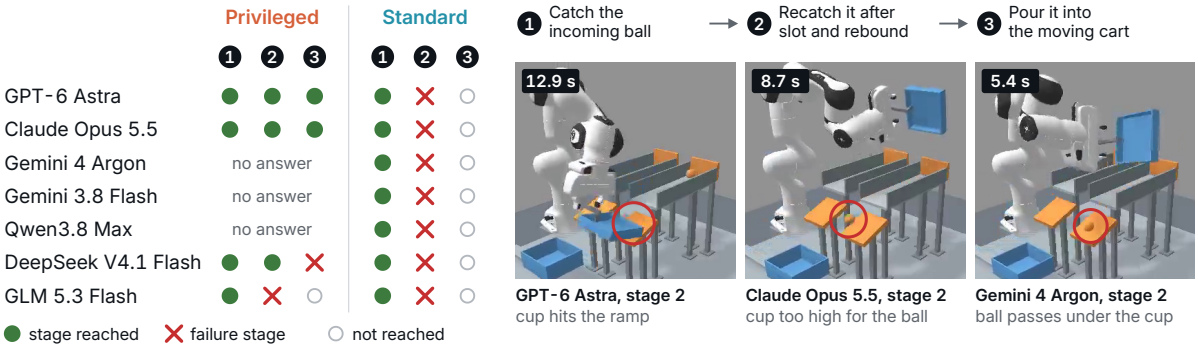


Figure 14 | Catching a ball twice (RoboDynamicExam). Left: the stages that each agent reached and the stage at which it failed, in each setting. Right: `Standard` replay frames at three failures at stage 2, with red circles where they happen.

each position in the task (first stage, second stage, and so on) taken from the failures on the other tasks of the setting (Section H.5). Over all seven agents, the concentration is 61% in the `Standard` Setting, against 37% for the baseline. The four weaker agents often fail before they reach any stage of a task (Section 3.4), so much of this agreement could come from failures at the first stage. We therefore check the three strongest agents separately. Over the 189 `Standard` tests with at least two located failures among them, the concentration is 57%, against 31% for the baseline built from their failures on the other `Standard` tasks. Figure 14 shows such a stage. In the `Standard` Setting, all seven agents catch a thrown ball in a cup and then fail, in different ways, to catch it again after it rebounds off a ramp. In Figure 32 in Section H.6, the three strongest agents each put the first of three batteries into a bin and then tip the robot over on the way to the second, whereas the four weaker agents fail at the first battery.

The stage at which agents fail also tends to be the same in the two settings. We take the 176 tasks whose failures have a single most common stage in both the `Privileged` and the `Standard` Setting. In 102 of them, this stage is the first stage in both settings, and we leave these tasks out. Of the other 74 tasks, 41 (55%) have the same common stage in both settings (Figure 13a). If the failure stages of each setting were drawn at random as above, this share would be 9%, again leaving out draws in which both common stages are the first stage (Section H.5).

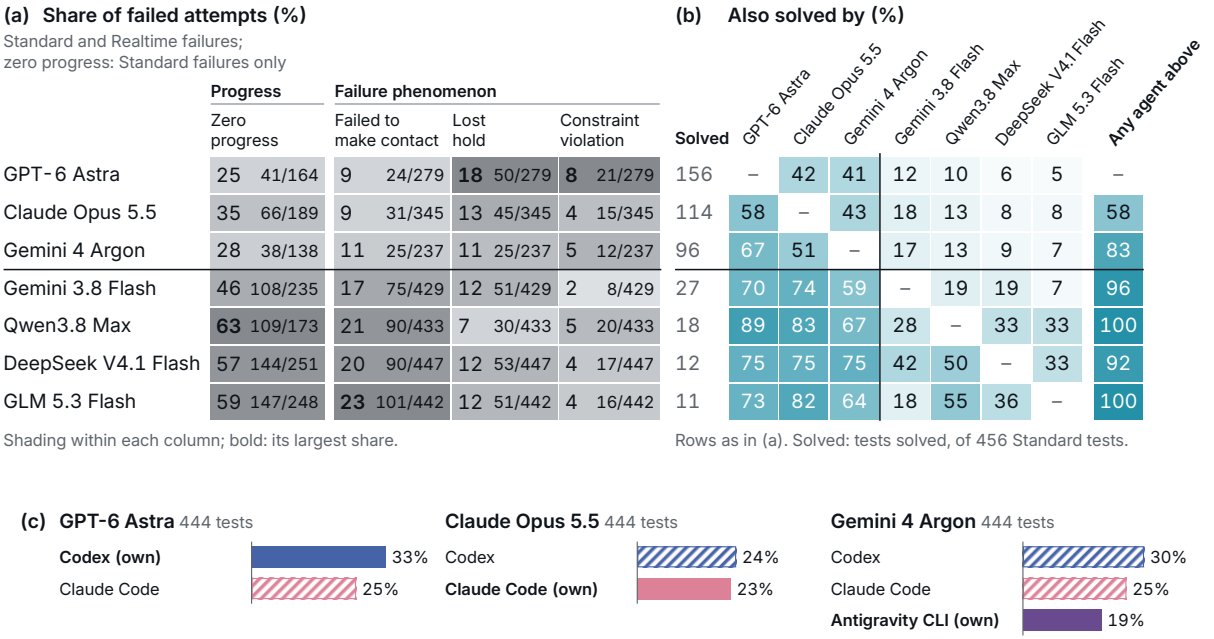


Figure 15 | (a) Share of failed `Standard` and `Realtime` attempts with each label; zero progress (no stage reached) counts only `Standard` attempts. (b) Share of the row agent's solved `Standard` tests (first column) that the column agent also solves; the last column uses any agent above it in Figure 7. (c) Success rate by model and harness; hatched bars mark other harnesses.

Stages that change a material or move the robot's body have the highest failure rates (Figure 13b). The failure rate of a stage category is the number of stages at which attempts fail divided by the number that they reach, with the `Privileged` and `Standard` settings pooled. Material stages, such as wiping, fail at 64%, body stages, such as walking, at 45%, and mechanism stages, such as opening a door, at 21%. Each category occurs mostly in particular suites and with particular robots, so these rates do not show that the category itself causes the failures (Section E.2).

3.4. How Do Agents Differ?

Key Takeaways: Differences between Agents

- **Weaker agents are much more likely than stronger agents to fail before making the required contact.** They do so at 11% to 18% of contact stages, the three strongest at 2% to 4%.
- **In the `Standard` Setting, the three strongest agents solve partly different sets of tests.** Between them, Claude Opus 5.5 and Gemini 4 Argon solve 64 tests that GPT-6 Astra fails.
- **Switching harnesses can change a model's success rate by about 10 percentage points.** Gemini 4 Argon solves 30% of 444 core tests in Codex and 19% in its own Antigravity CLI.

Weaker agents more often fail without making the required contact. In the failure profiles of Figure 15a, this happens in 17% to 23% of their failed `Standard` and `Realtime` attempts, against 9% to 11% for the three strongest agents. Of their failed `Standard` attempts on tasks with at least two stages, among those whose progress could be determined, they reach no stage at all in 46% to 63%, against 25% to 35%. Counted per contact stage that they reach, the weaker agents fail without contact at 11% to 18% of these stages, against 2% to 4% (Table 19). Figure 16

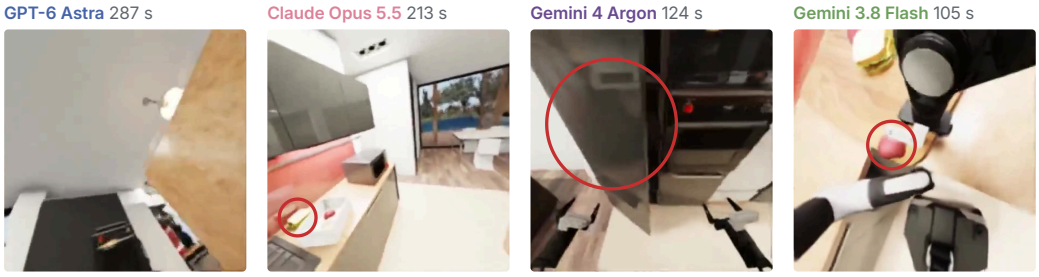


Figure 16 | Preparing a lunch box (BEHAVIOR-1K, `Standard` Setting): a head-camera frame of four failed attempts, with the failure circled.

shows a BEHAVIOR-1K test, preparing a lunch box, that all seven agents failed in the `Standard` Setting. The three strongest agents each packed three or four foods, whereas none of the four weaker agents packed any. Section H.6 describes all seven attempts and two more such tests (Figure 31). The labeling model, GPT-6.1 Sol, is from the same family as GPT-6 Astra, so these comparisons need a manual check of the labels (Section H.5).

The three strongest agents succeed on partly different tests. In the `Standard` Setting, no agent's solved tests are a subset of another agent's (Figure 15b). GPT-6 Astra solves 58% of Claude Opus 5.5's solved `Standard` tests and 67% of Gemini 4 Argon's. In the `Privileged` Setting, these shares are 92% and 95% (Figure 22). Between them, Claude Opus 5.5 and Gemini 4 Argon solve 64 `Standard` tests that GPT-6 Astra fails. Each agent made one attempt per test, so part of this difference may come from the variation between runs. Of the 47 `Standard` tests that at least one weaker agent solves, the three strongest agents solve 45. For each weaker agent, 92% to 100% of its solved tests are also solved by at least one agent above it in Figure 7, as the last column of Figure 15b shows (Section E.3).

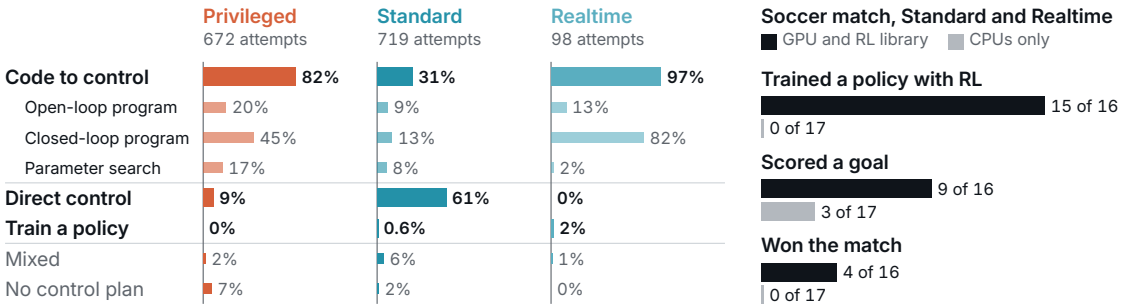
The harness also changes the result (Figure 15c). We ran GPT-6 Astra in Claude Code, Claude Opus 5.5 in Codex, and Gemini 4 Argon in both. On 444 core tests, Gemini 4 Argon solves 30% in Codex, 25% in Claude Code and 19% in its own Antigravity CLI. GPT-6 Astra solves 33% in Codex and 25% in Claude Code, and Claude Opus 5.5 solves 24% in Codex and 23% in Claude Code. When a model runs in another provider's harness, an adapter translates between the two, so these runs only approximate a native setup (Section G). Antigravity CLI ends the session when Gemini's content filter blocks a reply, whereas Codex and Claude Code let the session go on. This may contribute to Gemini 4 Argon's lower success rate in Antigravity CLI (Section E.3).

3.5. Which Control Approach Do Agents Use?

Key Takeaways: How Agents Control the Robot

- **In the `Standard` and `Realtime` settings of a soccer task, agents trained a policy when given a GPU and a training library, and only trained policies won.**
- **In the `Standard` Setting, agents mostly control the robot directly.** They do so in 61% of the sampled `Standard` attempts.
- **In the `Privileged` and `Realtime` settings, agents mostly write programs to control the robot.** They do so in 82% of the sampled `Privileged` attempts and 97% of the `Realtime` attempts.

In the `Standard` and `Realtime` settings of a soccer task, agents trained a policy when given



(a) (b)

Figure 17 | (a) Share of each approach by setting in 1,489 sampled attempts. Mixed attempts switched approach midway or used two approaches equally. (b) Soccer attempts in the `Standard` and `Realtime` settings that trained a policy, scored or won.



Figure 18 | GPT-6 Astra in Codex on the two soccer versions in the `Standard` Setting. Each row gives the approach and result, two replay frames and the goals over match time.

a GPU and a training library, and only trained policies won. The soccer match of Embodied Olympics - Match comes in two versions with the same frozen scene and opponent (Figure 17b). In both, the agent controls two humanoid robots joint by joint, so it needs a walking gait. The GPU version adds a GPU, a reinforcement-learning library, GPU copies of the match for training and an example of training in its instruction. On this version, 5 of 6 attempts trained a policy with reinforcement learning in the `Standard` Setting, and all 10 did in the `Realtime` Setting. Two attempts won in each setting, all with a trained policy. On the CPU version, agents wrote gaits by hand, and none of the 6 `Standard` and 11 `Realtime` attempts won. These counts include the runs of the harness comparison, so some models appear more than once (Table 4). In the `Privileged` Setting, agents also won on the CPU version, but every win there reused the opponent's walking network. The two versions differ in several ways, and this one task cannot show which difference matters. GPT-6 Astra, for example, trained policies and won 10–8 with the GPU, and tuned a hand-written gait and lost 3–13 without it (Figure 18).

In the `Standard` Setting, agents mostly control the robot directly, and in the `Privileged` and `Realtime` settings they mostly write programs (Figure 17a). A coding agent labeled the control approach of 1,489 sampled attempts, and no one has checked these labels by hand (Section E.4). Where the simulation pauses between requests, 61% of the sampled `Standard` attempts send one action chunk at a time and look at the camera images or sensor readings before

choosing the next. With the full state, 82% of the `Privileged` attempts compute the motion in a program. When the clock keeps running, 97% of the `Realtime` attempts use a program, and in 82% the program runs the control loop. Six of the 1,489 attempts were labeled as training a policy.

4. Related Work

This section reviews the closest lines of work. Section K gives an extended review.

Benchmarks for coding agents. Coding agents are evaluated mostly on digital tasks, such as resolving issues in real repositories [58, 99], completing tasks in a terminal [97], machine learning engineering and research [11, 137], and operating websites and desktop applications [163, 192]. The interface between the model and the computer affects how often an agent succeeds [168], and other work measures how often agents exploit the tests that grade them [190]. The Embodied First Exam extends these evaluations to robotics tasks, in which the agent must interpret sensor observations and often cannot undo an action. In every setting, a program that the agent cannot change grades each attempt.

Language models for robots. Language models have been used to sequence skills into plans [1, 46], to write robot programs that call perception and control primitives [78], to specify constraints that a planner turns into motions [47, 48], and to write reward functions [90]. Most of these systems pair a model with an interface designed for it. The Embodied First Exam evaluates unchanged general-purpose coding agents on the whole loop of reading an interface, writing a controller, running it and revising it. In embodied benchmarks for language models [72, 133, 170], the model chooses from an action space that the benchmark defines. The exam gives the agent the robot's own programming interface, so the agent also decides how its actions are produced.

Agentic robots and their harnesses. Recent work runs general-purpose agents on robots [119] with the three control approaches of Figure 2. In direct control, GPT-6 Astra serves as the policy on manipulation benchmarks [55, 185, 193]. In code to control, coding agents write programs for simulated and real robots [44, 50, 56, 144], often with skill libraries or other added tools [42, 88]. In train a policy, agents train policies in the real world or in simulation [77, 160]. Each of these studies examines one approach or a fixed combination of approaches. The exam supports all three and leaves the choice to the agent. Harnesses for these agents differ in the action interface they provide [15, 20, 29], in whether the agent can call learned policies as tools [28, 74], and in the support they add for tracking state and checking outcomes [26, 151]. The exam runs any harness and any model without modification, so different harnesses can be compared on the same tasks.

Benchmarks for robots. Most robotics benchmarks evaluate learned policies, such as vision-language-action models [7, 9, 31, 61], on manipulation, household and humanoid tasks [13, 70, 81, 101, 126]. A learned policy receives only a fixed set of observations. A coding agent can run any program on its computer, so the exam adds rules about what the agent may access (Section 2.1) and adapts 24 of these benchmarks (Section A.2). Benchmarks for agentic robots have also appeared. CaP-X [27] varies the level of abstraction of the primitives that coding agents compose, RobotWorld [172] spans manipulation, locomotion, driving and aerial control, and EmbodiedSWE [128] and BiGym 2.0 [188] evaluate coding agents on dexterous and humanoid tasks. The Embodied First Exam covers 47 robots of eight types. It also separates what an agent can do with full access to the simulator from what it can do through the robot's sensors alone.

5. Conclusion

We introduced the Embodied First Exam, a unified evaluation framework that connects any coding agent to simulated robots and lets the agent execute actions in three ways. The exam contains 1,156 tasks on 47 robots. On the 391 hard core tasks that have both settings, the strongest agent reaches a success rate of 77% in the `Privileged` Setting but only 30% in the `Standard` Setting. In controlled runs with this agent, adding unlimited resets closes more of this gap than adding the exact object state, and at stages that need precision, resets help but the exact state alone does not. Failures on a task tend to happen at the same stage, weaker agents are much more likely than stronger agents to fail before making the required contact, and in the `Standard` and `Realtime` settings of a soccer task, agents trained a policy when given a GPU and a training library, and only trained policies won. The strongest agent already solves most tasks when it has full access to the simulator, but in the `Standard` and `Realtime` settings, which are designed to mirror how a real robot is deployed and tested, it still has much room for improvement.

Future work. The stage-level labels were written and reviewed by agents, and we plan to check 150 to 200 failed attempts, sampled by agent and setting, manually against the replays and to report the agreement (Section H.5). Running each core test several times will separate failures that an agent repeats from failures of a single run, and it will give confidence intervals for the comparisons between agents and between settings. We will also add more `Realtime` tasks, since the `Realtime` Setting has only 54 core tests but the largest share of tests that no agent solves. Finally, because the `Standard` Setting gives the agent only the robot's sensor readings and control interface, the same interface can be used on physical hardware once there is a way to grade the outcome. In a first demonstration, two agents played a buzz wire game with a physical robot arm (Section I), and we plan to evaluate agents on real robots systematically through this interface.

Limitations. Each agent makes a single attempt per test, the stage-level and control-approach labels have not yet been checked manually, and all results are in simulation. Section J discusses these and further limitations.

Acknowledgements

We acknowledge and disclose the use of AI tools in coding and paper writing. Pieter Abbeel holds concurrent appointments as a Professor at UC Berkeley and as an Amazon Scholar. This paper describes work performed at UC Berkeley and is not associated with Amazon.

References

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, et al. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. *arXiv preprint arXiv:2204.01691*, 2022. URL https://arxiv.org/abs/2204.01691. CoRL 2022.

[2] Michael Ahn, Debidatta Dwibedi, Chelsea Finn, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Karol Hausman, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Sean Kirmani, Isabel Leal, Edward Lee, Sergey Levine, Yao Lu, Isabel Leal, Sharath Maddineni, Kanishka Rao, Dorsa Sadigh, Pannag Sanketi, Pierre Sermanet, Quan Vuong, Stefan Welker, Fei Xia, Ted Xiao, Peng Xu, Steve Xu, and Zhuo Xu. AutoRT: Embodied Foundation Models for Large Scale Orchestration of Robotic Agents. *arXiv preprint arXiv:2401.12963*, 2024. URL https://arxiv.org/abs/2401.12963. arXiv 2024.

[3] Firas Al-Hafez, Guoping Zhao, Jan Peters, and Davide Tateo. LocoMuJoCo: A Comprehensive

Imitation Learning Benchmark for Locomotion. *arXiv preprint arXiv:2311.02496*, 2023. URL https://arxiv.org/abs/2311.02496. arXiv 2023.

[4] Anthropic Frontier Red Team. Project Fetch: Can Claude train a robot dog? https://www.anthropic.com/research/project-fetch-robot-dog, 2025.

[5] ARISE Initiative. YAMLab. https://github.com/ARISE-Initiative/yamlab, 2026.

[6] 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, et al. RoboArena: Distributed Real-World Evaluation of Generalist Robot Policies. *arXiv preprint arXiv:2506.18123*, 2025. URL https://arxiv.org/abs/2506.18123. arXiv 2025.

[7] 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. *arXiv preprint arXiv:2410.24164*, 2024. URL https://arxiv.org/abs/2410.24164. arXiv 2024.

[8] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, et al. RT-1: Robotics Transformer for Real-World Control at Scale. *arXiv preprint arXiv:2212.06817*, 2022. URL https://arxiv.org/abs/2212.06817. RSS 2023.

[9] 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. *arXiv preprint arXiv:2307.15818*, 2023. URL https://arxiv.org/abs/2307.15818. CoRL 2023.

[10] Vittorio Caggiano, Huawei Wang, Guillaume Durandau, Massimo Sartori, and Vikash Kumar. MyoSuite – A contact-rich simulation suite for musculoskeletal motor control. *arXiv preprint arXiv:2205.13600*, 2022. URL https://arxiv.org/abs/2205.13600. arXiv 2022.

[11] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering. *arXiv preprint arXiv:2410.07095*, 2024. URL https://arxiv.org/abs/2410.07095. ICLR 2025.

[12] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, et al. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374. arXiv 2021.

[13] Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, Weiliang Deng, Yubin Guo, Tian Nian, Xuanbing Xie, Qiangyu Chen, Kailun Su, Tianling Xu, Guodong Liu, Mengkang Hu, Huan ang Gao, Kaixuan Wang, Zhixuan Liang, Yusen Qin, Xiaokang Yang, Ping Luo, and Yao Mu. RoboTwin 2.0: A Scalable Data Generator and Benchmark with Strong Domain Randomization for Robust Bimanual Robotic Manipulation. *arXiv preprint arXiv:2506.18088*, 2025. URL https://arxiv.org/abs/2506.18088. ICML 2026.

[14] Tianxing Chen, Yue Chen, Zixuan Li, Junyuan Tang, Kailun Su, Haoran Lu, Weijie Wan, Baijun Chen, Songling Liu, Haowen Yan, Honghao Su, Zhiyang Dou, Kaixuan Wang, Dandan Zhang, Yunze Liu, Yan Qin, Qiwei Liang, Qiwei Wu, Zijian Lin, Wenwei Lin, Yuran Wang, Minghua He, Tianshu Wu, Ruihai Wu, Jingquan Zhou, Kai-Chong Lei, Haibao Yu, Yuanfeng Ji, Weiyang Jin, Guanyu Lin, Xiaofan Li, Qi Xiong, Renjing Xu, Zhongyu Li, Wenhao Chai, Enze Xie, Ziwei Wang, Yao Mu, Hao Dong, Wojciech Matusik, Mingyu Ding, Wenbo Ding, Ping Luo, and Masayoshi Tomizuka. RoboDojo: A Unified Sim-and-Real Benchmark for Comprehensive Evaluation of Generalist Robot Manipulation Policies. *arXiv preprint arXiv:2607.04434*, 2026. URL https://arxiv.org/abs/2607.04434. arXiv 2026.

[15] Yanzhe Chen, Zechen Bai, Zhijun Cao, Wenzheng Zeng, Kevin Qinghong Lin, Yiqi Lin, Guoqiang Liang, Kevin Yuchen Ma, Qiming Huang, and Mike Zheng Shou. Show-Harness: Just a VLM Agent Can Play Robots. *arXiv preprint arXiv:2609.10522*, 2026. URL https://arxiv.org/abs/2609.10522. arXiv 2026.

[16] Yanzhe Chen, Ziyi Yang, Jifeng Zhu, Qiming Huang, Ruihe An, Peiyao Xu, Hesen Yang, Runda Liu, Chang Gong, Zhijun Cao, Zechen Bai, Wenzheng Zeng, et al. Multimodal Embodied Agents: A Unified Capability-centric Perspective from Computer-Use to Robot-Use. *arXiv preprint arXiv:2610.30108*, 2026. URL https://arxiv.org/abs/2610.30108. arXiv 2026.

[17] Yuanpei Chen, Tianhao Wu, Shengjie Wang, Xidong Feng, Jiechuang Jiang, Stephen Marcus McAleer, Yiran Geng, Hao Dong, Zongqing Lu, Song-Chun Zhu, and Yaodong Yang. Towards Human-Level Bimanual Dexterous Manipulation with Reinforcement Learning. *arXiv preprint arXiv:2206.08686*, 2022. URL https://arxiv.org/abs/2206.08686. arXiv 2022.

[18] Dongzhou Cheng, Taoran Yi, Ye Fang, Xingwu Zhang, Fan Feng, Yixuan Li, Gengxiong Zhuang, Rongze Wang, Shuai Yang, Wei Song, Weizhi Xue, Minyan Wu, Jie Gui, Jiaqi Wang, and Tong Wu. In-Context Robot Learning with VLM Agents. *arXiv preprint arXiv:2609.19138*, 2026. URL https://arxiv.org/abs/2609.19138. arXiv 2026.

[19] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. *arXiv preprint arXiv:2303.04137*, 2023. URL https://arxiv.org/abs/2303.04137. RSS 2023.

[20] Zhaoyang Chu, Earl T. Barr, Claire Le Goues, Peter O'Hearn, Mark Harman, Federica Sarro, and He Ye. OpenRUA: Robot-Use Agents Are Zero-Shot Visuomotor Policies. *arXiv preprint arXiv:2610.02459*, 2026. URL https://arxiv.org/abs/2610.02459. arXiv 2026.

[21] David B. D'Ambrosio, Saminda Abeyruwan, Laura Graesser, Atil Iscen, Heni Ben Amor, Alex Bewley, Barney J. Reed, Krista Reymann, Leila Takayama, Yuval Tassa, et al. Achieving Human Level Competitive Robot Table Tennis. *arXiv preprint arXiv:2408.03906*, 2024. URL https://arxiv.org/abs/2408.03906.

[22] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PaLM-E: An Embodied Multimodal Language Model. *arXiv preprint arXiv:2303.03378*, 2023. URL https://arxiv.org/abs/2303.03378. ICML 2023.

[23] Yilun Du, Mengjiao Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Joshua B. Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning Universal Policies via Text-Guided Video Generation. *arXiv preprint arXiv:2302.00111*, 2023. URL https://arxiv.org/abs/2302.00111. NeurIPS 2023.

[24] Karim Elmaaroufi, Justin Svegliato, Sarunas Kalade, Graham Schelle, Sanjit A. Seshia, and Matei Zaharia. RHO: Your Coding Agent is Secretly a Roboticist. *arXiv preprint arXiv:2606.16458*, 2026. URL https://arxiv.org/abs/2606.16458.

[25] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge. *arXiv preprint arXiv:2206.08853*, 2022. URL https://arxiv.org/abs/2206.08853. NeurIPS 2022 Datasets and Benchmarks Track.

[26] Zhirui Fang, Qingchi Yu, Ziyang Chen, Longfei Li, Haoran Ma, Keru Zhou, Xinrun Xu, Samith Va, Yuxuan Hu, Peixuan Song, Qiang Du, Bin Qian, Yongkang Deng, Xin Li, Yezhen Wang, Zhe Li, Hao Luo, Shuyan Li, Ziwei Wang, Weijian Deng, and Xiu Li. EMERGE-Policy: A Robot Mind Emerges Beyond a Single Policy. *arXiv preprint arXiv:2608.29896*, 2026. URL https://arxiv.org/abs/2608.29896. arXiv 2026.

[27] Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, Guanya Shi, Jiajun Wu, Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi Fan. CaP-X: A Framework for Benchmarking and Improving Coding Agents for Robot Manipulation. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, volume 306 of *PMLR*, 2026. URL https://arxiv.org/abs/2603.22435.

[28] Liane Galanti, Dhruv Shah, and Tri Dao. Addressing the Orchestration Gap in Generalist Robots via Physical Agency. *arXiv preprint arXiv:2607.21725*, 2026. URL https://arxiv.org/abs/2607.21725. arXiv 2026.

[29] Hongcheng Gao, Jianshu Zeng, Alex Zhou, Shijia Ge, Xuan Shangguan, Yazhe Wang, Yexing Wan, Zelin Zheng, Jay Zhu, Xiping Li, Di Wu, Yongli Wang, et al. HexaShell: A Universal Harness for Robot Coding Agents in Simulation and the Real World. *arXiv preprint arXiv:2610.23479*, 2026. URL https://arxiv.org/abs/2610.23479. arXiv 2026.

[30] Hongcheng Gao, Jingjing Zhou, Zelin Zheng, Shijia Ge, Jay Zhu, Yazhe Wang, Jianshu Zeng, Xuan Shangguan, Di Wu, Lingyu He, Zhiqi Jia, Sihang Wu, and Xiao He. Self-Evolving Coding Agents: From Digital Programs to Physical-World Intelligence. *arXiv preprint arXiv:2609.35432*, 2026. URL https://arxiv.org/abs/2609.35432. arXiv 2026.

[31] Gemini Robotics Team. Gemini Robotics: Bringing AI into the Physical World. *arXiv preprint arXiv:2503.20020*, 2025. URL https://arxiv.org/abs/2503.20020. arXiv 2025.

[32] Gemini Robotics Team. Gemini Robotics 1.5: Pushing the Frontier of Generalist Robots with Advanced Embodied Reasoning, Thinking, and Motion Transfer. *arXiv preprint arXiv:2510.03342*, 2025. URL https://arxiv.org/abs/2510.03342. arXiv 2025.

[33] Gemini Robotics Team, Krzysztof Choromanski, Coline Devin, Yilun Du, Debidatta Dwibedi, Ruiqi Gao, Abhishek Jindal, Thomas Kipf, et al. Evaluating Gemini Robotics Policies in a Veo World Simulator. *arXiv preprint arXiv:2512.10675*, 2025. URL https://arxiv.org/abs/2512.10675.

[34] Haoran Geng, Feishi Wang, Songlin Wei, Yuyang Li, Bangjun Wang, Boshi An, Charlie Tianyue Cheng, Haozhe Lou, Peihao Li, Yen-Jen Wang, Yutong Liang, Dylan Goetting, et al. RoboVerse: Towards a Unified Platform, Dataset and Benchmark for Scalable and Generalizable Robot Learning. *arXiv preprint arXiv:2504.18904*, 2025. URL https://arxiv.org/abs/2504.18904. arXiv 2025.

[35] Ken Goldberg. Good Old-Fashioned Engineering Can Close the 100,000-Year "Data Gap" in Robotics. *Science Robotics*, 10(105):eaea7390, 2025. doi: 10.1126/scirobotics.aea7390.

[36] Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, Xiaodi Yuan, Pengwei Xie, Zhiao Huang, Rui Chen, and Hao Su. ManiSkill2: A Unified Benchmark for Generalizable Manipulation Skills. *arXiv preprint arXiv:2302.04659*, 2023. URL https://arxiv.org/abs/2302.04659. ICLR 2023.

[37] Yanjiang Guo, Lucy Xiaoyang Shi, Jianyu Chen, and Chelsea Finn. Ctrl-World: A Controllable Generative World Model for Robot Manipulation. *arXiv preprint arXiv:2510.10125*, 2025. URL https://arxiv.org/abs/2510.10125. ICLR 2026.

[38] Tuomas Haarnoja, Ben Moran, Guy Lever, Sandy H. Huang, Dhruva Tirumala, Jan Humplik, Markus Wulfmeier, Saran Tunyasuvunakool, Noah Y. Siegel, Roland Hafner, et al. Learning Agile Soccer Skills for a Bipedal Robot with Deep Reinforcement Learning. *arXiv preprint arXiv:2304.13653*, 2023. URL https://arxiv.org/abs/2304.13653.

[39] Sida He, Lingxi Xie, Yunning Cao, Pengfei Chen, Kaiwen Duan, Jiannan Ge, Xinyue Huo, Jiacheng Shao, and Qi Tian. Robot Manipulation with GPT-6-Astra: Body Knowledge, Experience Reuse, Emergent Skills, and Sim2Real Transfer. *arXiv preprint arXiv:2609.31770*, 2026. URL https://arxiv.org/abs/2609.31770. arXiv 2026.

[40] Tairan He, Jiawei Gao, Wenli Xiao, Yuanhang Zhang, Zi Wang, Jiashun Wang, Zhengyi Luo, Guanqi He, Nikhil Sobanbab, Chaoyi Pan, Zeji Yi, Guannan Qu, Kris Kitani, Jessica Hodgins, Linxi Fan, Yuke Zhu, Changliu Liu, and Guanya Shi. ASAP: Aligning Simulation and Real-World Physics for Learning Agile Humanoid Whole-Body Skills. In *Robotics: Science and Systems (RSS)*, 2025. URL https://arxiv.org/abs/2502.01143.

[41] Hengyuan Hu, Jensen Gao, Priya Sundaresan, Satvik Sharma, Jeannette Bohg, and Dorsa Sadigh. VIA: Visual Interface Agent for Robot Control. *arXiv preprint arXiv:2607.11119*, 2026. URL https://arxiv.org/abs/2607.11119. arXiv 2026.

[42] Jiaheng Hu, Roberto Martin-Martin, Peter Stone, Rocky Duan, Zhenyu Jiang, and Guanya Shi. SimEX: Simulation-Integrated Robotics AutoResearch. *arXiv preprint arXiv:2609.38982*, 2026. URL https://arxiv.org/abs/2609.38982. arXiv 2026.

[43] Kairui Hu, Siyuan Hu, Fangzhou Hong, Zhaoxi Chen, and Ziwei Liu. Embodied Turing Machines: Stateful Code for Robot Recursive Self-Improvement. *arXiv preprint arXiv:2610.12369*, 2026. URL https://arxiv.org/abs/2610.12369. arXiv 2026.

[44] Runhan Huang, Xiangcheng Zhang, Haoyu Chen, Yitong Li, Yen-Jen Wang, Bohan Lyu, Haotian Ye, Chuye Hong, Yixuan Wang, Chloe Su, Haoyu Xiong, Yangzhen Wu, et al. Scaling Online Exploration for Robot Agents. *arXiv preprint arXiv:2610.22496*, 2026. URL https://arxiv.org/abs/2610.22496. arXiv 2026.

[45] Siyuan Huang, Zhengkai Jiang, Hao Dong, Yu Qiao, Peng Gao, and Hongsheng Li. Instruct2Act: Mapping Multi-modality Instructions to Robotic Actions with Large Language Model. *arXiv preprint arXiv:2305.11176*, 2023. URL https://arxiv.org/abs/2305.11176. arXiv 2023.

[46] 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. *arXiv preprint arXiv:2207.05608*, 2022. URL https://arxiv.org/abs/2207.05608. CoRL 2022.

[47] 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. In *Conference on Robot Learning (CoRL)*, 2023.

[48] Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. ReKep: Spatio-Temporal Reasoning of Relational Keypoint Constraints for Robotic Manipulation. *arXiv preprint arXiv:2409.01652*, 2024. URL https://arxiv.org/abs/2409.01652. CoRL 2024.

[49] Yixuan Huang, Bowen Li, Vaibhav Saxena, Yichao Liang, Utkarsh Aashu Mishra, Liang Ji, Lihan Zha, Jimmy Wu, Nishanth Kumar, Sebastian Scherer, Danfei Xu, and Tom Silver. KinDER: A Physical Reasoning Benchmark for Robot Learning and Planning. *arXiv preprint arXiv:2604.25788*, 2026. URL https://arxiv.org/abs/2604.25788. RSS 2026.

[50] Michael Ilie, C. Daniel Freeman, and Kevin K. Troy. Project Fetch: Phase two. https://www.anthropic.com/research/project-fetch-phase-two, 2026.

[51] Yuki Imajuku, Kohki Horie, Yoichi Iwata, Kensho Aoki, Naohiro Takahashi, and Takuya Akiba. ALE-Bench: A Benchmark for Long-Horizon Objective-Driven Algorithm Engineering. *arXiv preprint arXiv:2506.09050*, 2025. URL https://arxiv.org/abs/2506.09050. NeurIPS 2025 Datasets and Benchmarks Track.

[52] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. *arXiv preprint arXiv:2403.07974*, 2024. URL https://arxiv.org/abs/2403.07974. ICLR 2025.

[53] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. RLBench: The Robot Learning Benchmark & Learning Environment. *arXiv preprint arXiv:1909.12271*, 2019. URL https://arxiv.org/abs/1909.12271. IEEE RA-L 2020.

[54] Joel Jang, Seonghyeon Ye, Zongyu Lin, Jiannan Xiang, Johan Bjorck, Yu Fang, Fengyuan Hu, Spencer Huang, et al. DreamGen: Unlocking Generalization in Robot Learning through Video World Models. *arXiv preprint arXiv:2505.12705*, 2025. URL https://arxiv.org/abs/2505.12705.

[55] Jack B. Jedlicki, Amish Sethi, and Heng Yang. Can a Frontier VLM Serve as a Robot Policy for Memory-Dependent Manipulation? *arXiv preprint arXiv:2610.17350*, 2026. URL https://arxiv.org/abs/2610.17350. arXiv 2026.

[56] Mengzhao Jia, Yang Lin, Xixin Zhang, Zhihan Zhang, Xiaobai Liu, and Meng Jiang. Agent as Policy for Robotic Manipulation. *arXiv preprint arXiv:2609.12541*, 2026. URL https://arxiv.org/abs/2609.12541.

[57] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. VIMA: General Robot Manipulation with Multimodal Prompts. *arXiv preprint arXiv:2210.03094*, 2022. URL https://arxiv.org/abs/2210.03094. ICML 2023.

[58] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *arXiv preprint arXiv:2310.06770*, 2023. URL https://arxiv.org/abs/2310.06770. ICLR 2024.

[59] Kushal Kedia, Tyler Ga Wei Lum, Jeannette Bohg, and C. Karen Liu. SimToolReal: An Object-Centric Policy for Zero-Shot Dexterous Tool Manipulation. *arXiv preprint arXiv:2602.16863*, 2026. URL https://arxiv.org/abs/2602.16863. arXiv 2026.

[60] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, Peter David Fagan, Joey Hejna, et al. DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset. *arXiv preprint arXiv:2403.12945*, 2024. URL https://arxiv.org/abs/2403.12945. RSS 2024.

[61] 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. *arXiv preprint arXiv:2406.09246*, 2024. URL https://arxiv.org/abs/2406.09246. CoRL 2024.

[62] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-Tuning Vision-Language-Action Models: Optimizing Speed and Success. *arXiv preprint arXiv:2502.19645*, 2025. URL https://arxiv.org/abs/2502.19645. RSS 2025.

[63] Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, et al. Cosmos Policy: Fine-Tuning Video Models for Visuomotor Control and Planning. *arXiv preprint arXiv:2601.16163*, 2026. URL https://arxiv.org/abs/2601.16163.

[64] Seungyeon Kim, Junhoo Lee, Minkyu Kim, Baekseung Kim, and Nojun Kwak. Recursive Harness Distillation across Agents for Robot Manipulation. *arXiv preprint arXiv:2609.33378*, 2026. URL https://arxiv.org/abs/2609.33378. arXiv 2026.

[65] Yejin Kim, Wilbert Pumacay, Omar Rayyan, Max Argus, Winson Han, Eli VanderBilt, Jordi Salvador, Abhay Deshpande, Rose Hendrix, Snehal Jauhri, Shuo Liu, Nur Muhammad Mahi Shafiullah, Maya Guru, Ainaz Eftekhar, Karen Farley, Donovan Clay, Jiafei Duan, Arjun Guru, Piper Wolters, Alvaro Herrasti, Ying-Chun Lee, Georgia Chalvatzaki, Yuchen Cui, Ali Farhadi, Dieter Fox, and Ranjay Krishna. MolmoSpaces: A Large-Scale Open Ecosystem for Robot Navigation and Manipulation. *arXiv preprint arXiv:2602.11337*, 2026. URL https://arxiv.org/abs/2602.11337. arXiv 2026.

[66] So Kuroki, Kento Kawaharazuka, and Yujin Tang. Learning and Transferring Closed-Loop Robot Software. *arXiv preprint arXiv:2609.19906*, 2026. URL https://arxiv.org/abs/2609.19906. arXiv 2026.

[67] Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Harold Miles, Seraphina Nix, Tao Lin, Chris Painter, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel M. Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring AI Ability to Complete Long Software Tasks. *arXiv preprint arXiv:2503.14499*, 2025. URL https://arxiv.org/abs/2503.14499. arXiv 2025.

[68] Zhiqian Lan, Yuxuan Jiang, Ruiqi Wang, Xuanbing Xie, Rongkui Zhang, Yicheng Zhu, Peihang Li, Tianshuo Yang, Tianxing Chen, Haoyu Gao, Xiaokang Yang, Xuelong Li, Hongyuan Zhang, Yao Mu, and Ping Luo. AutoBio: A Simulation and Benchmark for Robotic Automation in Digital Biology Laboratory. *arXiv preprint arXiv:2505.14030*, 2025. URL https://arxiv.org/abs/2505.14030.

[69] Hojoon Lee, Ajay Subramanian, Ben Abbatematteo, Vijay Veerabadran, Pedro Matias, Karl Ridgeway, and Nitin Kamra. RDA: Reward Design Agent for Reinforcement Learning. *arXiv preprint arXiv:2606.01672*, 2026. URL https://arxiv.org/abs/2606.01672. RLC 2026.

[70] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Wensi Ai, Benjamin Martinez, Hang Yin, Michael Lingelbach, Minjune Hwang, Ayano Hiranaka, Sujay Garlanka, Arman Aydin, Sharon Lee, Jiankai Sun, Mona Anvari, Manasi Sharma, Dhruva Bansal, Samuel Hunter, Kyu-Young Kim, Alan Lou, Caleb R Matthews, Ivan Villa-Renteria, Jerry Huayang Tang, Claire Tang, Fei Xia, Yunzhu Li, Silvio Savarese, Hyowon Gweon, C. Karen Liu, Jiajun Wu, and Li Fei-Fei. BEHAVIOR-1K: A Human-Centered, Embodied AI Benchmark with 1,000 Everyday Activities and Realistic Simulation. *arXiv preprint arXiv:2403.09227*, 2024. URL https://arxiv.org/abs/2403.09227. arXiv 2024.

[71] Long Li, Qichao Zhao, Yue Yang, Fan Xu, Zhe Wang, Alan Wee-Chung Liew, Chao Qu, Heng Tao Shen, and Shirui Pan. Spotter: Let the Embodied Model Lead, and the VLM Reflect for It. *arXiv preprint arXiv:2609.36808*, 2026. URL https://arxiv.org/abs/2609.36808. arXiv 2026.

[72] 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. *arXiv preprint arXiv:2410.07166*, 2024. URL https://arxiv.org/abs/2410.07166. NeurIPS 2024 Datasets and Benchmarks Track.

[73] Rui Li, Zixuan Hu, Wenxi Qu, Jinouwen Zhang, Zhenfei Yin, Sha Zhang, Xuantuo Huang, Hanqing Wang, Tai Wang, Jiangmiao Pang, Wanli Ouyang, Lei Bai, Wangmeng Zuo, Ling-Yu Duan, Dongzhan Zhou, and Shixiang Tang. LabUtopia: High-Fidelity Simulation and Hierarchical Benchmark for Scientific Embodied Agents. *arXiv preprint arXiv:2505.22634*, 2025. URL https://arxiv.org/abs/2505.22634. NeurIPS 2025 Datasets and Benchmarks Track.

[74] Ruiying Li, Yunlang Zhou, YuYao Zhu, Kylin Chen, Jingyuan Wang, Sukai Wang, Kongtao Hu, Minhui Yu, Bowen Jiang, Zhan Su, Jiayao Ma, Xin He, Yongjian Shen, Yang Yang, Guanghui Ren, Maoqing Yao, Wenhao Wang, and Yao Mu. RoboClaw: An Agentic Framework for Scalable

Long-Horizon Robotic Tasks. *arXiv preprint arXiv:2603.11558*, 2026. URL https://arxiv.org/abs/2603.11558.

[75] Shuang Li, Yihuai Gao, Dorsa Sadigh, and Shuran Song. Unified Video Action Model. *arXiv preprint arXiv:2503.00200*, 2025. URL https://arxiv.org/abs/2503.00200. RSS 2025.

[76] Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating Real-World Robot Manipulation Policies in Simulation. *arXiv preprint arXiv:2405.05941*, 2024. URL https://arxiv.org/abs/2405.05941. CoRL 2024.

[77] Zechu Li, Yufeng Jin, Xiaoyang Liu, Puze Liu, Vignesh Prasad, Carlo D'Eramo, and Georgia Chalvatzaki. HARBOR: A Harness Framework for Agentic Robot Reinforcement Learning. *arXiv preprint arXiv:2606.08610*, 2026. URL https://arxiv.org/abs/2606.08610. arXiv 2026.

[78] 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. *arXiv preprint arXiv:2209.07753*, 2022. URL https://arxiv.org/abs/2209.07753. ICRA 2023.

[79] Chunru Lin, Hongxin Zhang, Fenghao Yu, Zhehuan Chen, Thomas L. Griffiths, Yejin Choi, David Held, and Chuang Gan. RoboWits: Unexpected Challenges for Robotic Creative Problem Solving. *arXiv preprint arXiv:2605.30326*, 2026. URL https://arxiv.org/abs/2605.30326. arXiv 2026.

[80] Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. Text2Motion: From Natural Language Instructions to Feasible Plans. *arXiv preprint arXiv:2303.12153*, 2023. URL https://arxiv.org/abs/2303.12153. Autonomous Robots 2023.

[81] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking Knowledge Transfer for Lifelong Robot Learning. *arXiv preprint arXiv:2306.03310*, 2023. URL https://arxiv.org/abs/2306.03310. NeurIPS 2023 Datasets and Benchmarks Track.

[82] Fangchen Liu, Kuan Fang, Pieter Abbeel, and Sergey Levine. MOKA: Open-World Robotic Manipulation through Mark-Based Visual Prompting. *arXiv preprint arXiv:2403.03174*, 2024. URL https://arxiv.org/abs/2403.03174. RSS 2024.

[83] Fangcheng Liu, Yeqing Shen, Anda Cheng, Weishi Mi, Chao Tang, Chenyuan Liu, Yushun Xiang, Tingguang Li, Yong-Lu Li, and Yehui Tang. RoboICL: Embodied In-Context Learning with GPT-6 Astra. *arXiv preprint arXiv:2609.34261*, 2026. URL https://arxiv.org/abs/2609.34261. arXiv 2026.

[84] Renhang Liu, Navonil Majumder, Tej Deep Pala, and Soujanya Poria. RoboQuest: Generalist Physical Agents that Search, Inspect and Test. *arXiv preprint arXiv:2610.10388*, 2026. URL https://arxiv.org/abs/2610.10388. arXiv 2026.

[85] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as Agents. *arXiv preprint arXiv:2308.03688*, 2023. URL https://arxiv.org/abs/2308.03688. ICLR 2024.

[86] Xiao Liu, Tianjie Zhang, Yu Gu, Iat Long Iong, Yifan Xu, Xixuan Song, Shudan Zhang, Hanyu Lai, Xinyi Liu, Hanlin Zhao, Jiadai Sun, Xinyue Yang, Yu Yang, Zehan Qi, Shuntian Yao, Xueqiao Sun, Siyi Cheng, Qinkai Zheng, Hao Yu, Hanchen Zhang, Wenyi Hong, Ming Ding, Lihang Pan, Xiaotao Gu, Aohan Zeng, Zhengxiao Du, Chan Hee Song, Yu Su, Yuxiao Dong, and Jie Tang. VisualAgentBench: Towards Large Multimodal Models as Visual Foundation Agents. *arXiv preprint arXiv:2408.06327*, 2024. URL https://arxiv.org/abs/2408.06327. arXiv 2024.

[87] Yuyao Liu, Jiayuan Mao, David Hsu, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. RAPID: Robot Agentic Programming from Demonstrations. *arXiv preprint arXiv:2609.30249*, 2026. URL https://arxiv.org/abs/2609.30249. arXiv 2026.

[88] Runyu Lu, Yubo Wu, Ethan Kou, Haotian Lin, Letian Fu, Wenli Xiao, Ajay Mandlekar, Yinzhen Xu, Guanya Shi, Ken Goldberg, Ang Chen, Mosharaf Chowdhury, Jan Kautz, Yuke Zhu, Linxi Fan, and Guanzhi Wang. ASPIRE: Agentic /Skills Discovery for Robotics. *arXiv preprint arXiv:2607.00272*, 2026. URL https://arxiv.org/abs/2607.00272.

[89] Zuxing Lu, Hongjia Zhai, Guanzhi Wang, Huajian Zeng, Jiaqi Yang, Jingyu Liu, Lei Cheng, Yuantai Zhang, et al. World-Action Models for Robot Learning and Control: A Survey. *arXiv preprint arXiv:2609.16074*, 2026. URL https://arxiv.org/abs/2609.16074.

[90] 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. In *International Conference on Learning Representations (ICLR)*, 2024.

[91] Yecheng Jason Ma, William Liang, Hung-Ju Wang, Sam Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. DrEureka: Language Model Guided Sim-To-Real Transfer. *arXiv preprint arXiv:2406.01967*, 2024. URL https://arxiv.org/abs/2406.01967. RSS 2024.

[92] Yuntao Ma, Andrei Cramariuc, Farbod Farshidian, and Marco Hutter. Learning Coordinated Badminton Skills for Legged Manipulators. *Science Robotics*, 10(102):eadu3922, 2025. URL https://arxiv.org/abs/2505.22974.

[93] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning. *arXiv preprint arXiv:2108.10470*, 2021. URL https://arxiv.org/abs/2108.10470. NeurIPS 2021 Datasets and Benchmarks Track.

[94] Reginald McLean, Evangelos Chatzaroulas, Luc McCutcheon, Frank Röder, Tianhe Yu, Zhanpeng He, K. R. Zentner, Ryan Julian, J K Terry, Isaac Woungang, Nariman Farsad, and Pablo Samuel Castro. Meta-World+: An Improved, Standardized, RL Benchmark. *arXiv preprint arXiv:2505.11289*, 2025. URL https://arxiv.org/abs/2505.11289. NeurIPS 2025 Datasets and Benchmarks.

[95] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. CALVIN: A Benchmark for Language-Conditioned Policy Learning for Long-Horizon Robot Manipulation Tasks. *arXiv preprint arXiv:2112.03227*, 2021. URL https://arxiv.org/abs/2112.03227. IEEE RA-L 2022.

[96] Matteo Merler, Bowen Li, Josh Roy, Yichao Liang, Qianwei Wang, Yixuan Huang, and Tom Silver. Coding Agents for Generalized Task and Motion Planning Problems. *arXiv preprint arXiv:2609.30233*, 2026. URL https://arxiv.org/abs/2609.30233. arXiv 2026.

[97] Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-Bench: Benchmarking Agents on Hard, Realistic Tasks in Command Line Interfaces. *arXiv preprint arXiv:2601.11868*, 2026. URL https://arxiv.org/abs/2601.11868. ICLR 2026.

[98] Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for General AI Assistants. *arXiv preprint arXiv:2311.12983*, 2023. URL https://arxiv.org/abs/2311.12983. ICLR 2024.

[99] Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. SWE-Lancer: Can Frontier LLMs Earn $1 Million from Real-World Freelance Software Engineering? *arXiv preprint arXiv:2502.12115*, 2025. URL https://arxiv.org/abs/2502.12115. arXiv 2025.

[100] Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, Ajay Mandlekar, Buck Babich, Gavriel State, Marco Hutter, and Animesh Garg. Orbit: A Unified Simulation Framework for Interactive Robot Learning Environments. *arXiv preprint arXiv:2301.04195*, 2023. URL https://arxiv.org/abs/2301.04195. IEEE RA-L 2023.

[101] Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. RoboCasa: Large-Scale Simulation of Everyday Tasks for Generalist Robots. *arXiv preprint arXiv:2406.02523*, 2024. URL https://arxiv.org/abs/2406.02523. RSS 2024.

[102] Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A Large-Scale Simulation Framework for Training and Benchmarking Generalist Robots. *arXiv preprint arXiv:2603.04356*, 2026. URL https://arxiv.org/abs/2603.04356. ICLR 2026.

[103] Chang Nie, Zhe Liu, and Hesheng Wang. Teach and Grow: An Agent-Centered Architecture for General Robot Learning. *arXiv preprint arXiv:2608.17209*, 2026. URL https://arxiv.org/abs/2608.17209. arXiv 2026.

[104] NVIDIA, Niket Agarwal, Arslan Ali, Maciej Bala, Yogesh Balaji, Erik Barker, Tiffany Cai, et al. Cosmos World Foundation Model Platform for Physical AI. *arXiv preprint arXiv:2501.03575*, 2025. URL https://arxiv.org/abs/2501.03575.

[105] NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An Open Foundation Model for Generalist Humanoid Robots. *arXiv preprint arXiv:2503.14734*, 2025. URL https://arxiv.org/abs/2503.14734. arXiv 2025.

[106] NVIDIA, Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, Octi Zhang, Peter Du, Antonio Serrano-Muñoz, Xinjie Yao, René Zurbrügg, Nikita Rudin, Lukasz Wawrzyniak, et al. Isaac Lab: A GPU-Accelerated Simulation Framework for Multi-Modal Robot Learning. *arXiv preprint arXiv:2511.04831*, 2025. URL https://arxiv.org/abs/2511.04831. arXiv 2025.

[107] Michael O'Connell, Guanya Shi, Xichen Shi, Kamyar Azizzadenesheli, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung. Neural-Fly Enables Rapid Learning for Agile Flight in Strong Winds. *Science Robotics*, 7(66):eabm6597, 2022. URL https://arxiv.org/abs/2205.06908.

[108] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An Open-Source Generalist Robot Policy. *arXiv preprint arXiv:2405.12213*, 2024. URL https://arxiv.org/abs/2405.12213. RSS 2024.

[109] Open X-Embodiment Collaboration. Open X-Embodiment: Robotic Learning Datasets and RT-X Models. *arXiv preprint arXiv:2310.08864*, 2023. URL https://arxiv.org/abs/2310.08864. ICRA 2024.

[110] OpenAI. GPT-6 Astra, 2026.

[111] Anne Ouyang, Simon Guo, Simran Arora, Alex L. Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. KernelBench: Can LLMs Write Efficient GPU Kernels? *arXiv preprint arXiv:2502.10517*, 2025. URL https://arxiv.org/abs/2502.10517. ICML 2025.

[112] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. *arXiv preprint arXiv:1710.06537*, 2017. URL https://arxiv.org/abs/1710.06537.

[113] Yiyan Peng, Philip Wang, Simon Sinong Zhan, Yiqi Lyu, Zhenyang Ni, Jixin Yan, Fiorelli Wong, Ruochen Jiao, Hang Yin, Xinyu Cao, Huajie Shao, Manling Li, Ruohan Zhang, and Qi Zhu. MANIGUARD: A Benchmark and Data Suite for Specification-Grounded Safety Evaluation and Improvement of Robotic Manipulation. *arXiv preprint arXiv:2608.17386*, 2026. URL https://arxiv.org/abs/2608.17386. arXiv 2026.

[114] Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, et al. FAST: Efficient Action Tokenization for Vision-Language-Action Models. *arXiv preprint arXiv:2501.09747*, 2025. URL https://arxiv.org/abs/2501.09747.

[115] Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, et al. $\pi^*_{0.6}$: a VLA That Learns From Experience. *arXiv preprint arXiv:2511.14759*, 2025. URL https://arxiv.org/abs/2511.14759.

[116] 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, et al. $\pi_{0.5}$: a Vision-Language-Action Model with Open-World Generalization. *arXiv preprint arXiv:2504.16054*, 2025. URL https://arxiv.org/abs/2504.16054. arXiv 2025.

[117] Xavier Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Tsung-Yen Yang, Ruslan Partsey, Ruta Desai, Alexander William Clegg, Michal Hlavac, So Yeon Min, Vladimír Vondruš, Theophile Gervet, Vincent-Pierre Berges, John M. Turner, Oleksandr Maksymets, Zsolt Kira, Mrinal Kalakrishnan, Jitendra Malik, Devendra Singh Chaplot, Unnat Jain, Dhruv Batra, Akshara Rai, and Roozbeh Mottaghi. Habitat 3.0: A Co-Habitat for Humans, Avatars and Robots. *arXiv preprint arXiv:2310.13724*, 2023. URL https://arxiv.org/abs/2310.13724. ICLR 2024.

[118] Wilbert Pumacay, Ishika Singh, Jiafei Duan, Ranjay Krishna, Jesse Thomason, and Dieter Fox. THE COLOSSEUM: A Benchmark for Evaluating Generalization for Robotic Manipulation. *arXiv preprint arXiv:2402.08191*, 2024. URL https://arxiv.org/abs/2402.08191. RSS 2024.

[119] Chonghan Qin, Dongming Wu, Kaituo Feng, Jiaxu Wang, Xinlei Yu, Peiwen Sun, Yijia Fan, Xilin He, Bingao Chen, Weitong Lian, Ryan Ji, Wenqi Ren, et al. Towards Agentic Robots: Model, Data, Environment, and Harness. *arXiv preprint arXiv:2610.20442*, 2026. URL https://arxiv.org/abs/2610.20442. arXiv 2026.

[120] Julian Quevedo, Ansh Kumar Sharma, Yixiang Sun, Varad Suryavanshi, Percy Liang, and Sherry Yang. WorldGym: World Model as An Environment for Policy Evaluation. *arXiv preprint arXiv:2506.00613*, 2025. URL https://arxiv.org/abs/2506.00613. ICLR 2026.

[121] Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Robot Task Planning. *arXiv preprint arXiv:2307.06135*, 2023. URL https://arxiv.org/abs/2307.06135. CoRL 2023.

[122] Alexander Robey, Zachary Ravichandran, Vijay Kumar, Hamed Hassani, and George J. Pappas. Jailbreaking LLM-Controlled Robots. *arXiv preprint arXiv:2410.13691*, 2024. URL https://arxiv.org/abs/2410.13691.

[123] Peter Schaldenbrand, James McCann, and Jean Oh. FRIDA: A Collaborative Robot Painter with a Differentiable, Real2Sim2Real Planning Environment. *arXiv preprint arXiv:2210.00664*, 2022. URL https://arxiv.org/abs/2210.00664.

[124] SenseNova Joint Research Team, Zhongang Cai, Ziang Cao, Tailai Chen, Han Chong Chua, Linjun Dai, Hanming Deng, Hokin Deng, Haiwen Diao, Chenjing Ding, Weichen Fan, Xiao Fu, et al. Has GPT-6-Astra Achieved Embodied Intelligence? *arXiv preprint arXiv:2610.22471*, 2026. URL https://arxiv.org/abs/2610.22471. arXiv 2026.

[125] Pierre Sermanet, Anirudha Majumdar, Alex Irpan, Dmitry Kalashnikov, and Vikas Sindhwani. Generating Robot Constitutions & Benchmarks for Semantic Safety. *arXiv preprint arXiv:2503.08663*, 2025. URL https://arxiv.org/abs/2503.08663. CoRL 2025.

[126] Carmelo Sferrazza, Dun-Ming Huang, Xingyu Lin, Youngwoon Lee, and Pieter Abbeel. Humanoid-Bench: Simulated Humanoid Benchmark for Whole-Body Locomotion and Manipulation. *arXiv preprint arXiv:2403.10506*, 2024. URL https://arxiv.org/abs/2403.10506. arXiv 2024.

[127] Qiuhong Shen, Shihua Zhang, Yue Liao, Qi Li, Zhenxiong Tan, Shizun Wang, Shuicheng Yan, and Xinchao Wang. World Action Models: A Survey. *arXiv preprint arXiv:2606.20781*, 2026. URL https://arxiv.org/abs/2606.20781.

[128] Zeyu Shen, Haoxiang You, Yilang Liu, Zhicheng Zheng, Lihan Zha, Kashu Yamazaki, Mingtong Zhang, Suning Huang, Jiankai Sun, Qianzhong Chen, Lucy He, Kaiyuan Liu, Haoran Chang, Katerina Fragkiadaki, Dhruv Shah, Mac Schwager, Peter Henderson, Ian Abraham, and Canwen Xu. EmbodiedSWE: Coding Agents for Long Horizon Dexterous Robotics. *arXiv preprint arXiv:2609.27308*, 2026. URL https://arxiv.org/abs/2609.27308. arXiv 2026.

[129] Guanya Shi, Xichen Shi, Michael O'Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung. Neural Lander: Stable Drone Landing Control Using Learned Dynamics. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 9784–9790, 2019. URL https://arxiv.org/abs/1811.08027.

[130] Lucy Xiaoyang Shi, Brian Ichter, Michael Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, et al. Hi Robot: Open-Ended Instruction Following with Hierarchical Vision-Language-Action Models. *arXiv preprint arXiv:2502.19417*, 2025. URL https://arxiv.org/abs/2502.19417. ICML 2025.

[131] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. *arXiv preprint arXiv:2303.11366*, 2023. URL https://arxiv.org/abs/2303.11366. NeurIPS 2023.

[132] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. *arXiv preprint arXiv:1912.01734*, 2019. URL https://arxiv.org/abs/1912.01734. CVPR 2020.

[133] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. *arXiv preprint arXiv:2010.03768*, 2020. URL https://arxiv.org/abs/2010.03768. ICLR 2021.

[134] Ruiyang Si, Jianxin Bi, Shunyu Yang, Rui Ni, Wenbo Huang, Qiang Wang, Shulong Jiang, Yintianrun Zhang, Feixiang Ruan, Kewei Zhang, Duomin Wang, Xiuyu Li, Haiwen Feng, Zhen Dong, and Daquan Zhou. Fewer Tokens, Better Action: GPT-6 Astra Robot Agents with 14% Higher Success Rate but 65% Fewer Tokens. *arXiv preprint arXiv:2610.01939*, 2026. URL https://arxiv.org/abs/2610.01939. arXiv 2026.

[135] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating Situated Robot Task Plans using Large Language Models. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[136] Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, C. Karen Liu, Silvio Savarese, Hyowon Gweon, Jiajun Wu, and Li Fei-Fei. BEHAVIOR: Benchmark for Everyday Household Activities in Virtual, Interactive, and Ecological Environments. *arXiv preprint arXiv:2108.03332*, 2021. URL https://arxiv.org/abs/2108.03332. CoRL 2021.

[137] Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI's Ability to Replicate AI Research. *arXiv preprint arXiv:2504.01848*, 2025. URL https://arxiv.org/abs/2504.01848. arXiv 2025.

[138] Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training Home Assistants to Rearrange their Habitat. *arXiv preprint arXiv:2106.14405*, 2021. URL https://arxiv.org/abs/2106.14405. NeurIPS 2021.

[139] Xiao Tan, Yixuan Wang, Zhexi Wang, and Wenzhen Yuan. FORCA: A Force-Aware Coding Agent for Contact-Rich Robot Manipulation. *arXiv preprint arXiv:2610.31841*, 2026. URL https://arxiv.org/abs/2610.31841. arXiv 2026.

[140] Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse kai Chan, Yuan Gao, Xuanlin Li, Tongzhou Mu, Nan Xiao, Arnav Gurha, Viswesh Nagaswamy Rajesh, Yong Woo Choi, Yen-Ru Chen, Zhiao Huang, Roberto Calandra, Rui Chen, Shan Luo, and Hao Su. ManiSkill3: GPU Parallelized Robotics Simulation and Rendering for Generalizable Embodied AI. *arXiv preprint arXiv:2410.00425*, 2024. URL https://arxiv.org/abs/2410.00425. arXiv 2024.

[141] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind Control Suite. *arXiv preprint arXiv:1801.00690*, 2018. URL https://arxiv.org/abs/1801.00690. arXiv 2018.

[142] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. *arXiv preprint arXiv:1703.06907*, 2017. URL https://arxiv.org/abs/1703.06907.

[143] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, 2012.

[144] Brian Y. Tsui, Alan Y. Fang, and Tiffany J. Hwu. Demonstration-Free Robotic Control via LLM Agents. *arXiv preprint arXiv:2601.20334*, 2026. URL https://arxiv.org/abs/2601.20334. IROS 2026.

[145] Sai Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. ChatGPT for Robotics: Design Principles and Model Abilities. *arXiv preprint arXiv:2306.17582*, 2023. URL https://arxiv.org/abs/2306.17582. arXiv 2023.

[146] Vainavi Viswanath, Kaushik Shivakumar, Justin Kerr, Brijen Thananjeyan, Ellen Novoseller, Jeffrey Ichnowski, Alejandro Escontrela, Michael Laskey, Joseph E. Gonzalez, and Ken Goldberg. Autonomously Untangling Long Cables. In *Robotics: Science and Systems (RSS)*, 2022. URL https://arxiv.org/abs/2207.07813.

[147] Chengcheng Wang, Yuheng Shi, Siyu Xu, Yutong Wang, Jinjing Zhao, Chenghao Xia, Jinqi Liao, Jiankun Peng, Ligong Bi, Yuechuan Ma, Jiaqi Liu, Wenbo Wang, Zijian Wang, Yuemin Wu, Minjing Dong, Xiu Su, Tao Huang, Jianyuan Guo, Chengbin Du, Yunke Wang, and Chang Xu. Auriga Technical Report: A Task-Agnostic Agent Harness for Zero-Shot Embodied Manipulation. *arXiv preprint arXiv:2611.01112*, 2026. URL https://arxiv.org/abs/2611.01112. arXiv 2026.

[148] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv preprint arXiv:2305.16291*, 2023.

[149] Hanwen Wang, Weizhi Zhao, Xiangyu Wang, Siyuan Huang, He Lin, Boyuan Zheng, Rongtao Xu, Gang Wang, Yao Mu, He Wang, Lue Fan, Hongsheng Li, Zhaoxiang Zhang, and Tieniu Tan. DexJoCo: A Benchmark and Toolkit for Task-Oriented Dexterous Manipulation on MuJoCo. *arXiv preprint arXiv:2605.16257*, 2026. URL https://arxiv.org/abs/2605.16257. arXiv 2026.

[150] Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. GenSim: Generating Robotic Simulation Tasks via Large Language Models. *arXiv preprint arXiv:2310.01361*, 2023. URL https://arxiv.org/abs/2310.01361. ICLR 2024.

[151] Qi Wang, Tianyi Wang, Chengyang Li, Shikun Ban, Yurun Chen, Yizhong Ge, Jason Qin, Chengtai Li, and Wentao Zhu. Towards the Harness of Embodied Agents. *arXiv preprint arXiv:2608.11246*, 2026. URL https://arxiv.org/abs/2608.11246. arXiv 2026.

[152] Ruocheng Wang, Yunzhi Zhang, Jiayuan Mao, Ran Zhang, Chin-Yi Cheng, and Jiajun Wu. IKEA-Manual: Seeing Shape Assembly Step by Step. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2022. URL https://arxiv.org/abs/2302.01881.

[153] Siyin Wang, Junhao Shi, Zhaoyang Fu, Xinzhe He, Feihong Liu, Chenchen Yang, Yikang Zhou, Zhaoye Fei, Jingjing Gong, Jinlan Fu, Mike Zheng Shou, Xuanjing Huang, Xipeng Qiu, and Yu-Gang Jiang. World Action Models: The Next Frontier in Embodied AI. *arXiv preprint arXiv:2605.12090*, 2026. URL https://arxiv.org/abs/2605.12090.

[154] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. *arXiv preprint arXiv:2407.16741*, 2024. URL https://arxiv.org/abs/2407.16741. ICLR 2025.

[155] Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. RoboGen: Towards Unleashing Infinite Data for Automated Robot Learning via Generative Simulation. *arXiv preprint arXiv:2311.01455*, 2023. URL https://arxiv.org/abs/2311.01455. ICML 2024.

[156] Yutong Wang, Dongjae Lee, Xiaofeng Guo, Yuanzhu Zhan, Yufei Jiang, Bavin Saravanan, Muqing Cao, Jia Xie, Chenyang Mao, Sebastian Scherer, Junyi Geng, and Guanya Shi. AM-Bench: A Modular Simulation Suite and Benchmark for Aerial Manipulation Policy Learning. *arXiv preprint arXiv:2609.00641*, 2026. URL https://arxiv.org/abs/2609.00641. arXiv 2026.

[157] Ziyong Wang, Yupei Lin, Runxuan Huang, Liang Lin, Dongyu Zhang, and Guangrun Wang. GPT-6-Astra Understands Touch and Outperform Vision–Tactile–Language–Action Models. *arXiv preprint arXiv:2610.28457*, 2026. URL https://arxiv.org/abs/2610.28457. arXiv 2026.

[158] Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Holden Karnofsky, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. RE-Bench: Evaluating frontier AI R&D capabilities of language model agents against human experts. *arXiv preprint arXiv:2411.15114*, 2024. URL https://arxiv.org/abs/2411.15114. arXiv 2024.

[159] Wenli Xiao, Haoru Xue, Tony Tao, Dvij Kalaria, John M. Dolan, and Guanya Shi. AnyCar to Anywhere: Learning Universal Dynamics Model for Agile and Adaptive Mobility. *arXiv preprint arXiv:2409.15783*, 2024. URL https://arxiv.org/abs/2409.15783.

[160] Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian "Max" Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, Jimmy Wu, Guanzhi Wang, S. Shankar Sastry, Ken Goldberg, Linxi "Jim" Fan, Yuke Zhu, and Guanya Shi. ENPIRE: Agentic Robot Policy Self-Improvement in the Real World. *arXiv preprint arXiv:2606.19980*, 2026. URL https://arxiv.org/abs/2606.19980. CoRL 2026.

[161] Shuangyu Xie, Kaiyuan Chen, and Ken Goldberg. Revisiting the "Push-T" Robot Manipulation Task with Agentic Robotics. *arXiv preprint arXiv:2608.18227*, 2026. URL https://arxiv.org/abs/2608.18227. arXiv 2026.

[162] 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. *arXiv preprint arXiv:2309.11489*, 2023. URL https://arxiv.org/abs/2309.11489. ICLR 2024.

[163] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024.

[164] Bingxin Xu, Yuzhang Shang, Zhen Dong, and Emilio Ferrara. Coding Agents with Harness for Safe Robot Control. *arXiv preprint arXiv:2609.20822*, 2026. URL https://arxiv.org/abs/2609.20822. arXiv 2026.

[165] Mengdi Xu, Peide Huang, Wenhao Yu, Shiqi Liu, Xilun Zhang, Yaru Niu, Tingnan Zhang, Fei Xia, Jie Tan, and Ding Zhao. Creative Robot Tool Use with Large Language Models. *arXiv preprint arXiv:2310.13065*, 2023. URL https://arxiv.org/abs/2310.13065. arXiv 2023.

[166] Jenai Xuning Yang, Rishit Dagli, Alex Zook, Hugo Hadfield, Ankit Goyal, Stan Birchfield, Fabio Ramos, and Jonathan Tremblay. RoboLab: A High-Fidelity Simulation Benchmark for Analysis of Task Generalist Policies. *arXiv preprint arXiv:2604.09860*, 2026. URL https://arxiv.org/abs/2604.09860. arXiv 2026.

[167] Jihan Yang, Shusheng Yang, Anjali W. Gupta, Rilyn Han, Li Fei-Fei, and Saining Xie. Thinking in Space: How Multimodal Large Language Models See, Remember, and Recall Spaces. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. URL https://arxiv.org/abs/2412.14171.

[168] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. *arXiv preprint arXiv:2405.15793*, 2024. URL https://arxiv.org/abs/2405.15793. NeurIPS 2024.

[169] John Yang, Carlos E. Jimenez, Alex L. Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R. Narasimhan, Diyi Yang, Sida I. Wang, and Ofir Press. SWE-bench Multimodal: Do AI Systems Generalize to Visual Software Domains? *arXiv preprint arXiv:2410.03859*, 2024. URL https://arxiv.org/abs/2410.03859. ICLR 2025.

[170] 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. *arXiv preprint arXiv:2502.09560*, 2025. URL https://arxiv.org/abs/2502.09560. ICML 2025.

[171] Sherry Yang, Yilun Du, Kamyar Ghasemipour, Jonathan Tompson, Leslie Kaelbling, Dale Schuurmans, and Pieter Abbeel. Learning Interactive Real-World Simulators. *arXiv preprint arXiv:2310.06114*, 2023. URL https://arxiv.org/abs/2310.06114. ICLR 2024.

[172] Zhiqin Yang, Chenxin Li, Xiaomeng Hu, Yibin Liu, Weidong Huang, Jiankai Sun, Haitao Li, Zijian Wu, Yuzhi Huang, Fanding Huang, Hanwen Sun, Jiashun Liu, et al. RobotWorld: Benchmarking Multimodal Agents for Robot Use Across Diverse Tasks and Embodiments. *arXiv preprint arXiv:2610.10409*, 2026. URL https://arxiv.org/abs/2610.10409. arXiv 2026.

[173] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv preprint arXiv:2210.03629*, 2022. URL https://arxiv.org/abs/2210.03629. ICLR 2023.

[174] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. *arXiv preprint arXiv:2406.12045*, 2024. URL https://arxiv.org/abs/2406.12045. arXiv 2024.

[175] Yifei Yao, Yude Zou, Guanbo Jiao, Xing Gao, Ning Gao, Yukai Wang, Jiantong Chen, Zihou Zhu, Xinyu Li, Ningyu Yan, Shuai Wang, Jianming Ma, et al. Profiling a Frontier Agent with EBench: From Direct Control to Harness with Policy. *arXiv preprint arXiv:2610.35832*, 2026. URL https://arxiv.org/abs/2610.35832. arXiv 2026.

[176] Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, Ayaan Malik, Kyungmin Lee, William Liang, Nadun Ranawaka, Jiasheng Gu, Yinzhen Xu, Guanzhi Wang, Fengyuan Hu, Avnish Narayan, Johan Bjorck, Jing Wang, Gwanghyun Kim, Dantong Niu, Ruijie Zheng, Yuqi Xie, Jimmy Wu, Qi Wang, Ryan Julian, Danfei Xu, Yilun Du, Yevgen Chebotar, Scott Reed, Jan Kautz, Yuke Zhu, Linxi Fan, and Joel Jang. World Action Models are Zero-shot Policies. *arXiv preprint arXiv:2602.15922*, 2026. URL https://arxiv.org/abs/2602.15922.

[177] Sheng Yin, Xianghe Pang, Yuanzhuo Ding, Menglan Chen, Yutong Bi, Yichen Xiong, Wenhao Huang, Zhen Xiang, et al. SafeAgentBench: A Benchmark for Safe Task Planning of Embodied LLM Agents. *arXiv preprint arXiv:2412.13178*, 2024. URL https://arxiv.org/abs/2412.13178.

[178] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning. *arXiv preprint arXiv:1910.10897*, 2019. URL https://arxiv.org/abs/1910.10897. CoRL 2019.

[179] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao, Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei Xia. Language to Rewards for Robotic Skill Synthesis. *arXiv preprint arXiv:2306.08647*, 2023. URL https://arxiv.org/abs/2306.08647. CoRL 2023.

[180] Jiarui Yuan, Yuxin Zuo, Xiaoyu Huang, Wenbo Zhang, Kwanyoung Park, Junlin Yang, Tianxiang Zhang, Ruoqu Chen, Yutao Ouyang, Bingxiang He, Xingtai Lv, Hongjin Wu, et al. Democratizing Robot-Use Agents: A Unified Harness with Open Models. *arXiv preprint arXiv:2611.01560*, 2026. URL https://arxiv.org/abs/2611.01560. arXiv 2026.

[181] Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A. Kahrs, Carmelo Sferrazza, Yuval Tassa, and Pieter Abbeel. MuJoCo Playground. *arXiv preprint arXiv:2502.08844*, 2025. URL https://arxiv.org/abs/2502.08844. arXiv 2025.

[182] Michał Zawalski, William Chen, Karl Pertsch, Oier Mees, Chelsea Finn, and Sergey Levine. Robotic Control via Embodied Chain-of-Thought Reasoning. *arXiv preprint arXiv:2407.08693*, 2024. URL https://arxiv.org/abs/2407.08693.

[183] Andy K. Zhang, Neil Perry, Riya Dulepet, Joey Ji, Celeste Menders, Justin W. Lin, Eliot Jones, Gashon Hussein, Samantha Liu, Donovan Jasper, Pura Peetathawatchai, Ari Glenn, Vikram Sivashankar, Daniel Zamoshchin, Leo Glikbarg, Derek Askaryar, Mike Yang, Teddy Zhang, Rishi Alluri, Nathan Tran, Rinnara Sangpisit, Polycarpos Yiorkadjis, Kenny Osele, Gautham Raghupathi,

Dan Boneh, Daniel E. Ho, and Percy Liang. Cybench: A Framework for Evaluating Cybersecurity Capabilities and Risks of Language Models. *arXiv preprint arXiv:2408.08926*, 2024. URL https://arxiv.org/abs/2408.08926. ICLR 2025.

[184] 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. *arXiv preprint arXiv:2412.18194*, 2024. URL https://arxiv.org/abs/2412.18194. arXiv 2024.

[185] Wenbo Zhang, Kaixuan Wang, Yutao Ouyang, Xiaoyu Huang, Liyang Li, Kailun Su, Weiyang Jin, Wenhao Chai, Haotian Liang, Zhiyang Dou, Yue Chen, and Tianxing Chen. An Unexpected Robot Policy: Early Evaluations of GPT-6 Astra on RoboDojo and Beyond. *arXiv preprint arXiv:2609.24170*, 2026. URL https://arxiv.org/abs/2609.24170. arXiv 2026.

[186] Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Yi Nie, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, Jiaji Rao, Zhengru Fang, Changxu Wei, Yu Wang, Wenbo Ding, and Chao Yu. Harness VLA: Steering Frozen VLAs into Reliable Manipulation Primitives via Memory-Guided Agents. *arXiv preprint arXiv:2607.08448*, 2026. URL https://arxiv.org/abs/2607.08448. arXiv 2026.

[187] Yunbei Zhang, Zijian Jin, Yuanzhe Liu, Janet Wang, Xilun Zhang, Yuyou Zhang, Zhenyu Zhang, Daoan Zhang, Shuaicheng Niu, Gen Li, Jianfei Yang, Jihun Hamm, Ismini Lourentzou, Weirui Ye, Bo Liu, Peter Stone, and Marco Pavone. LIBERO-MAX: Do Robot Policies Adapt When the World Changes? *arXiv preprint arXiv:2609.36518*, 2026. URL https://arxiv.org/abs/2609.36518. arXiv 2026.

[188] Zexi Zhang, Zecheng Zhu, Zidong Chen, Zulkhuu Tuya, and Stephen James. BiGym 2.0: Benchmarking Learned and Agent-Developed Policies for Humanoid Household Manipulation. *arXiv preprint arXiv:2610.07594*, 2026. URL https://arxiv.org/abs/2610.07594. arXiv 2026.

[189] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. *arXiv preprint arXiv:2304.13705*, 2023. URL https://arxiv.org/abs/2304.13705. RSS 2023.

[190] Ziqian Zhong, Aditi Raghunathan, and Nicholas Carlini. ImpossibleBench: Measuring LLMs' Propensity of Exploiting Test Cases. *arXiv preprint arXiv:2510.20270*, 2025. URL https://arxiv.org/abs/2510.20270. arXiv 2025.

[191] Enshen Zhou, Qi Su, Cheng Chi, Zhizheng Zhang, Zhongyuan Wang, Tiejun Huang, Lu Sheng, and He Wang. Code-as-Monitor: Constraint-aware Visual Programming for Reactive and Proactive Robotic Failure Detection. *arXiv preprint arXiv:2412.04455*, 2024. URL https://arxiv.org/abs/2412.04455. CVPR 2025.

[192] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A Realistic Web Environment for Building Autonomous Agents. *arXiv preprint arXiv:2307.13854*, 2023. URL https://arxiv.org/abs/2307.13854. ICLR 2024.

[193] Youwei Zhou, Bohan Hou, Yankai Fu, Yaoxu Lyu, Yifan Ye, Shilin Shan, Chen Guo, Jiaming Liu, Ran He, Caifeng Shan, Shanghang Zhang, Jianfei Yang, and Zhihe Lu. Large Multimodal Models for Direct Robot Control: GPT-6 as a Beginning. *arXiv preprint arXiv:2610.28259*, 2026. URL https://arxiv.org/abs/2610.28259. arXiv 2026.

[194] Zhiyuan Zhou, Pranav Atreya, Abraham Lee, Homer Walke, Oier Mees, and Sergey Levine. Autonomous Improvement of Instruction Following Skills via Foundation Models. *arXiv preprint arXiv:2407.20635*, 2024. URL https://arxiv.org/abs/2407.20635. CoRL 2024.

[195] Zhiyuan Zhou, Pranav Atreya, You Liang Tan, Karl Pertsch, and Sergey Levine. AutoEval: Autonomous Evaluation of Generalist Robot Manipulation Policies in the Real World. *arXiv preprint arXiv:2503.24278*, 2025. URL https://arxiv.org/abs/2503.24278. arXiv 2025.

[196] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Kevin Lin, Abhiram Maddukuri, Soroush Nasiriany, and Yifeng Zhu. robosuite: A Modular Simulation Framework and Benchmark for Robot Learning. *arXiv preprint arXiv:2009.12293*, 2020. URL https://arxiv.org/abs/2009.12293. arXiv 2020.

[197] Yuxuan Zhu, Tengjun Jin, Yada Pruksachatkun, Andy Zhang, Shu Liu, Sasha Cui, Sayash Kapoor, Shayne Longpre, Kevin Meng, Rebecca Weiss, Fazl Barez, Rahul Gupta, Jwala Dhamala, Jacob Merizian, Mario Giulianelli, Harry Coppock, Cozmin Ududec, Jasjeet Sekhon, Jacob Steinhardt, Antony Kellermann, Sarah Schwettmann, Matei Zaharia, Ion Stoica, Percy Liang, and Daniel Kang. Establishing Best Practices for Building Rigorous Agentic Benchmarks. *arXiv preprint arXiv:2507.02825*, 2025. URL https://arxiv.org/abs/2507.02825. arXiv 2025.

Appendix

Contents

A Suites and Tasks . . . **36**
A.1 New Suites . . . 36
A.2 Adapted Suites . . . 37

B Protocol Details . . . **37**

C Interface, Grading and Physics . . . **39**
C.1 Interface . . . 39
C.2 Grading . . . 41
C.3 Physics . . . 42

D Additional Results . . . **43**

E Further Analysis . . . **46**
E.1 Causes of the Drop from the `Privileged` to the `Standard` Setting . . . 46
E.2 Where Agents Fail . . . 47
E.3 Differences between Agents . . . 48
E.4 Control Approaches . . . 51

F Additional Attempts . . . **53**

G Controlled Runs . . . **56**

H Failure Labels . . . **58**
H.1 Labeling Procedure . . . 58
H.2 Taxonomy . . . 60
H.3 Excluded Attempts . . . 60
H.4 Failures by Suite, Robot Body, Agent and Setting . . . 63
H.5 Further Statistics . . . 66
H.6 Case Studies . . . 75

I Real-Robot Demonstration . . . **80**

J Limitations . . . **82**

K Extended Related Work . . . **82**

A. Suites and Tasks

What a task contains. We package every task in the Harbor format with a fixed directory structure. The file `instruction.md` gives the agent its prompt: the goal, the submission requirements, the limits, and every convention the agent cannot infer, including coordinate frames, units, the action layout, how to use the robot client, and the reset budget. Any goal specified as an image, video, or other file is placed inside the agent's container and named in `instruction.md`. We store the task configuration in `task.toml`, which specifies the time limit, the CPU, memory, and disk caps of the agent's container, the verifier settings, and task metadata (the simulator, the robot, and whether a reference solution exists). The `environment/` directory defines the runtime via a `Dockerfile` for the agent's workspace container and a `docker-compose.yaml` that adds the remaining containers (the simulator service in `Standard` and `Realtime`, and a resource monitor) along with their resource caps. Within it, `environment/instance.json` defines the frozen scene (the objects, the robot, and their initial state), which the simulator checks against a stored hash at every build and reset so every episode starts from the same state. The `tests/` directory contains the verifier: its own `Dockerfile` and `docker-compose.yaml`, the entry script `test.sh`, and the grading code with the success rule, all running in an isolated container that the agent never sees. When a task has a reference solution, we include `solution/solve.sh`, which validates the task and is never shown to the agent, and we provide `README.md` to describe the task for human readers. Finally, we pin every base image to a fixed tag or digest so every run of a task uses the exact same software.

Table 1 gives the size of the exam, and Table 2 lists its 35 suites, of which 9 were built for the exam and 26 are adapted from existing benchmarks, whose scenes, robots and success rules we keep. The Robots column counts distinct robot models in the suite. The same robot can appear in several suites, so the total counts each of the 47 robots once. A test is one task in one setting, and the Core column counts the tasks in the core set. Time limits range from 30 minutes to four hours by suite. Of the 1,156 tasks, 590 allow one hour and 351 allow two hours or more.

A.1. New Suites

RoboARC, DexHandExam and RoboPaint require contact-rich manipulation of deformable objects, multi-part assemblies and tools. In RoboARC, single- and dual-arm robots untie knots, thread ropes through holes, disentangle rings from wire puzzles and assemble IKEA furniture. Earlier work studied such tasks with dedicated systems for untangling cables [146] and with datasets of furniture assembly manuals [152]. In DexHandExam, dexterous hands turn book pages, open jars and insert a USB stick, and objects move only through contact and gravity. In RoboPaint, a robot arm reproduces a painting or a calligraphy character with a brush or a pen, so the task tests fine tool control. Dedicated robot painters such as FRIDA [123] plan their strokes in a differentiable simulation of the painting process.

RoboDynamicExam and the two Embodied Olympics suites involve dynamic objects and opponents, so the agent has to act under tight timing constraints. These three suites contain 53 of the 86 tasks that have a `Realtime` test. RoboDynamicExam asks single- and dual-arm robots to catch, strike and hand over moving objects such as tossed balls and sliding pucks. In Embodied Olympics - Fun, a humanoid performs athletic skills from 79 sports, such as dribbling a ball through a slalom, and in Embodied Olympics - Match, robots play full matches of table tennis, badminton, soccer, foosball or fencing against scripted or learned opponents. Robots have learned to play table tennis [21], badminton [92] and soccer [38] with reinforcement learning and task-specific training.

Table 1 | Size of the exam. Rows list the number of suites (with the number built for the exam in parentheses), robots, tasks, tests, tasks with a full or partial reference solution, core tasks and core tests.

Item	Count
Suites (new)	35 (9)
Robots	47
Tasks	1,156
Tests	2,288
Tasks with a full / partial reference solution	686 / 78
Core tasks	456
Core tests	901

The remaining three suites test abilities beyond executing a stated goal. In iSpatialGym, a mobile bimanual robot receives the goal only as an image of what its head camera should show when the task is done, and it must rearrange the scene or reposition itself until its camera view matches this goal image. Multimodal models have been evaluated on spatial reasoning by answering questions about videos of indoor scenes [167], and iSpatialGym requires the agent to act on such reasoning. Embodied Safety Exam pairs hazardous requests, whose literal execution would cause harm, with benign requests of the same kind, so the agent must refuse the hazardous requests and complete the benign ones, even when the hazard is hidden or emerges during the task. Its verdict is decided from what physically happens in the simulation. RoboEngineer asks the agent to tune a controller on a nominal simulator that it can access, deploy the controller to an evaluation simulator with undisclosed dynamics and refine it from the deployment feedback. This loop resembles sim-to-real transfer, which prior work addresses with domain and dynamics randomization [112, 142] and by aligning simulated with real physics [40]. RoboEngineer's drone and racing-car tasks build on the learned dynamics models of Neural Lander, Neural-Fly and AnyCar [107, 129, 159].

A.2. Adapted Suites

The other 26 suites come from existing benchmarks, and we group them by their main embodiment, as Figure 5 (right) also does. The single-arm suites are LIBERO-MAX [187], RoboLab [166], MetaWorld+ [94], VLABench [184], ManiGuard [113], EmbodiedSWE [128], LabUtopia [73], MolmoSpaces [65] and AutoBio [68], and they range from language-conditioned tabletop manipulation to laboratory automation. The bimanual suites are RoboDojo [14], RoboTwin 2.0 [13], ABC Sim, RoboWits [79] and YAMLab [5], and the mobile manipulation suites are BEHAVIOR-1K [70], RoboCasa365 [102], RoboCasa [101] and KinDER [49], which include long-horizon household activities in house-scale scenes. RoboCasa-GR1 [105], BiGym 2.0 [188] and HumanoidBench [126] cover humanoid manipulation and whole-body control. DexToolBench [59] and DexJoCo [149] cover dexterous manipulation and tool use. MyoSuite [10] covers musculoskeletal control, MuJoCo Playground [181] covers quadruped locomotion together with a few arm tasks, and AM-Bench [156] covers aerial manipulation.

B. Protocol Details

Budgets. Each attempt runs in a container with 2 to 16 CPU cores and 4 to 39 GiB of memory, with a GPU where the simulator or the task needs one, and with a wall-clock limit of 30 minutes to four hours, depending on the suite. The harness can reach only the API of its model and

Table 2 | The suites of the exam. The top block lists the 9 suites that we built, and the bottom block lists the 26 suites adapted from existing benchmarks, which Section A.2 cites. The Kind column gives the kinds of robot body in the suite (mobile manip. = mobile manipulator). The Robots column counts distinct robot models, the Tests column counts tests over all tasks, and the Core column counts tasks in the core set.

Suite	Simulation backend	Kind	Robots	Tasks	Tests	Core
DexHandExam	MuJoCo	hand	3	30	60	23
Embodied Olympics - Fun	MuJoCo	humanoid	2	79	179	56
Embodied Olympics - Match	MuJoCo	arm, humanoid, mobile manip.	5	14	41	13
Embodied Safety Exam	MuJoCo	arm, two arms, mobile manip.	4	72	76	50
iSpatialGym	OmniGibson (Isaac Sim)	mobile manip.	1	12	24	8
RoboARC	MuJoCo	arm, two arms	3	106	212	58
RoboDynamicExam	ManiSkill / SAPIEN	arm, two arms	2	25	69	10
RoboEngineer	MuJoCo + learned dynamics	aerial, humanoid, mobile manip., quadruped	5	30	30	15
RoboPaint	ManiSkill / SAPIEN	arm	1	43	86	22
ABC Sim	MuJoCo	two arms	1	29	60	0
AM-Bench	Isaac Lab / Isaac Sim	aerial	1	9	17	2
AutoBio	MuJoCo	arm, two arms	4	9	18	0
BEHAVIOR-1K	OmniGibson (Isaac Sim)	mobile manip.	1	100	200	71
BiGym 2.0	MuJoCo	humanoid	1	13	39	10
DexJoCo	MuJoCo	hand	2	11	22	2
DexToolBench	MuJoCo	hand	1	21	42	20
EmbodiedSWE	Isaac Lab / Isaac Sim	arm, two arms, humanoid	3	24	48	12
HumanoidBench	MuJoCo	humanoid	2	10	30	8
KinDER	MuJoCo	mobile manip.	1	5	10	4
LabUtopia	Isaac Sim	arm	1	9	18	0
LIBERO-MAX	robosuite / MuJoCo	arm	1	64	128	2
ManiGuard	OmniGibson (Isaac Sim)	arm	1	30	60	7
MetaWorld+	MuJoCo	arm	1	50	100	2
MolmoSpaces	MuJoCo	arm, mobile manip.	2	8	16	2
MuJoCo Playground	MuJoCo	arm, two arms, quadruped	5	13	27	7
MyoSuite	MuJoCo	musculoskeletal	6	22	45	13
RoboCasa	robosuite / MuJoCo	mobile manip.	1	29	58	2
RoboCasa-GR1	robosuite / MuJoCo	humanoid	1	24	48	1
RoboCasa365	robosuite / MuJoCo	mobile manip.	1	50	101	5
RoboDojo	Isaac Lab / Isaac Sim	two arms	1	54	102	12
RoboLab	Isaac Lab / Isaac Sim	arm	1	53	106	6
RoboTwin 2.0	SAPIEN	two arms	1	47	94	3
RoboWits	Genesis	two arms	1	15	30	6
VLABench	MuJoCo (dm_control)	arm	1	36	72	2
YAMLab	Isaac Lab / Isaac Sim	two arms	1	10	20	2
Total			47	1,156	2,288	456

the package registries, except in Embodied Olympics - Fun, which runs with open network access. The harness's web search is off in every task. A proxy between the harness and the model provider forwards only the requests to the model's API.

Reference solutions and integrity checks. Every task must fail when the robot does nothing and pass with its reference solution wherever one exists (686 tasks have a full reference solution, and 78 have a partial one). Tasks without a reference solution rely on the original success rule and a human review. Reference solutions never enter the agent's container, and the solutions of the original benchmarks are removed from the container images. Before a suite is added, an agent is instructed to pass its tasks without solving them. If the agent succeeds on any task, the suite is not added until its verifier is fixed.

Leaderboard rules. Each agent makes one attempt per test. When an attempt is rerun after an infrastructure failure, the agent's latest finished attempt counts. A test counts as solved when the verdict of its attempt is pass, and a test without a verdict counts as failed. The Core view counts each of the 456 core tasks once, in its default setting (`Standard` for 403 tasks and `Realtime` for 53). Each per-setting count includes all core tests of that setting. When we pool the three strongest agents, as in Sections 3.1 and D, each test counts once per agent, and rates are reported per attempt.

Approach labels. For Section 3.5, an agent read the logs of a random sample of 1,489 attempts (about a third of all attempts, snapshot of 18 October 2026) and assigned each attempt the approach that produced its graded answer. *Code to control* covers three kinds of program. An open-loop program computes the whole motion in advance, as waypoints, key frames, inverse-kinematics solutions or a recorded sequence, and runs it without feedback. A closed-loop program reads the state or the cameras at every control step or action chunk and adjusts its commands. A parameter search finds a passing motion, or its parameters, by trying many episodes and resets or by running an optimizer such as the cross-entropy method (CEM), CMA-ES, or a grid or random search. In *Direct control*, the agent sends one action chunk at a time and decides on the next one after looking at images or readings. In *Train a policy*, the agent first trains a control policy, such as a neural network or a reinforcement-learning policy, which then controls the robot. Attempts that switched approach midway or used two approaches equally are labeled mixed, and attempts that only explored or never ran anything are labeled as having no control plan. These labels have not yet been checked manually. Figure 2 shows an example of each approach.

C. Interface, Grading and Physics

This section expands Section 2.1 with the protocol rules and each suite's values and examples.

C.1. Interface

`Privileged` Setting: a simulator library in the agent's container. The agent's container holds the simulator of the suite and a small Python library that the exam provides, which we call the tooling. The tooling rebuilds the frozen scene, resets it, advances it by applying actions, reads the state of every object and records each applied action as one row of an array. The agent may also call the planning primitives of the original benchmark, such as the grasp and place primitives of RoboTwin 2.0 with its CuRobo planner, or write its own planner, optimizer or learned controller.

Whatever the agent calls, the tooling records the low-level action row applied at every control step, and the answer is this array of actions, saved as a trajectory file. For RoboTwin 2.0, for example, a row has 34 columns (16 joint position targets, 16 joint velocity targets and two gripper commands) and one row is applied per 4 ms physics step. In the `Privileged` tests of Embodied Olympics - Match and EmbodiedSWE the answer is instead a program (one that plays the match, or one that controls the robot through the task), and the verifier runs it in a fresh simulator.

`Standard` Setting: the simulator as a service. In the `Standard` and `Realtime` settings, the simulator runs as the simulator service in a separate container that the agent cannot access. The agent reaches it over HTTP through a Python client library that the exam provides. One call returns the specification of the robot: the action types with their layouts, units and limits; every joint with its position, velocity and effort limits and drive gains; the grippers and tool frames; the cameras with their intrinsics and mounting; the layout of the joint readings; the conventions of the world frame; and the budgets. A second call returns observations: either the current camera images, without advancing the simulation, or the images after the simulation has run for a requested number of seconds with the last command held. A third call executes an action chunk of T rows, each of dimension d, and returns the video of the execution. Where the service offers it, a fourth computes forward and inverse kinematics on the model of the robot without advancing physics. A reset call exists only in tasks that grant resets. Every response contains the camera images (RGB, and depth on request), one row of joint readings per image (joint positions and velocities, gripper opening, tool pose and an estimate of the drive effort), the camera calibration, the simulated time and the time the agent has left. Object poses, segmentation, contact forces on objects, scene files, the seed and the success rule are never sent, and any other request, such as a query for object states or for success, returns an error. The images come back as binary data that the agent's code can store and process, so they do not have to pass through the model's context.

From a chunk to motion. Joint position rows are dt seconds apart, where dt is a multiple of the control period and at most 0.5 s. The service rejects a value of dt that breaks this rule and does not round it. The service streams the rows to the drives of the joints in the way a position controller on a real robot follows a trajectory, either interpolating between rows or holding each row until the next. It clips any value outside the joint limits and reports each clipped entry in the response. End-effector rows go through the inverse kinematics or the motion planner of the suite. Where the original benchmark controls the robot through a higher-level controller, the agent sends the commands of that controller, as in RoboCasa and in the whole-body controller of BiGym 2.0. Each suite keeps its own control rate and physics step. Control rates range from 5 Hz (the mobile robot in MolmoSpaces) to 500 Hz (AutoBio, where one row is one 2 ms physics step), and one control step runs up to 100 physics steps (VLABench). The success rule is checked after every action chunk and every observation request that advances physics, and the first success ends the episode.

`Realtime` Setting: a simulation that keeps running. The `Realtime` Setting uses the same calls. From the first request that advances physics, the simulation advances one control step every S control periods of wall-clock time, whatever the agent does, where $S \geq 1$ is a slowdown factor that each task states. A submitted chunk is answered at once and replaces the queued actions starting at a control step that the agent specifies. When the queue is empty, the robot holds its last target. The agent can therefore run a program in its own container that observes the robot and sends new chunks in a loop. By default, S is 1 when the simulator runs at least three times faster than real time, and larger otherwise, so that the service can run each step on

time. The service records the steps it could not run on time, and the time the robot stood still because of them.

RoboEngineer. RoboEngineer uses the `Standard` interface in a different way. The agent tunes a controller in a simulator that the agent can access and deploys the controller to a second simulator, which the agent cannot access and whose dynamics differ from those of the first in ways not disclosed to the agent. The verifier then scores the controller that the agent selects on episodes that the agent has not seen.

Step limits and time budgets. The `Standard` and `Realtime` settings have no step limit, apart from RoboDojo's own (Figure 4) and a cap on episode length in many other suites, which the instruction states. A `Privileged` trajectory may have at most a stated number of rows. The main limit is the agent's wall-clock time, from 30 minutes to four hours depending on the suite, and it is the same in every setting of a task. Each response reports the time left, counted from the first request of the agent. One call may return only a stated number of images and may run for at most 15 minutes by default, so that one call cannot exhaust the memory of the service or occupy it indefinitely.

Limits. The limits of a test are the agent's wall-clock time, the resets that the task grants, the slowdown S in the `Realtime` Setting, the per-call limits above, and the CPU, memory, process, disk and GPU memory caps of each container, which a resource monitor records and enforces where the container runtime does not. `Privileged` tests allow unlimited resets, and `Standard` and `Realtime` tests allow a number of resets that each task sets and states (0 by default). These values are part of the configuration of a result, and results are compared only between runs with the same configuration.

C.2. Grading

Success rules. Each task is graded by one success rule, a condition on the simulator state. Adapted suites keep the success rule of the original benchmark and tighten it in some tasks. Examples are the success function of a RoboTwin 2.0 task, the BDDL goal of a BEHAVIOR-1K activity evaluated on OmniGibson object states, and the solved flag of MyoSuite. For the suites that we built, we wrote our own rules on the simulator state. The success rule of Embodied Olympics - Fun checks the suite's scoring checkpoints, legal contacts and fouls after every 1 ms physics step, RoboDynamicExam checks ordered phases (a catch, a hold, a crossing, a release) after every 1 ms physics step, and DexHandExam computes its checkpoints from contact pairs and contact forces. The instruction states each condition of the success rule with the same thresholds that the rule uses.

Settle and hold windows. Some rules are checked over a time window. A hold window requires the goal to hold for a stated time, so that an object in flight or a grasp that is about to slip does not count. In BEHAVIOR-1K the goal must hold for 1 s with the object released and at rest. MyoSuite requires its solved flag for 1 s (5 s for the Baoding balls), BiGym 2.0 requires its goal for 1 s and no fall during the episode, and ManiGuard requires its goal for 10 consecutive control steps and no safety violation during the episode. A settle window keeps the simulation running with the last command held while the rule is checked, as in RoboLab, which holds the last command for 2 s. Rules without a window are checked on a single state. A window is part of the

rule of the task and applies in every setting. It is checked on the live simulator in the `Standard` and `Realtime` settings and after the last row of the replay in the `Privileged` Setting.

`Privileged` Setting: a fresh replay. The verifier rebuilds the frozen scene in a fresh process, checks that its initial state matches the stored hash, replays the submitted actions open loop and applies the success rule to the final state of the replay. By default it replays twice and requires both replays to succeed and to end in the same final-state hash, so a trajectory that succeeds only by chance in a nondeterministic run does not pass. Because only the actions are replayed, a change that the agent made directly to the state of its own simulator, such as moving an object directly, has no effect on the verdict. A file that is not a valid trajectory fails.

`Standard` and `Realtime` settings: the live episode. The simulator service grades the episode that it ran and checks the hash of the start state when the episode begins and after every reset. When the agent's time is up, the program that runs the evaluation stops every process in the agent's container before it signals the service. From then on the service refuses every request that would advance the simulator, runs any settle window and writes its verdict with the recorded trajectory. The verifier reports that verdict, and nothing is replayed to decide it. The verdict depends only on the state of the simulator, never on what the agent reports. An episode in which the agent's actions cause a simulator error fails. A failure of the infrastructure, such as a scene that does not rebuild to its stored start state or a service that crashed, gives no verdict, and the whole attempt can be rerun. With resets, only the last episode is graded. We also record whether the first episode, before any reset, succeeded, and report this as first-episode success.

Isolation. The verifier runs in its own container, built from the task's grading files, without network, and receives only the files that the task declares for grading: the trajectory file in the `Privileged` Setting and the record of the simulator service in the `Standard` and `Realtime` settings. It first removes from the agent's output directory every entry that is not a regular file, such as a link. No writable volume is shared between the agent's container and the containers of the simulator service and the verifier, and the simulator service opens no port on the host. The agent can never read the reference solution or the files of the verifier, and in the `Standard` and `Realtime` settings it cannot read the success rule either. A program submitted as the answer runs under a user account without administrator rights and without network, cannot read the files of the simulator or the verifier, and every process it starts ends before the verdict is written. The verdict is written before the video of the attempt is rendered, and the video never changes it.

C.3. Physics

Simulators. Each suite keeps the simulator of its original benchmark, and new suites use the simulator that suits their physics. MuJoCo [143] runs most suites, directly or through robosuite, dm_control and MyoSuite, in versions 3.2 to 3.14 and mostly on the CPU. NVIDIA Isaac Sim and Isaac Lab with PhysX run RoboLab, RoboDojo, YAMLab, AM-Bench, LabUtopia and EmbodiedSWE, and OmniGibson on Isaac Sim runs BEHAVIOR-1K, ManiGuard and iSpatialGym. SAPIEN 3 with PhysX runs RoboTwin 2.0, and through ManiSkill 3 it also runs RoboPaint and RoboDynamicExam. Genesis runs RoboWits. RoboEngineer combines MuJoCo with learned dynamics models, such as a learned ground-effect model of a quadrotor and a wind-force model fitted to published flight logs.

What is simulated. All suites simulate rigid bodies with frictional contact, joint limits and actuator limits. Several also simulate deformable and particle matter: cloth in the blanket tasks of RoboARC (a deterministic GPU solver) and the folding tasks of RoboDojo (GPU PhysX), ropes and cords as soft bodies in RoboARC, liquids as GPU PhysX particles in RoboDojo and ManiGuard, and the object states of OmniGibson in BEHAVIOR-1K, where objects can be sliced, cooked or filled with particles. MyoSuite moves its limbs with simulated muscles instead of motors, and RoboPaint models brushes and pens on paper. Physics steps range from 1 ms (Embodied Olympics - Fun and RoboDynamicExam) to 1/60 s (LabUtopia), with 2 ms in DexHandExam, DexJoCo and AutoBio and 1/120 s in iSpatialGym, ManiGuard, AM-Bench and YAMLab.

Determinism. `Privileged` grading needs a simulator that replays bit for bit, and the verifier checks this on every attempt by replaying twice. The frozen scene is also rebuilt and checked against its stored start-state hash before every episode, and a host that cannot rebuild it gives no verdict. Where a simulator does not replay bit for bit, as with the GPU cloth and liquids of RoboDojo, the task is offered only in the `Standard` Setting, where the live episode is graded, and its video is a recording of the live episode rather than a replay.

Realism rules. The robot moves only through its own actuators and the controllers stated in the task. In the `Standard` and `Realtime` settings the agent has no way to set the state of the simulator, and in the `Privileged` Setting only actions are replayed, so moving an object directly has no effect on the grade in any setting. The suites we built for the exam add stricter rules. In Embodied Olympics - Fun, the simulator never attaches a held object rigidly to the hand, never pins a body or moves it with a motion-capture constraint, and never sets a velocity directly after the start. Balls come from ball machines and goalkeepers are force-limited mechanisms. Each robot uses its own hands and actuators, modeled on measured actuator data from the manufacturer (torque-speed curves, rotor inertia and friction). Where we increased a parameter above the manufacturer's value, the increase is at most a factor of 1.5 to 2 and is stated in the task; the one stated exception is the grip on the high bar. DexHandExam has no rigid attachments or scripted object motion, so objects move only through contact and gravity, and in the furniture assembly tasks of RoboARC, the robot holds parts only by finger friction. Where a suite keeps a grasp aid, its documentation states it. In the knot, ring and rope tasks of RoboARC, a rigid attachment holds a grasped object and slips when the pull exceeds 60 to 65 N for 20 ms. Adapted suites keep the grasp model of the original benchmark, such as the assisted grasping of OmniGibson and the hand of MetaWorld+, which a motion-capture constraint moves.

D. Additional Results

This section describes the agents, the counting rules and the data snapshots behind Section 3. It then shows the success rates of Figure 7 as bars and gives further results by suite, by cost and by task type (Section 3.1).

Agents. We evaluate seven agents, each a model running in a harness at high reasoning effort. The three strongest are GPT-6 Astra in Codex, Claude Opus 5.5 in Claude Code and Gemini 4 Argon in Antigravity CLI. The other four are Qwen3.8 Max in Claude Code and three Flash-class models: Gemini 3.8 Flash in Antigravity CLI, GLM 5.3 Flash in Claude Code and DeepSeek V4.1 Flash in dsh. We report each result for a model in a particular harness, because the way the harness exposes tools, manages the context and ends a session is part of what we measure. Switching

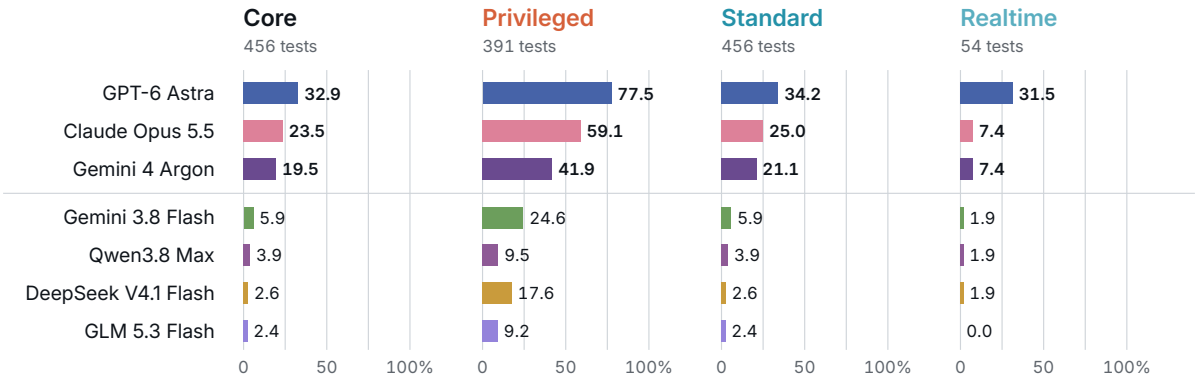


Figure 19 | Success rate of each agent on the core set. Each bar gives the share of core tests that one agent solves in the Core view or in one setting.

harnesses can change a model's success rate by about 10 percentage points (Section 3.4).

Counting rules. Section B gives the leaderboard rules. The Core column of Figure 7, which we call the Core view, counts each core task once in its default setting, and each of the other three columns covers all core tests of one setting (391 `Privileged`, 456 `Standard` and 54 `Realtime` tests). Every core task has a `Standard` test, and 403 of them use it as their default, so the Core view and the `Standard` column differ only on the other 53 tasks. When we pool the three strongest agents, each test counts once per agent. For results by suite, we treat the IKEA assembly tasks of RoboARC as a separate suite.

Data snapshots. Success rates are the final results of the public leaderboard, which have not changed since 2 November 2026. The stage-level failure labels were made on the results of 27 October. The analysis uses only the labeled attempts that the final results still count (Section H.1). The labels of control approaches use a sample of attempts from 18 October. The controlled runs add features of the `Privileged` Setting to the `Standard` Setting or run a model in a different harness, and we use the final results of these runs. The 423-test analysis of content-filter blocks in Section E.3 uses the runs dataset of 3 November.

Success rate of each agent. Figure 19 shows the success rates of Figure 7 as bars, for the Core view and for each setting. GPT-6 Astra leads in every column and solves a third of the core tasks in the Core view. The three strongest agents keep the same order in every column except the `Realtime` column, where Claude Opus 5.5 and Gemini 4 Argon tie. Outside the core set, success rates are higher: the three strongest agents solve 85% of their attempts on the 154 `Privileged` tests outside it and 70% on the 167 `Standard` tests.

Paired outcomes. Figure 10 in Section 3.2 splits the 391 core tasks that have both a `Privileged` and a `Standard` test by the settings in which each agent solves them. GPT-6 Astra solves 5 tasks in the `Standard` Setting but not in the `Privileged` Setting, and Gemini 4 Argon solves 18 such tasks. On a single task, the `Privileged` result is therefore not an upper bound on the `Standard` result (Section 2.1).

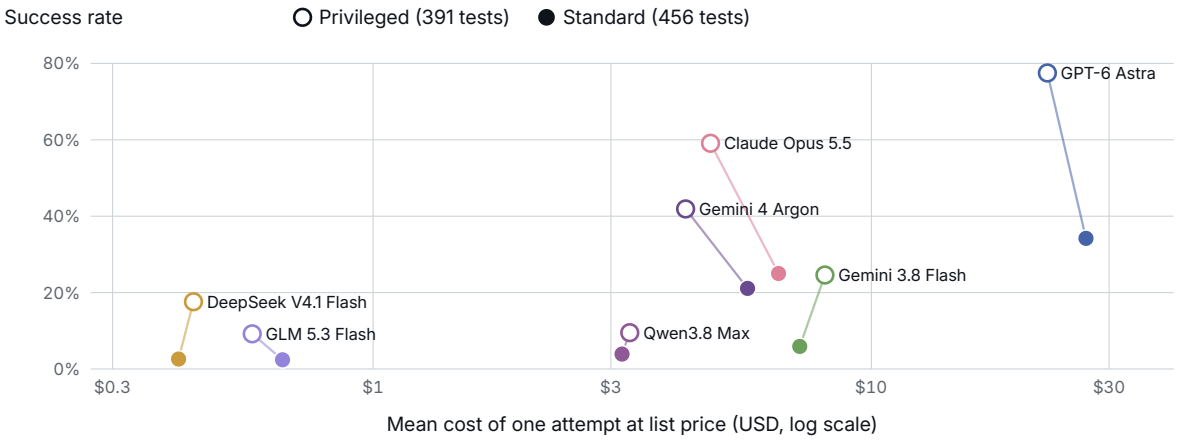


Figure 20 | Success rate of each agent (Figure 7) against the mean list-price cost of its attempts, in the `Privileged` Setting (hollow circles) and the `Standard` Setting (filled circles). A line joins the two circles of an agent.

Results by suite. GPT-6 Astra leads or ties in most suites, while Claude Opus 5.5 and Gemini 4 Argon each lead in some (Figure 9). We count only suites with at least five tests in the setting. In the `Standard` Setting, at least half of the tests are solved by none of the three strongest agents in 13 of 22 suites, among them DexToolBench, the IKEA assembly tasks, HumanoidBench and BEHAVIOR-1K. The three strongest agents solve at least 60% of their attempts only in ManiGuard, whose safety rules are stated in the instruction. In the `Privileged` Setting, at least half of the tests are unsolved only in HumanoidBench, and the three strongest agents solve at least 70% of their attempts in 10 of 20 suites: Embodied Olympics - Match, EmbodiedSWE, RoboDynamicExam, RoboCasa365, RoboWits, MuJoCo Playground, Embodied Olympics - Fun, BiGym 2.0, RoboPaint and ManiGuard.

Cost of an attempt. Figure 20 prices each attempt from the tokens that its harness logged, at the list prices of 5 November 2026. Several agents ran on subscriptions, so these costs are not what the runs were billed. Each of the 6,248 attempts with a verdict is priced from the run that the leaderboard counts for it in the released logs. One of these runs logged no tokens and is left out. The success rates are those of Figure 7. In the `Standard` Setting, the mean cost of an attempt is $27.0 for GPT-6 Astra, $6.5 for Claude Opus 5.5 and $5.6 for Gemini 4 Argon. Gemini 3.8 Flash averages $7.2 per `Standard` attempt and solves 5.9% of the `Standard` tests. For every agent, in the `Privileged` and in the `Standard` Setting, the median failed attempt costs more than the median solved attempt. The folder `app_cost_vs_success` of the figure sources gives the prices and the matching of attempts to runs.

Results by task type. A task type describes a core task as a whole, whereas the stage-level failure labels of Section 3 assign a type to each stage of a task. By task type (Figure 21), GPT-6 Astra's lead is widest in the Dynamic Interaction and Precision & Contact types, and Claude Opus 5.5 ties with it in Object Transport, at 37%. For the three strongest agents together, Object Transport has the highest success rate of the nine task types, at 35% of their attempts, and Tool Use has the lowest, at 12%.

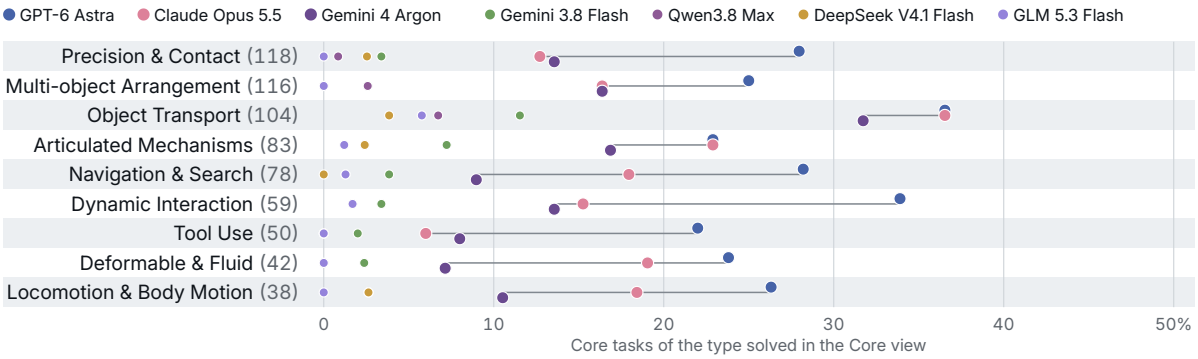


Figure 21 | Share of the core tasks of each type that each agent solves in the Core view, with a gray line joining the three strongest agents. The number of core tasks of the type is in parentheses, and a task can have several types.

E. Further Analysis

This section gives the full analyses behind the key takeaways of Sections 3.2 to 3.5, in the same order. The definitions and statistics that these analyses use are in Sections G and H.

E.1. Causes of the Drop from the `Privileged` to the `Standard` Setting

To measure how much two features that the `Standard` Setting lacks contribute to the drop, we ran GPT-6 Astra in Codex on the `Standard` and `Realtime` core tests with one or both of them added to the `Standard` interface (Figure 11a, Section G). The first feature is a query that returns the state of every object, including its pose, size and velocity. The second is unlimited resets together with calls that save the simulator state and restore it later. All numbers of the controlled runs in this subsection are the final results of these runs. On the 456 tests, GPT-6 Astra solves 33% with the unmodified interface. The state query raises this rate to 47%, and unlimited resets raise it to 52%. With both features, the agent solves 66%, against 77% on the 391 `Privileged` tests. **With unlimited resets, GPT-6 Astra uses saved states to retry a hard stage without repeating the whole task.** Of the 236 successes of the reset run, 22% came in the first episode, and 58% came in an episode that started from a saved state.

The stage labels show at which kinds of stages the `Standard` Setting raises the failure rate (Figure 11b). The failure rate of a stage type is the number of its stages at which attempts fail, divided by the number of its stages that attempts reach. We compare it between the two settings within each suite and average over the suites (Table 12). Section H.5 defines the precise and coarse groups of stage types and gives the statistical tests. Among the stage types compared in at least two suites, the failure rate rises by 23 to 52 percentage points for precise types and by 3 to 23 points for coarse types. The failure rate of cleaning, which is in neither group, also rises by 46 points. Pooled within suites, the failure rate rises from 22% to 54% at precise stages and from 22% to 40% at coarse stages. Figure 12 shows an example, and Section H.6 gives it in full. The difference between the two groups is smaller when the same agent reaches the same stage of a task in both settings. There, the rise is 21 points at precise stages and 17 points at coarse stages. At follow-path stages, most of them waypoints of DexToolBench, the rate rises by 11 points in this comparison. Without these stages, the rise at precise stages is 28 points.

Combining the controlled runs with the stage labels shows which feature helps at which kind of stage (Figure 11c, Table 14). We take the `Standard` and `Realtime` tests that GPT-6 Astra failed

in the leaderboard run, which is the baseline of the controlled runs, and group them by the type of the failure stage. Of these failures, 80 are at precise stages and 97 are at coarse stages. Some of these failures would turn into successes if the test were simply run again. We therefore compare each run with a fourth controlled run in which the interface no longer reports when an episode succeeds. This run repeats the same tests and changes only the success signal, so we use it as a rough approximation of a second attempt. Its success rate is within 4 points of the baseline's (Table 5), but this run does not separate the variation between attempts from the effect of hiding the success signal. At precise stages, the state query solves 20% of the failures, against 19% for the run without the success signal, Unlimited resets solve 36%, and 28 of these 29 successes came after a reset or a restored state. With the state query, the agent called it on every precise-stage failure for which the number of calls was recorded. **At precise stages, adding the state query to the resets raises the share solved from 36% to 50%.** At coarse stages, the state query alone solves 34% of the failures, against 15% for the run without the success signal, and the two features together solve 67%, against 60% in the `Privileged` Setting. With both features, the agent solves 50% of the precise failures, compared with 74% in the `Privileged` Setting, and most of this remaining gap lies at follow-path, thread and assemble stages. Neither added feature lets the agent advance the physics one step at a time or call planning primitives, as the `Privileged` Setting does. Whether these two abilities account for the remaining gap has not been determined.

Claude Opus 5.5 reached no goal pose in the 20 resets that a DexToolBench task allows (Section F). The comparisons in this subsection have limits. The controlled runs use one agent. Whether a failed precise stage comes from an error of perception or an error of control has not been determined, because the labels cannot separate the two (Section H.5).

E.2. Where Agents Fail

Failures on a task tend to happen at the same stage (Figure 13a). We measure this by the concentration of failure stages, which is the probability that two of the failed attempts on a test, chosen at random, failed at the same stage. Each agent makes one attempt per test, so the two attempts always come from different agents, and the measure pools the agents that failed the test. On the `Standard` tests that at least two agents failed, the concentration averages 61%, and we compare it with two baselines. In the first baseline, each agent fails at a stage chosen at random, with the probability of each stage position set by how often failures occur at that position across the exam, so early stages are more likely. This baseline gives 37%. In the second baseline, an agent fails at each stage with the failure rate of the stage's type on the other tasks, and this baseline gives 48% (Table 16). For the three strongest agents alone, the concentration is 57%, against 31% for the first baseline built from their failures on the other tasks (Section H.5). **Failures therefore concentrate more than the stage types predict.** The most common failure stage is usually the first stage of the task. Of the 250 `Standard` tests whose failures have a single most common stage, this stage is the first stage in 70%, and it is the first stage in 60% of those whose tasks have at least two types of stages. In a RoboDynamicExam task, for example, all seven agents caught a thrown ball in a cup in the `Standard` Setting and failed to catch the ball again after it rebounded off a ramp, in three different ways (Figure 14, with all seven attempts in Section H.6).

Failure rates differ between stage categories (Figure 13b). Pooled over the `Privileged` and `Standard` settings, attempts fail at 64% of the material stages (such as wiping or pouring) that they reach, at 45% of the body stages (such as walking or keeping balance) and at 21% of the mechanism stages. Within a category, the rate also depends on how the robot must act. Opening a door or a lid fails in 41% of cases, whereas closing fails in 7% and pressing fails in 3% (Table 18). Relation stages, which place or fit one object relative to another (Table 7), fail more often when

two arms act together or the legs carry the robot's weight than when one hand acts (Figure 28 in Section H.5). **However, which stage types fail most differs from suite to suite.** When we rank each pair of stage types by their failure rates on all suites but one, the ranking holds in the held-out suite for 51% of the 203 pairs, about as often as a random ranking would. Whether an attempt fails at a stage that it reaches is also predicted better by the identity of the agent than by the stage type (Section H.5). Each category occurs mostly in particular suites and with particular robots, so these rates do not show that the category itself causes the failures.

The most common failure stage of a task is also usually the same in both settings. Of the tasks whose failures have a single most common stage in both settings, 81% have the same such stage in the two settings, compared with 51% if each setting's failure stages were sampled independently. Excluding the tasks whose most common failure stage is the first stage in both settings, the share is still 55%, compared with 9% for random failure stages (Section H.5). **Of the attempts that get past the stage at which the other agents on the same test most often failed, 67% pass the test in the `Privileged` Setting and 34% in the `Standard` Setting.** Part of this link between getting past the stage and passing follows from the order of the stages, because an attempt that gets past the common stage is already closer to the end of the task.

E.3. Differences between Agents

Tests solved by one agent and by another. For two agents A and B, the cell in row A and column B of Figure 15b gives the share of the tests that A solves that B also solves. The denominator of row A is the number of tests that A solves. It is given in parentheses after the name of A, and it equals A's count of solved tests in the `Standard` column of Figure 7. The diagonal is left empty, because it would always be 100%. The last column gives the share of A's solved tests that at least one agent above A in Figure 7 also solves. A test breaks the order when some agent solves it and GPT-6 Astra fails it. We count such a test under the highest agent in Figure 7 that solves it, so every agent above this agent fails the test. We use the `Standard` Setting in the main text because every core task has a `Standard` test and all agents reach the robot through the same interface there. The Core view differs from the `Standard` column on only 53 tasks. In the rows of the three strongest agents, the Core-view shares differ from those of Figure 15b by at most 3 percentage points, and 66 tests break the order, as in the `Standard` column. Each of the four weaker agents solves 11 to 27 `Standard` tests. A single test therefore changes a share in its row by 4 to 9 percentage points. Among these four agents, the order of Figure 7 does not hold. Gemini 3.8 Flash, for example, solves 2 of the 11 `Standard` tests that GLM 5.3 Flash solves.

In the `Privileged` Setting, GPT-6 Astra solves 92% of the 231 `Privileged` tests that Claude Opus 5.5 solves and 95% of the 164 that Gemini 4 Argon solves (Figure 22). The three strongest agents solve all 125 `Privileged` tests that a weaker agent solves. Of the 325 `Privileged` tests that some agent solves, 22 break the order: Claude Opus 5.5 solves 18 of them and Gemini 4 Argon solves 4. In the `Standard` Setting, 66 of the 222 tests that some agent solves break the order. Every agent except GPT-6 Astra solves at most 4 `Realtime` tests, so we do not draw a `Realtime` matrix. Of the 19 `Realtime` tests that some agent solves, 2 break the order, and Claude Opus 5.5 solves both.

Where the order breaks. Table 3 counts the tests that break the order by suite. In the `Standard` Setting, Embodied Safety Exam has the most, 12 of its 50 tests, followed by RoboARC with 10 of 50. In the `Privileged` Setting, BEHAVIOR-1K has the most, 8 of 71. We checked four of these tests (Figure 23). For three of them we read the attempt's log and the simulator service's record and watched the replay video of the graded episode. In all three, the live run and the

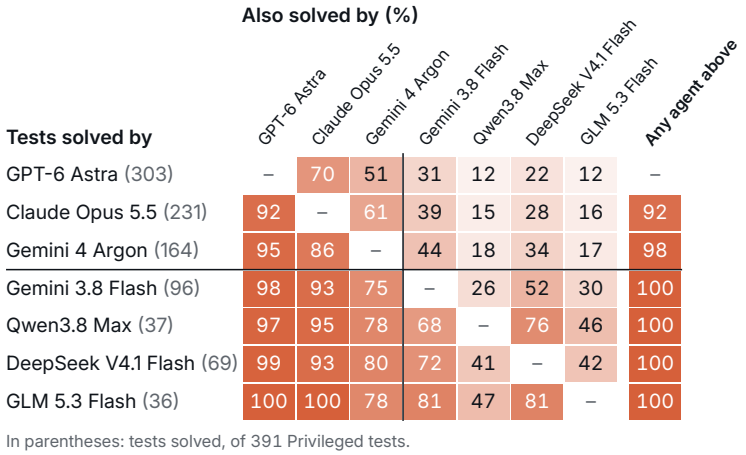


Figure 22 | Share of the `Privileged` tests solved by the row agent that the column agent also solves, as in Figure 15b. The number in parentheses counts the tests that each agent solves, and the last column gives the share that any agent above the row agent in Figure 7 also solves.

Table 3 | Tests that break the order of Figure 7, by suite and setting. "Tests" gives the number of tests of the suite in the setting. Each test that breaks the order is counted in the column of the highest agent in Figure 7 that solves it, and "Weaker" sums the four weaker agents. The six suites with the most such `Standard` tests are listed, and "Other suites" sums the rest.

	`Standard`				`Privileged`		
Suite	Tests	Claude Opus 5.5	Gemini 4 Argon	Weaker	Tests	Claude Opus 5.5	Gemini 4 Argon
Embodied Safety Exam	50	8	4	0	–	–	–
RoboARC	50	8	1	1	50	5	1
Embodied Olympics - Fun	56	5	1	0	56	1	0
ManiGuard	7	4	0	0	7	0	0
BEHAVIOR-1K	71	1	2	0	71	6	2
DexHandExam	23	1	2	0	23	1	0
Other suites	199	21	6	1	184	5	1
All	456	48	16	2	391	18	4

replay agree on the verdict. The fourth attempt has no published replay, so we checked it from its log and record only. None of the four records shows an infrastructure error. In the RoboARC blanket test, which all six other agents failed, Gemini 3.8 Flash computed the corner positions from camera images and depth. It grasped the two far corners with both grippers and folded the blanket in half. It then ran four adjustment programs before the episode ended with success. In an iSpatialGym test, whose goal is given as a picture of the finished scene, DeepSeek V4.1 Flash placed a wrapped food package on a plate. The simulator service ended the episode with success at control step 407 of the 1,900 that the task allows. Gemini 4 Argon opened a jam jar with a dexterous hand in a DexHandExam test. It hung three mugs on a mug tree in a RoboDojo test. GPT-6 Astra and Claude Opus 5.5 failed both tests. These checks cannot show that the agents that failed would fail again. Each agent made one attempt per test, and the leaderboard has no repeated attempts. A test that one agent solves and another fails may therefore reflect the variation between runs as well as a difference between the agents.

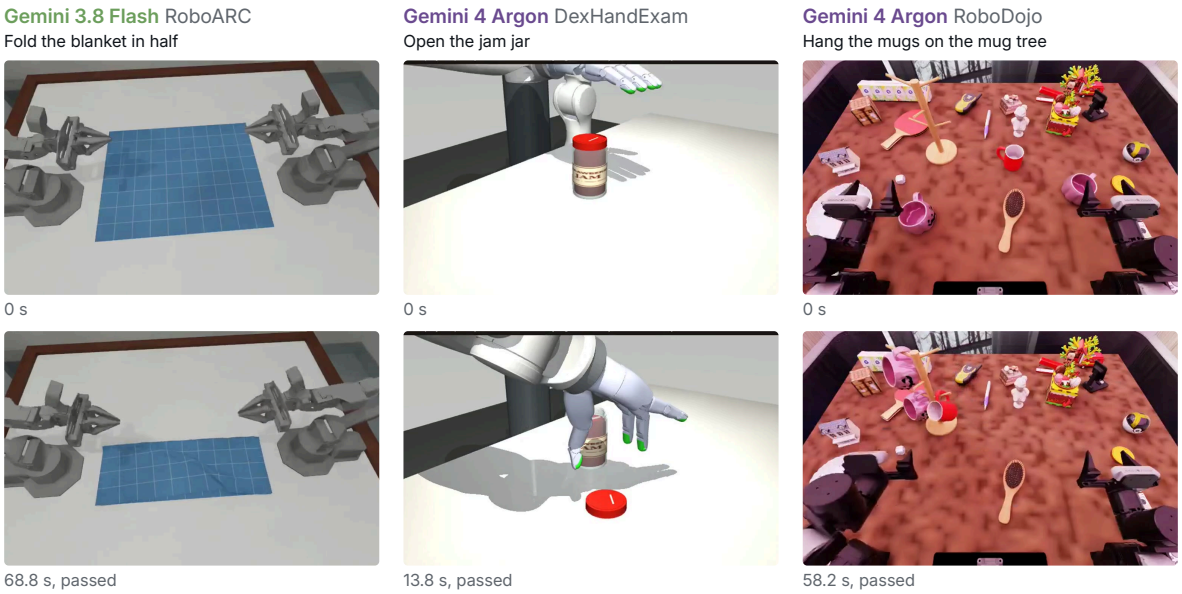


Figure 23 | First and last frames of the graded episodes of three `Standard` tests that break the order of Figure 7. Gemini 3.8 Flash folds a blanket in RoboARC. Gemini 4 Argon opens a jam jar in DexHandExam and hangs three mugs on a mug tree in RoboDojo.

How the agents fail. When GPT-6 Astra and another agent both fail a test after meeting different numbers of stages, GPT-6 Astra has usually met more of the task's stages (Table 19). The kind of failure differs as well (Figure 15a). The four weaker agents fail without contact at 11% to 18% of the contact stages they reach, against 2% to 4% for the three strongest agents, and the weaker agents' rate is higher in 17 of the 18 suites with at least 20 such stages per group (Section H.5). A lost hold makes up a larger share of GPT-6 Astra's failures than of any other agent's, but GPT-6 Astra also reaches more stages that need a hold. Per stage reached that needs a grasp, it loses the hold less often: in 5.5% of cases, against 8.3% for the other six agents, with a 95% interval of the difference from 0.3 to 4.8 points (Section H.5). In a BEHAVIOR-1K task that all seven agents failed, for example, the three strongest agents each packed three or four foods into a lunch box and failed in three different ways, whereas none of the four weaker agents packed any food (Figure 30 in Section H.6).

Harness. The harness also changes the result (Figure 15c). We ran GPT-6 Astra in Claude Code, Claude Opus 5.5 in Codex, and Gemini 4 Argon in Codex and in Claude Code, with the same tasks, time limits and reasoning effort as in the main evaluation (Section G). Gemini 4 Argon solves 30% of 444 core tests in Codex and 25% in Claude Code, compared with 19% in its own Antigravity CLI, and both differences are significant. GPT-6 Astra solves 33% of 444 tests in Codex and 25% in Claude Code, and this difference comes mostly from episodes after a reset. Claude Opus 5.5 solves 24% of 444 tests in Codex and 23% in Claude Code. Part of Gemini 4 Argon's gap may come from how each harness handles a reply that Gemini's content filter blocks. Antigravity CLI takes the blocked reply as the agent's final answer and ends the session. Among Gemini 4 Argon's labeled failed attempts, all in Antigravity CLI, 252 ended in this way (Section H.3). Codex sends the blocked request again, and Claude Code asks the model to continue, so the session goes on. In the runs dataset of 3 November 2026, on the 423 core tests that all three of its harnesses had graded, 126 of Gemini 4 Argon's attempts in Antigravity CLI had a blocked reply, and 6 of them (4.8%) passed, against 26.3% of its attempts without a block.

In Claude Code, its attempts with and without a block passed at 25.2% and 25.6%. In Codex, its attempts with a block passed less often than those without (18.2% and 38.4%), so a block may also indicate a harder attempt. These cross-provider runs pass the requests through an adapter that translates them for the other provider. The runs therefore only approximate a harness that supports the model directly.

Overlap among the three strongest agents. Figure 24 groups the core tests of each setting by which of the three strongest agents solve them. Sections 3.1 and 3.4 give the totals. Here, an agent solves a test alone when the other two of the three fail it. In the `Standard` Setting, all three solve 33 tests and none of them solves 236. GPT-6 Astra alone solves 59, Claude Opus 5.5 alone solves 32 and Gemini 4 Argon alone solves 16. In the `Privileged` Setting, all three solve 136 tests and none of them solves 66. GPT-6 Astra alone solves 71, Claude Opus 5.5 alone solves 13 and Gemini 4 Argon alone solves 4. The `Privileged` tests that only GPT-6 Astra solves include tasks of precise contact and of arranging several objects. The `Standard` tests that only Claude Opus 5.5 solves include object transport tasks. In a ManiGuard task, for example, Claude Opus 5.5 closed the hinged lid of a glass jar and carried the jar to the goal, whereas GPT-6 Astra knocked the jar over (Section F).

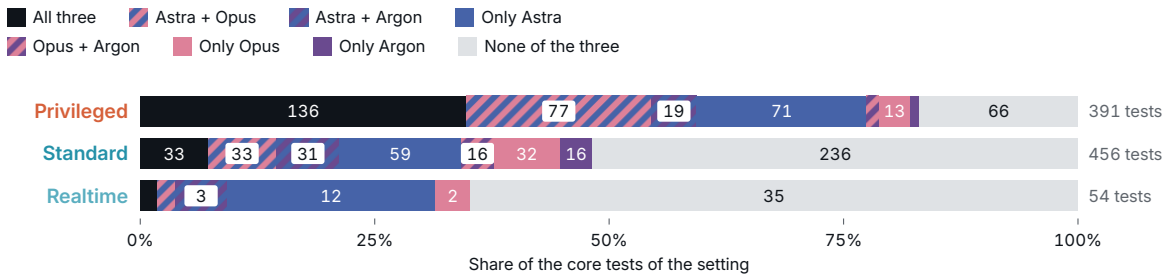


Figure 24 | Core tests of each setting, grouped by which of the three strongest agents solve them.

E.4. Control Approaches

A separate coding agent read the logs of 1,489 attempts, sampled at random from the runs of 18 October, and labeled each attempt with the approach that produced its graded answer (Section B). These labels have not been checked manually, so the shares are approximate. Only counts per setting are available, so they cannot show whether the approach changes with the suite, the task or its resources. Many `Privileged` tests give the agent a GPU to run the simulator, and their instructions name no training library. Apart from these, only eight tests give the agent a GPU: the GPU versions of the soccer match below and of a table tennis match, in each setting, and two RoboEngineer tests that ask the agent to train a walking policy. In the runs dataset of 3 November, 51 of the 12,237 attempts are on these eight tests. Figure 2 shows one example of each approach.

The labels split *Code to control* into three kinds. Open-loop programs compute the whole motion in advance. Closed-loop programs read the robot's state or cameras as they run. Parameter searches try many episodes to find a passing motion. The available counts do not show how a search over a closed-loop program was classified. In the `Privileged` Setting, agents use the simulator to compute or search for a motion before they submit it. Most of their programs in this setting are closed-loop, and 114 of the 549 programs search over many episodes. **The same agent can use different control approaches for the same task in different settings.** On a RoboTwin 2.0 task, GPT-6 Astra scripted a hand-over of two bottles with the simulator's motion

primitives in the `Privileged` Setting. In the `Standard` Setting, it sent end-effector commands one at a time and checked the camera images after each command (Section F).

In the `Standard` Setting, *Direct control* places the model inside the control loop, so the robot waits for the model's reasoning before each chunk. **In the `Realtime` Setting, the environment keeps running, and most attempts run the control loop in a program.** In a badminton match of Embodied Olympics - Match, Claude Opus 5.5 let a program track the shuttlecock and swing the racket. At each pause, it revised the program from the program's logs and won 12–10 (Section F).

The soccer match is the only task with a GPU version and a CPU version that the seven agents attempted. A table tennis task also has a version with a GPU, but only the screening agent GPT-6 Luna attempted it. In both soccer versions, each team has two Booster T1 humanoids controlled at the joint level, and the agent's team plays a team with a learned policy on the same frozen scene, with a limit of four hours. The GPU version gives the agent a GPU with 12 GiB of memory, the mjlab simulator with the rsl_rl reinforcement-learning library and PyTorch, and training copies of the match that run on the GPU. Its instruction lets the agent train a controller or write one, and it gives an example of training a walking policy with rsl_rl. The CPU version has no GPU and no machine-learning framework, and its instruction allows learning code in NumPy or SciPy.

Table 4 counts every attempt of the seven agents on the two versions in their final results, including the runs of the harness comparison (Section G).

Table 4 | Attempts of the seven agents on the two versions of the soccer task, in their own harness and in the harness comparison (final results). "Played with it" counts the attempts that played the match with the policy they trained. The `Privileged` Setting does not record goals.

Setting	Version	Attempts	Trained with RL	Played with it	Scored a goal	Won
`Standard`	GPU	6	5	4	3	2
	CPUs only	6	0	0	2	0
`Realtime`	GPU	10	10	8	6	2
	CPUs only	11	0	0	1	0
`Privileged`	GPU	6	2	0	–	3
	CPUs only	6	0	0	–	4

In the `Standard` and `Realtime` settings, 15 of the 16 attempts with the GPU trained a policy with reinforcement learning, and none of the 17 with only CPUs did. Of these 15, 12 played the match with the trained policy. The one attempt that did not train, by DeepSeek V4.1 Flash in the `Standard` Setting, wrote a joint controller by hand. With only CPUs, agents wrote walking gaits by hand, for example with inverse kinematics or a capture-point model. Of the 13 attempts whose final reports we read, 8 tuned the gait with CMA-ES, the cross-entropy method, or a grid or random search. One attempt, by GPT-6 Astra in Codex in the `Realtime` Setting, distilled its hand-written controller into a small network by supervised learning. We do not count it as training with reinforcement learning.

With the GPU, agents scored in 9 of 16 matches and won 4, all with a trained policy. With only CPUs, they scored in 3 of 17 matches and won none. Counting only the attempts in each agent's own harness, the GPU attempts trained in 11 of 12 cases and won 3 times, and the 13 CPU attempts neither trained nor won. Qwen3.8 Max has no attempt on the GPU version. With GPT-6 Luna added, 17 of 18 GPU attempts trained and 4 won, and none of the 19 CPU attempts trained or won. The same models appear in several settings and harnesses, so the attempts are not independent, and we report these counts without a significance test.

Figure 18 shows GPT-6 Astra in Codex on both versions of the soccer match in the `Standard` Setting. With the GPU, it trained walking and ball-control policies and combined them with hand-written tactics. Its robots walked 690 m, against 693 m for the opponent's, and it won 10–8. With only CPUs, it wrote a gait by hand and tuned it with differential evolution and CMA-ES. Its robots walked less than half as far as the opponent's and fell 59 times. The mercy rule ended the match early at 3–13. In this example, the two robots of GPT-6 Astra touched the ball 150 times with the GPU and 54 times with only CPUs. The opponent's robots touched it 142 and 97 times, respectively. The goal times in the figure come from the referee state that the agent queried after each goal, and the frames come from the replay of each attempt.

In the `Privileged` Setting, every win reused the opponent's walking network. With the GPU, 2 of 6 attempts ran reinforcement-learning code and 3 won. With only CPUs, 4 of 6 attempts won. In this setting, the agent can replay the same match in its own simulator. According to their code or final reports, all seven winning attempts reused the opponent's walking network that comes with the simulator, and none used a policy that it had trained.

We decided whether an attempt trained a policy from three signals: calls to a reinforcement-learning library or gradient updates in the agent's code, the GPU memory that the resource monitor recorded, and the attempt's final report, which another agent read. The three signals agree on every `Standard` and `Realtime` attempt with the GPU. No person has checked these labels. The comparison covers one task and one frozen scene. The two versions differ at the same time in the GPU, the library, the training copies of the match and the instruction, so the comparison cannot separate their effects.

F. Additional Attempts

This section describes further attempts that illustrate the findings. Every attempt is public and can be replayed step by step on the exam's website.

RoboLab, utensils in a mug (`Standard`, solved only by Gemini 4 Argon). Gemini 4 Argon located the spoon, fork and mug with three depth cameras, planned waypoints with inverse kinematics and sent them as six action chunks, and it succeeded in its first episode. In the attempt of Claude Opus 5.5, the fork slipped.

DexToolBench, red brush (`Standard`, solved by none of the three strongest agents). A DexToolBench task asks the agent to move a tool with a dexterous hand through goal poses, 38 in this task. Claude Opus 5.5 pushed the brush until it stuck out over the table edge, grasped it and reached goals 0 to 4. Goal 5 needed a 90-degree turn about the brush's long axis. The palm-down grasp could not make it, and the brush flew out of the hand as the arm changed configuration.

RoboARC, IKEA Stegesund (`Privileged`, failed). With the full state, GPT-6 Astra pressed the halves together, slid in the shelf and drove the first bolt. Its hex key turned only 120 degrees per insertion, so one bolt took 40 minutes, and at the three-hour limit the second bolt was still 3.7 mm short. None of the three strongest agents solved this task.

BEHAVIOR-1K, setting a table (`Privileged`, failed). GPT-6 Astra spent 62 minutes placing two plates, and the robot then tipped over against the kitchen island on the way to the fridge.

RoboARC, threading a rope through six holes (`Standard`, failed). Claude Opus 5.5 mapped the wall, its holes and the coiled rope from depth point clouds. The rope slipped from the gripper whenever the robot turned its wrist, and the agent threaded none of the six holes.

RoboDojo, inserting a key (`Standard`, failed). Claude Opus 5.5 located the key and the lock from depth images but dropped the key at least nine times during hand-overs and large joint moves. Its most accurate insertion, at minute 36, was about 9 mm off and hit the face of the lock.

RoboDynamicExam, toss, recatch and deliver (`Standard`, failed). GPT-6 Astra caught the ball using depth images, but its first episodes ended early (one ended when the cup grazed a moving box) and used up its three allowed resets. In the last episode it caught and tossed the ball, but on the second catch the ball's contact with the tilted cup was flagged as a limit violation.

RoboPaint, a character in running script (`Standard`, failed). With no resets available, GPT-6 Astra calibrated the stroke width on a practice sheet, wrote five strokes and then retouched the character in one long stroke for 38 minutes. The overlap and edge scores met their thresholds, but the long stroke raised the total length of the inked path to 2,048 mm, above the limit of 1,450 mm. In addition, one of the nine regions was only 65.7% covered, below the required 80%.

HumanoidBench, opening a door (`Standard`, failed). Claude Opus 5.5 kept the humanoid standing with ankle feedback and located the handle with stereo cameras. It tried more than ten ways of opening the door (pressing, hooking, pushing up and using both hands) and pulled the handle back by up to 26 cm, but the door never opened. The agent used 45 resets in 27 minutes.

Embodied Olympics - Fun, dribble slalom (`Privileged` solved, `Standard` failed). In the `Privileged` Setting, Gemini 4 Argon saved and restored exact simulator states to try candidate kicks in eight parallel processes and passed all 13 checkpoints of the slalom. In the `Standard` Setting, it estimated the positions of the robot and the ball from the pixels of a fixed camera, about 9 cm off to the side, and its best episode passed four poles before the 120 minutes ran out.

BiGym 2.0, closing the dishwasher (`Privileged`, solved). Gemini 4 Argon read the dishwasher's code and collision shapes and pushed both racks in. The robot then crouched, hooked the door's front edge and swung the door shut along an arc. Its answer, a 693-step trajectory, passed two independent replays.

RoboCasa365, defrosting meat in the microwave (`Privileged`, solved). Claude Opus 5.5 used Jacobian inverse kinematics and closed-loop control of the base to take the box out of the freezer, put it in the microwave, close the door and press start. The agent submitted its answer at minute 46, and the answer passed on replay.

ManiGuard, dusty transfer (Gemini 4 Argon, `Privileged`, solved). Working in a copy of the simulator that it kept running, Gemini 4 Argon wiped 18 dust particles from the box and then tilted the tray so that the potato rolled in without being touched. It finished at minute 26, and two independent replays passed.

RoboPaint, cursive character (`Privileged`, solved). GPT-6 Astra read how the brush model draws a stroke, wrote a C++ program that reproduces this model and fitted three strokes with it. The first strokes that the agent executed passed every check, and the agent submitted its answer after four minutes.

BEHAVIOR-1K, carrying two boards (`Privileged` solved, `Standard` failed). A mobile robot with two arms must bring two boards into a hallway. In the `Privileged` Setting, Claude Opus 5.5 kept a copy of the simulator running, wrote separate scripts for grasping, opening the door and carrying, and tested each stage in this copy, where a fall of the robot required only a reset. Its answer passed after 72 minutes. In the `Standard` Setting, the same agent picked up both boards but then drove the base backwards and sideways with a single command and did not check the camera images while the base moved, until the robot hit a curb and tipped over.

ManiGuard, dusty transfer (GPT-6 Astra, `Privileged` solved, `Standard` failed). The robot must wipe a box clean and then tip a tray so that a potato rolls into the box without being touched. GPT-6 Astra touched or flung the potato in its first three `Privileged` episodes, then reset and passed with a new wrist angle. In the `Standard` Setting, the test failed because a finger touched the potato once while the robot regrasped the tray, although the potato later reached the box.

DexToolBench, screwdriver (`Standard`, failed). Claude Opus 5.5 estimated the axis of a screwdriver from the point cloud but could not determine how far the handle had rolled about that axis, and it used all 20 resets that this task allows without reaching any of its 76 goal poses.

ManiGuard, closing a jar (`Standard`, solved by Claude Opus 5.5). The robot must close the hinged lid of a glass jar and carry the jar to a goal without breaking the task's safety rules. Claude Opus 5.5 estimated the jar's shape and the position of the hinge from the depth point cloud, slid the robot's closed fingers under the lid and swung the lid shut about the hinge. It then lifted the jar by 1 cm to check the grasp before carrying the jar to the goal. GPT-6 Astra knocked the jar over while pushing the lid.

RoboDynamicExam, toss and recatch (`Privileged`). Claude Opus 5.5 used the arm's Jacobian to compute the largest downward acceleration of the point from which the ball is tossed, and it chose its toss and recatch to stay within this limit.

RoboWits, tipping a cube out of a container (`Privileged`). Claude Opus 5.5 tried six pour positions to tip the cube out of the container and kept the steadiest.

RoboTwin 2.0, two bottles into a dustbin (`Privileged` and `Standard`). In the `Privileged` Setting, GPT-6 Astra read the task's source code, scripted the grasps with the simulator's motion primitives, handed the bottles from one arm to the other in mid-air and submitted its answer after five minutes. In the `Standard` Setting, it sent end-effector commands one at a time and checked the camera images after each command. It knocked one bottle over, picked the bottle up again with the other arm and finished about 30 seconds before the time limit.

Embodied Olympics - Match, badminton (`Realtime`, won). A program written by Claude Opus 5.5 ran in the background, located the shuttlecock with two cameras, fitted a drag model to its flight and swung the racket. Whenever play paused, the agent read the program's logs and revised the program. It won 12–10 against a scripted opponent after trailing 7–10.

G. Controlled Runs

This section defines the controlled runs of Sections 3.2 and 3.4 (Figures 11a and 15c). All numbers are the final results of these runs.

Baseline. Every controlled run of Figure 11a uses GPT-6 Astra in Codex with the same reasoning effort, time limit and task versions as the leaderboard run of Figure 7, which serves as the baseline. For RoboDojo, as for the other suites, the baseline is the final leaderboard result. Each run covers one test of each of the 456 core tasks, in the task's default setting, which is `Standard` or `Realtime`. Each run changes one or two features of the `Standard` interface and nothing else. Apart from these features, every task behaves exactly as in the leaderboard run. We compare each run with the baseline on the tests that both have graded, and Figure 11a uses the 456 tests that the baseline and all three runs with added features have graded.

Object-state query. We add two read-only calls to the `Standard` interface: one lists the objects in the scene, and one returns their state. The state consists of position, orientation, size and bounding box, linear and angular velocity, and the joint positions of the robot and of articulated objects such as doors and drawers. In Embodied Olympics - Match, the state of the opponent and of the ball counts as object state. The calls do not advance the simulation or use up simulated time, and they never return success flags, scores, thresholds or goal regions. The only progress value they return is the paint coverage of the canvas in RoboPaint. We remove the fields that would reveal what the agent is meant to find out itself: goal markers and zones, objects that appear only later in a BEHAVIOR-1K task, hint notes in RoboCasa365, and hazard labels in Embodied Safety Exam.

Unlimited resets with saved states. The reset budget of every task is raised to 1,000, and the interface gains calls that save the current simulator state under a handle and restore it later. The simulator service still grades the last episode as it runs. Restoring a saved state does not erase a rule violation that has already been recorded. In Embodied Safety Exam, for example, recorded violations remain after a restore.

Both. This run combines the two features above.

No success signal. The interface no longer reports that an episode has succeeded. The episode does not end at success, and no success flag, progress or score is returned. The agent ends the episode itself with a finish call, or the episode ends when the time runs out or the task's own failure rule is triggered. The simulator service checks the success rule on the state at the end of the episode, so a success that was reached and then undone counts as a failure.

Harness comparison. For Figure 15c, we ran GPT-6 Astra in Claude Code, Claude Opus 5.5 in Codex, and Gemini 4 Argon in Codex and in Claude Code, with the same tasks, time limits

and reasoning effort as the leaderboard runs. Codex and Claude Code support only their own provider's API format, so an adapter passes their requests to the other provider and translates the answers back. The adapter rewrites parts of the requests. For example, it rewrites the system prompt that Claude Code sends to Gemini 4 Argon, and because Codex has no built-in settings for Gemini 4 Argon, it runs the model with default model metadata. These runs therefore only approximate a native setup. They cover the default tests of 444 of the 456 core tasks, all except the 12 EmbodiedSWE tasks. Each group of bars in Figure 15c compares a model's runs on the tests that all of them have graded, and we test differences with a paired exact McNemar test.

Further results of the controlled runs. Table 5 gives the success rates of each run by setting. On the `Realtime` tests, resets help more than the state query, and the two together come within 7 points of the `Privileged` Setting. Without the success signal, the success rate stays within a few points of the baseline in both settings.

The agent uses the saved states to retry. With unlimited resets, only 22% of its successes came in the first episode, and 58% came in an episode that started from a saved state. The saved states let the agent return to the moment before a hard stage instead of repeating the whole task. The state query helps most where the scene is hard to measure from cameras, as in iSpatialGym, RoboDojo and BEHAVIOR-1K, and it changes little in Embodied Olympics - Fun and DexHandExam.

Table 5 | Success rate (%) of GPT-6 Astra in Codex in the controlled runs, on the tests that the baseline and all runs with added features have graded. The `Privileged` row uses the `Privileged` tests of the same tasks (391, 341 and 50 tasks). The last two rows compare the run without the success signal with the baseline on the tests that both have graded (456, 403 and 53).

Run	All (456)	`Standard` (403)	`Realtime` (53)
`Standard` (baseline)	33	33	30
+ object-state query	47	48	40
+ unlimited resets and saved states	52	50	64
+ both	66	64	77
`Privileged`	77	77	84
Baseline, with success signal	33	33	30
No success signal	33	34	26

Table 6 | Success rate (%) of each model in each harness, on the core tests that every run of that model has graded. The first number counts the final verdict, and the second counts first-episode success. The model's own harness is marked with an asterisk.

Model (tests)	Codex	Claude Code	Antigravity CLI
GPT-6 Astra (444)	33 / 25*	25 / 20	–
Claude Opus 5.5 (444)	24 / 19	23 / 19*	–
Gemini 4 Argon (444)	30 / 25	25 / 24	19 / 18*

Further results of the harness comparison. Table 6 gives the success rate of each model in each harness, counted by the final verdict and by first-episode success, that is, success in the first episode before any reset (Section C.2). Gemini 4 Argon does better in both other harnesses than in its own (paired exact test, $p < 0.01$ for each), and most of its gain comes from BEHAVIOR-1K.

For GPT-6 Astra, the difference between Codex and Claude Code is no longer significant when only the first episode counts, so this difference comes mostly from episodes after a reset. For Claude Opus 5.5, the difference is not significant under either count. These runs are reported separately from the leaderboard of Figure 7.

H. Failure Labels

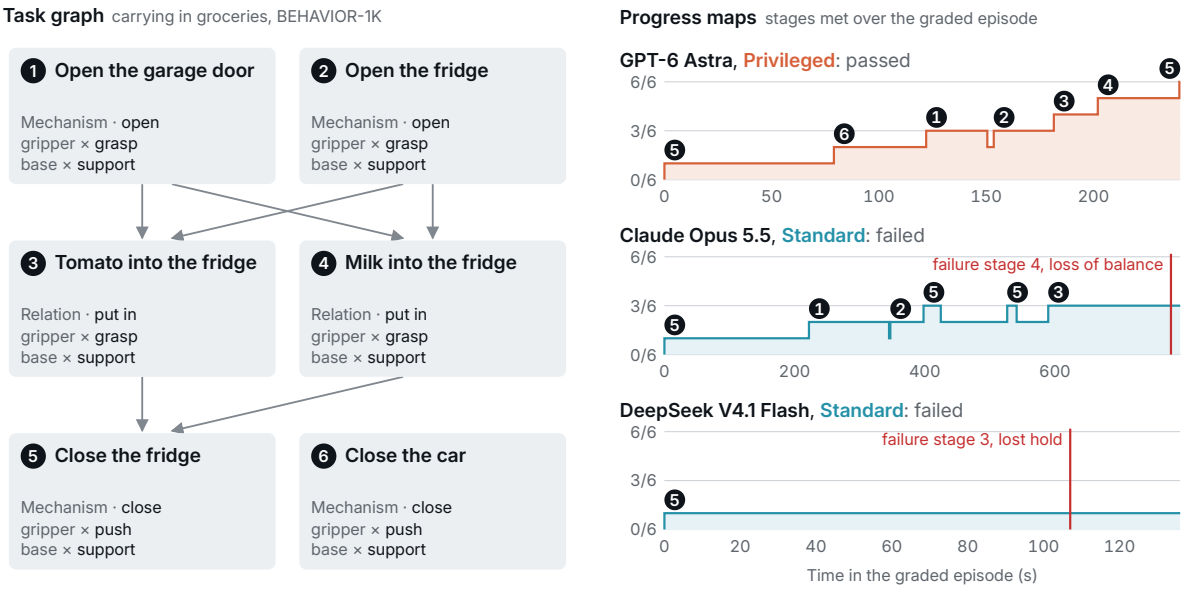
This section describes the stage-level failure labels of Section 3. Section H.1 explains how they were made, Section H.2 gives the taxonomy, and Section H.3 lists the excluded attempts. Section H.4 breaks the failures down by suite, robot body, agent and setting. Section H.5 gives further statistics, in the order of Sections 3.2 to 3.4, and the limitations of the labels. Section H.6 describes the case studies of the figures in full, with further examples. The labels are released with the attempts at `huggingface.co/datasets/MLL-Lab/efe-failure-analysis`.

H.1. Labeling Procedure

Task graph. For each of the 427 labeled core tasks, the labeling agent, GPT-6.1 Sol in Codex, wrote one graph of stages from the task instruction, the verifier's code and a successful attempt when one exists. A stage is a change in the world that a person would list when describing how to do the task, such as "open the fridge" or "put the first mango in the fridge". Conditions on how well a stage must be done, such as thresholds, alignment or "without touching X", belong to that stage's condition and are not stages of their own. A condition that is already true at the start and must still be true at the end, such as a closed fridge, becomes a stage of its own that is met from the start, so that the progress map shows when an attempt breaks it. Each stage has a stage type, its prerequisite stages, a condition that can be checked on the replay video, and the contact it needs. The contact is a list of pairs of the robot part that touches and how it touches (who × how), one list for each way of doing the stage. The edges of the graph give the order in which the stages must be met for the task to pass; a stage that holds from the start can hold before its prerequisites. The stage types fall into nine categories (Table 7). For example, material stages change the state of a material, such as pouring or cleaning, and body stages move the robot's own body, such as walking or keeping balance. The 427 graphs have 2,411 stages, 5.6 per task on average.

Progress map. For each attempt, the labels give the time intervals during which each stage is met, on the timeline of the graded episode's replay video. The condition of a stage can stop holding after it has been met, so the progress map can go down. Progress was read from the video in 4,933 attempts, from the logs in 206 attempts, most of them without a video, and from both in 2. The remaining 499 attempts, most of them without a submitted trajectory, have no readable progress. An attempt reaches a stage if it meets the stage's condition at some moment or if the stage is its failure stage.

Failure stage, phenomena and root causes. Each failed attempt is assigned at most one failure stage, so that failures can be counted by stage and compared across attempts and agents. A failure stage is located when the labels give one. If an irreversible event occurred, such as a fall or a violation of a task rule that makes the attempt fail, the failure stage is the stage during which the event occurred, unless an earlier stage had already failed permanently. Otherwise, it is the stage with the earliest uncorrected mistake among the stages whose condition was never met. Mistakes that the agent later corrected do not count. The labels also record where the



(a) (b)

Figure 25 | Task graph and progress maps of *carrying in groceries* (BEHAVIOR-1K). Panel (a) gives the stages, the order in which they must be met, and the type and contact of each stage. Panel (b) gives the share of the stages whose conditions hold at each moment of the graded episode, for three attempts. Red lines mark failure stages (Section H.6).

attempt was when it failed: during the stage, between stages, while exploring, before the first stage or at the final success check. Up to two failure phenomena describe what visibly went wrong at the failure stage, and up to two root causes describe why it happened, with a video time or log entry supporting each cause. A time limit, a used-up budget or the agent stopping is never a phenomenon or a cause: the label names what was going wrong when the attempt ended. A root cause is given only when the evidence supports it, and it may stay at the category level "Control" when the subtype cannot be determined, as in 54% of the analyzed failures.

Writer and reviewer. GPT-6.1 Sol in Codex, at high reasoning effort, wrote every graph and label file from the inputs above. Claude Sonnet 5.5 in Claude Code 2.1.302, at medium effort, reviewed each file, with up to two rounds of fixes and a final confirmation round. Three of the 5,640 label files used here remained disputed. The prompts were frozen before the final labeling. Section 3.5 uses a second set of labels, which give the control approach of a sample of attempts (Section B). Agents wrote both sets of labels.

Human oversight. We defined the taxonomy (Section H.2) and settled it in team meetings before the final labeling, and the agents assign its values to each task and attempt. The labels went through several versions. One of us reviewed earlier versions by hand case by case, and the team reviewed them in meetings. Each error we found was written into the prompts as a worked example: the task or attempt, the wrong label, the right label and the rule. Most of the worked examples in the final prompts come from this review, and the rest come from rechecks by agents. The labels used here were then written again with the final prompts. We have not yet checked them manually in a systematic way. One of us inspected part of them informally, without measuring agreement. Our plan for a manual check is given at the end of Section H.5.

Core set covered. The labels were made on the results of 27 October. They cover every core test of that date except the 15 of RoboEngineer, with at most one labeled attempt per agent and test, 5,835 in all. The results of Figure 7 are the final results, which include an update of the core set on 2 November. The update removed 12 tests of Embodied Olympics - Match, where each match family now keeps only its hardest level. It added 50 core tests outside RoboEngineer that have no labels yet: 26 tests of Embodied Olympics - Match and the 24 tests of EmbodiedSWE. In addition, 112 labeled attempts were replaced by reruns, and one was regraded. The analysis keeps a labeled attempt only if it is still the attempt of the final results. It therefore leaves out 195 labeled attempts: the 82 on the removed tests and the 113 that were rerun or regraded, among them 16 failures whose final result is a success. The remaining 5,640 attempts cover 836 tests of 427 tasks, every final core test except the 15 of RoboEngineer and the 50 added tests. The reruns have no labels yet.

H.2. Taxonomy

Tables 7 to 10 list the values of the taxonomy. A root cause can also stay at the category level, such as Control (Section H.1). Stage types let us compare stages across tasks. A stage type names the robot action that directly brings about the stage's change in the world, and the types fall into the nine categories of Table 7. In the task of Figure 25, for example, opening the fridge is a mechanism stage of type open, and putting the tomato in is a relation stage of type put in. The contact of a stage says who touches, such as a gripper, both arms or the mobile base, and how, such as grasping, pushing or supporting the robot's weight. Failure phenomena describe what visibly went wrong at the failure stage. In the same example, Claude Opus 5.5 fails with a loss of balance, and DeepSeek V4.1 Flash fails with a lost hold. Root causes, such as a wrong perception or an unsuitable grasp, describe why a failure happened.

Table 7 | Stage types, by category. The competition stages score, defend and win are labeled but not analyzed stage by stage.

Category	Stage types
Relation	place, put in, take out, gather, stack, insert, assemble, disassemble, screw/unscrew, thread/hang, connect, hand over
Pose	pick up, regrasp, push, pull, flip over, adjust pose, follow path, stabilize
Dynamic	throw, strike, catch, dribble
Mechanism	open, close, rotate, press, switch on/off
Material	fold, flatten, untangle, pour, scoop, cut, clean, stir, draw/write, change temperature, ignite
Body	navigate, traverse, jump, perform acrobatics, hang and swing, maintain balance, maintain pose
Safety	refuse, warn, mitigate
Information	search, inspect, communicate
Competition	score, defend, win

H.3. Excluded Attempts

The seven agents failed 4,429 of their 5,640 labeled attempts. The analyses in Sections 3.2 to 3.4 exclude 558 of these failures, in two groups that overlap in 85 attempts. Table 11 counts them by agent and setting. Both groups are failures under the rules of the exam. They are excluded only from the statistics computed from the failure labels.

Filter cutoffs, 265 attempts. In these attempts the model provider's content filter stopped the agent's session, and the attempt ended or submitted nothing. Gemini 4 Argon accounts

Table 8 | Contact of a stage: who touches and how.

Who	
Fingers/gripper	one hand or gripper touches the object directly, also when the pose of a held tool is graded
Bimanual	two hands or arms act on the stage together
Tool	the robot acts through a held or mounted tool
Legs/torso	feet, legs, head, torso or landing gear on a support surface
Mobile base	wheels or a wheeled base, while moving
No contact	no contact to label, as when flying, observing or reporting
How	
Grasp	the hand closes on the object
Push/sweep	push, nudge or sweep without closing the hand
Press/wipe	press or wipe with pressure on the contact surface
Cradle/scoop	support, catch or scoop the object from below
Hook	hook with a finger or a tool
In-hand adjustment	reorient or reposition the object with the fingers without letting go
Support	the body or base bears weight, as in standing or landing
Avoid contact	pass without touching a named object

Table 9 | Failure phenomena: what visibly went wrong at the failure stage, up to two per attempt.

Category	Phenomenon	Meaning
Contact	Failed to make contact	the contact the stage needs never existed, as in a missed grasp or fingers that never closed on the object
	Lost hold	held and then lost: slipped or dropped while held or carried
	Abnormal contact force	visible or sensed wrong force, such as a part not fully inserted or a latch not engaged
	Failed to release	the object stays in the hand or tool when the stage needs it to leave
	Unstable after release	after a planned release the object falls over, rolls away or bounces
	Drags the base	manipulating a part moves the body that should stay still
Motion	Inaccurate pose	the hand, tool or object misses the required position or orientation
	Obstruction/collision	blocked by contact, or knocking over or displacing objects
	Out of reach	evidence of a reachability or joint limit
	Dynamic timing error	contact with or release of a moving object too early or too late
Whole body	Loss of balance	any robot loses support or tips over
	Gait failure	walking, running or jumping falls short without a fall
Constraint	Constraint violation	a task rule broken, with no more specific phenomenon
Reporting	Missing report, incorrect report	a graded information act, such as a warning or a required answer, missing or wrong
Not attempted	Not attempted	a stage the graded attempt needed was never started

for 252 of them, Gemini 3.8 Flash for 11, and Qwen3.8 Max and GLM 5.3 Flash for one each. We identify them by the free-text summaries of the labels, which name the interruption, and spot-checked a sample against the run records. For a blocked turn, the run records keep only the harness's generic message and omit the provider's reason, so what triggered a block cannot be read from them. Antigravity CLI, the harness of both Gemini models, ends the session after a blocked reply, whereas Codex sends the request again and Claude Code asks the model to continue (Section E.3). This exclusion removes every attempt that the filter interrupted, including attempts cut off after the robot had already failed. In 74 of the 265 attempts, the labels also give a failure stage with a physical phenomenon, and the column "with failure stage" of Table 11 counts them. Of these 74, 71 are Gemini 4 Argon's, 60 of them in the `Standard` Setting.

Table 10 | Root causes: why the failure happened, up to two per attempt, each with its evidence.

Category	Root cause	Meaning
Cognition	Task understanding	the behavior shows the task was misread
	Reasoning/planning	wrong goal, stage order or overall strategy
	Perception	the robot acted on a wrong estimate of where or what something is
	Safety judgment	records show a hazard was not recognized or misjudged
	False completion belief	a false belief that the task was done affected stopping or decisions
Control	Grasp and contact design	the contact geometry or load layout does not suit the stage
	Trajectory or IK	goal and contact plan are feasible, but no trajectory or IK solution is found or followed
	Feedback control	tracking or stabilization under feedback fails
	Parameter tuning	parameters shown to be wrong
Delivery	Delivery operation error	recording, saving or checking the submitted answer went wrong, such as recording without a reset
Task or environment	Task or environment	every legal solution is ruled out, or an event the agent cannot react to

Table 11 | Failed attempts excluded from the failure analysis, by agent and setting. P, S and R stand for the `Privileged`, `Standard` and `Realtime` settings. The column "with failure stage" counts filter cutoffs whose labels also give a failure stage with a physical phenomenon. The column "Excluded" counts filter cutoffs and missing trajectories, and an attempt in both groups counts once. GPT-6 Astra, Claude Opus 5.5 and DeepSeek V4.1 Flash are not listed: they have no filter cutoffs and 1, 2 and 2 missing trajectories in the `Privileged` Setting.

Agent	Setting	Failed	Filter cutoffs	with failure stage	Missing trajectory	Excluded
Gemini 4 Argon	P	221	109	9	135	164
	S	331	118	60	0	118
	R	49	25	2	0	25
Gemini 3.8 Flash	P	284	4	1	100	101
	S	384	7	2	0	7
Qwen3.8 Max	P	307	1	0	100	100
GLM 5.3 Flash	P	325	1	0	38	38
All seven agents		4,429	265	74	378	558

Keeping these attempts would add at most 74 failures to the 3,871 analyzed, and fewer if some of them also lack a trajectory. Because almost all of them are Gemini 4 Argon's, its failure profile in Figure 15a and Table 19 is the statistic that this choice affects most.

Missing trajectories, 378 attempts. In the `Privileged` Setting the agent's answer is a recorded trajectory that the verifier replays. In these attempts the agent never submitted one, and the verifier recorded neither a trajectory nor a replay. The label summaries give the usual reasons: the agent was still testing in its own session when the time ran out, or its script for saving the answer did not finish. We exclude these attempts because they leave no submitted trajectory or graded replay to label. Apart from the 85 that are also filter cutoffs, a missing trajectory is the agent's own failure. Gemini 4 Argon has 135 such attempts, of which 80 are also filter cutoffs. Gemini 3.8 Flash and Qwen3.8 Max have 100 each, and GLM 5.3 Flash has 38. They are spread over more than 20 suites, so no single suite explains them. For Gemini 3.8 Flash and Qwen3.8 Max they are about a third of all `Privileged` failures, 100 of 284 and 100 of 307.

Failures without readable progress, 118 attempts. Besides the 558 failures above, the stage-level statistics, namely the failure rates of stages and the predictions of the failure stage, exclude 118 failed attempts whose labels give no stage progress, because neither the video nor the log shows when the stages held. These statistics therefore use 4,964 attempts, passed and failed: the 5,640 labeled attempts without the 558 excluded failures and these 118. The 118 attempts remain in the statistics that do not need stage progress, such as the failure profiles of Table 19. The 82 of them with a located failure stage, that is, a failure stage that the labels give, also remain in the concentration of failure stages and in the common failure stage.

H.4. Failures by Suite, Robot Body, Agent and Setting

Figures 26 and 27 break the 3,871 analyzed failures down by suite, robot body, agent and setting. Each row splits the group's failures among the labels in four blocks, so its values are shares of the failures and should not be read as failure rates. The first block says how far the failed attempt got. Failures on tasks with a single stage form their own column. For longer tasks, the block uses the attempt's peak progress, the largest share of the task's stages met at one moment, counting stages already met at the start. Its columns are no stage, more than none but under half, at least half but not all, all stages, and not readable. Peak progress is not readable when the status of at least one stage could not be read, and this column holds 349 failures, among them 115 of the 118 failures without any readable progress (Section H.3). The second block gives the category of the failure stage. The third and fourth blocks give the failure phenomena and the root causes. A failure can have up to two of each, so these two blocks can add up to more than 100%. The rows pool the three settings, except the rows of the settings themselves. The robot body is the kind of body of the task's robot, as in Table 2. RoboEngineer has no labels and no row.

Shares are rounded, <1 marks a share above zero but below 1%, and empty cells are zero. Gray labels mark groups with fewer than 20 failures. The phenomena and root causes are those of Tables 9 and 10, and None means that a failure has no label in the block. Other contact pools the four rarer contact phenomena, and Balance or gait pools loss of balance and gait failure. Other covers out of reach, reporting and defects that the taxonomy does not name, such as gaps in a painting or folds left in a cloth. No analyzed failure lies at a safety or information stage or has a task or environment root cause, so these columns are omitted.

By suite. Figure 26 gives the rows of the suites. The categories of the failure stages differ across suites, and in some suites all failures lie at one kind of stage. Every RoboPaint failure is at a material stage, every Embodied Olympics - Match failure is at a competition stage, and every DexToolBench failure with a located failure stage is at a pose stage. In Embodied Olympics - Match and DexToolBench, every stage of every task is of that category. How far the failed attempts get also differs. In DexToolBench and KinDER, most failed attempts never meet a stage of the task, while in RoboPaint almost every failed attempt meets at least half of the stages. In Embodied Safety Exam and ManiGuard, some failed attempts meet all stages at one moment. In Embodied Safety Exam, most of these attempts have a constraint violation. Failures at the same kind of stage can still differ between suites. In both RoboLab and RoboCasa365, most failures lie at relation stages, but most RoboLab failures are lost holds, and most RoboCasa365 failures lack the contact the stage needs. Among the suites with at least 20 failures, a cognitive root cause is most frequent in RoboPaint, Embodied Safety Exam, iSpatialGym and Embodied Olympics - Match, at 30 to 38% of their failures, compared with less than 7% in seven others, such as HumanoidBench, MyoSuite and RoboLab.

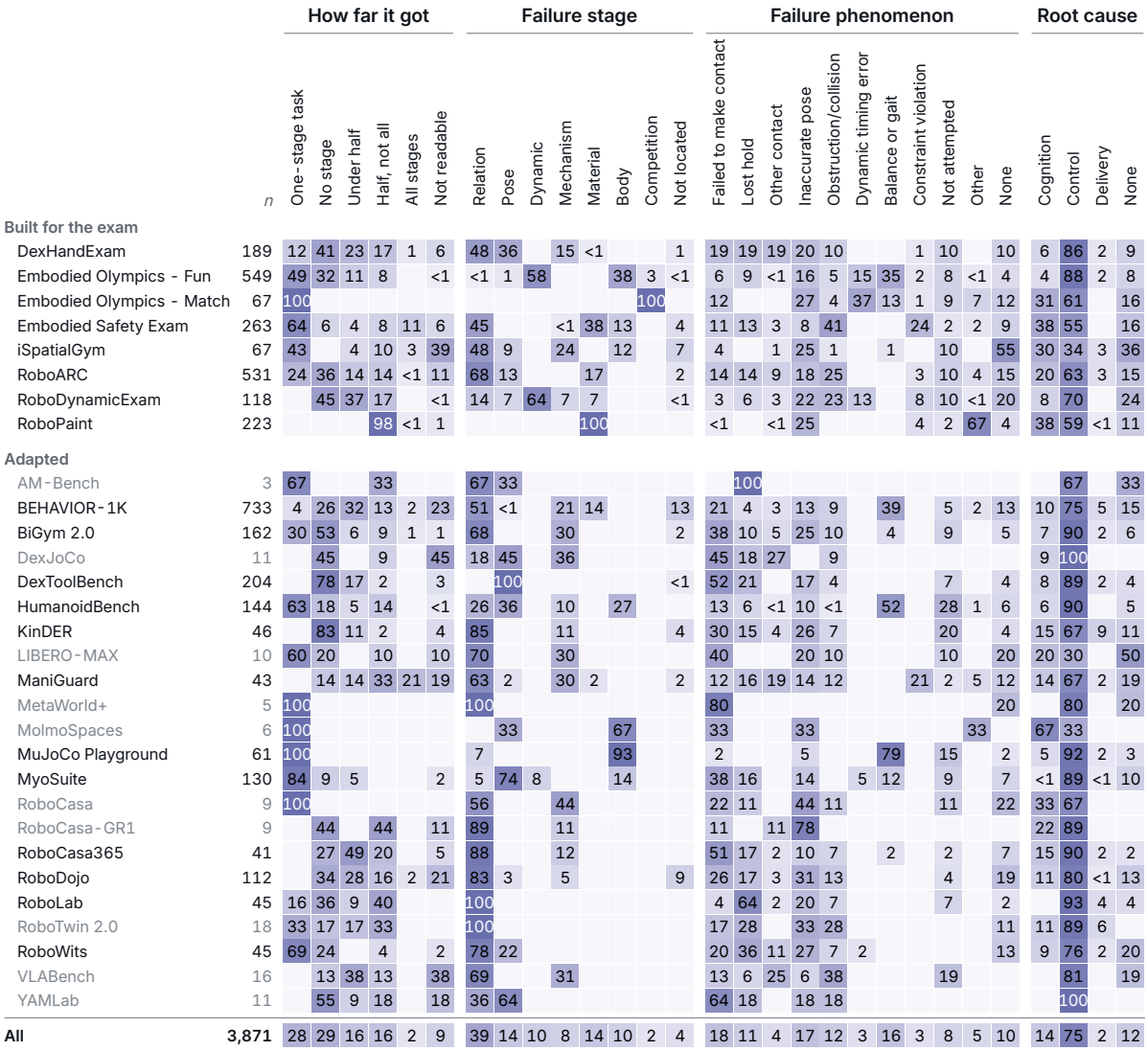
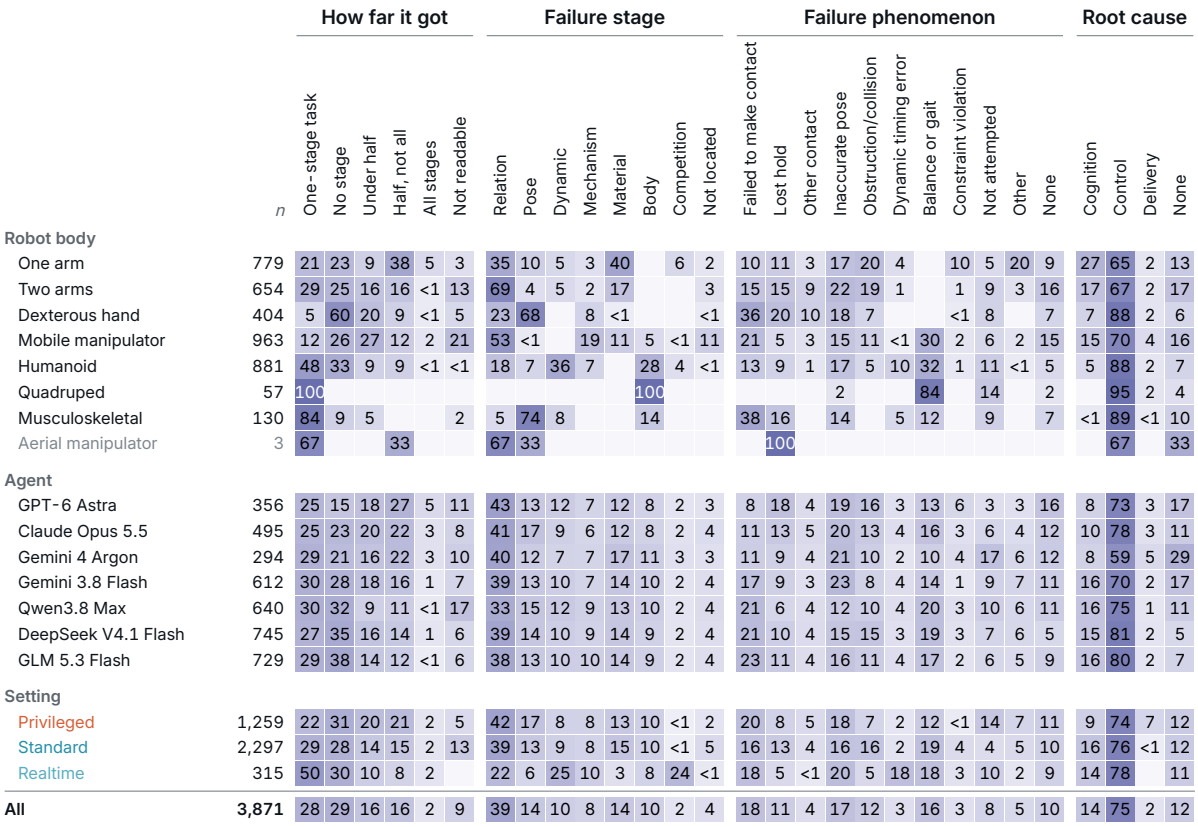


Figure 26 | Failures by suite. Each cell gives the share (%) of the suite's analyzed failures in the column, with the three settings pooled, and *n* is the number of failures. A failure can have up to two phenomena and up to two root causes, so these two blocks can add up to more than 100%.



		How far it got						Failure stage								Failure phenomenon												Root cause			
	n	One-stage task	No stage	Under half	Half, not all	All stages	Not readable	Relation	Pose	Dynamic	Mechanism	Material	Body	Competition	Not located	Failed to make contact	Lost hold	Other contact	Inaccurate pose	Obstruction/collision	Dynamic timing error	Balance or gait	Constraint violation	Not attempted	Other	None	Cognition	Control	Delivery	None
Robot body																															
One arm	779	21	23	9	38	5	3	35	10	5	3	40		6	2	10	11	3	17	20	4		10	5	20	9	27	65	2	13
Two arms	654	29	25	16	16	<1	13	69	4	5	2	17			3	15	15	9	22	19	1		1	9	3	16	17	67	2	17
Dexterous hand	404	5	60	20	9	<1	5	23	68		8	<1			<1	36	20	10	18	7			<1	8		7	7	88	2	6
Mobile manipulator	963	12	26	27	12	2	21	53	<1		19	11	5	<1	11	21	5	3	15	11	<1	30	2	6	2	15	15	70	4	16
Humanoid	881	48	33	9	9	<1	<1	18	7	36	7		28	4	<1	13	9	1	17	5	10	32	1	11	<1	5	5	88	2	7
Quadruped	57	100											100						2			84		14		2		95	2	4
Musculoskeletal	130	84	9	5			2	5	74	8			14			38	16		14		5	12		9		7	<1	89	<1	10
Aerial manipulator	3	67			33			67	33								100											67		33
Agent																															
GPT-6 Astra	356	25	15	18	27	5	11	43	13	12	7	12	8	2	3	8	18	4	19	16	3	13	6	3	3	16	8	73	3	17
Claude Opus 5.5	495	25	23	20	22	3	8	41	17	9	6	12	8	2	4	11	13	5	20	13	4	16	3	6	4	12	10	78	3	11
Gemini 4 Argon	294	29	21	16	22	3	10	40	12	7	7	17	11	3	3	11	9	4	21	10	2	10	4	17	6	12	8	59	5	29
Gemini 3.8 Flash	612	30	28	18	16	1	7	39	13	10	7	14	10	2	4	17	9	3	23	8	4	14	1	9	7	11	16	70	2	17
Qwen3.8 Max	640	30	32	9	11	<1	17	33	15	12	9	13	10	2	4	21	6	4	12	10	4	20	3	10	6	11	16	75	1	11
DeepSeek V4.1 Flash	745	27	35	16	14	1	6	39	14	10	9	14	9	2	4	21	10	4	15	15	3	19	3	7	6	5	15	81	2	5
GLM 5.3 Flash	729	29	38	14	12	<1	6	38	13	10	10	14	9	2	4	23	11	4	16	11	4	17	2	6	5	9	16	80	2	7
Setting																															
Privileged	1,259	22	31	20	21	2	5	42	17	8	8	13	10	<1	2	20	8	5	18	7	2	12	<1	14	7	11	9	74	7	12
Standard	2,297	29	28	14	15	2	13	39	13	9	8	15	10	<1	5	16	13	4	16	16	2	19	4	4	5	10	16	76	<1	12
Realtime	315	50	30	10	8	2		22	6	25	10	3	8	24	<1	18	5	<1	20	5	18	18	3	10	2	9	14	78		11
All	**3,871**	28	29	16	16	2	9	39	14	10	8	14	10	2	4	18	11	4	17	12	3	16	3	8	5	10	14	75	2	12

Figure 27 | Failures by robot body, agent and setting, in the same format as Figure 26. Here too, the phenomena and the root causes can add up to more than 100%.

By robot body. The upper rows of Figure 27 give the robot bodies. Each kind of body comes from a few suites: three quarters of the failures of mobile manipulators are BEHAVIOR-1K's, and all those of quadrupeds come from MuJoCo Playground. The bodies also differ in their mix of settings. These rows therefore do not isolate the effect of the body. Loss of balance is the most common phenomenon for humanoids, quadrupeds and mobile manipulators. Failure to make contact is the most common for dexterous hands and for the musculoskeletal models of MyoSuite, and most failed attempts with a dexterous hand never meet a stage of the task. For robots with one or two arms, collisions and inaccurate poses are the most common phenomena apart from task-specific defects.

By agent and setting. The lower rows of Figure 27 give the agents and the settings. For every agent, relation stages hold the largest share of the failures, from 33 to 43%, and the shares of the other categories differ by about five points or less between agents. The agents differ in how far they get and in what goes wrong, as in Section 3.4. The failures of the four weaker agents more often meet no stage of the task, and more often lack the contact that the stage needs. Gemini 4 Argon's failures more often have the not-attempted label and more often have no root-cause label. Between the settings, `Realtime` failures more often lie at competition and dynamic stages and more often involve a timing error. `Privileged` failures more often have the not-attempted label. Collisions and losses of balance are each more common in `Standard` failures. The settings cover different mixes of tests, so these comparisons do not isolate the effect of the setting.

H.5. Further Statistics

Failure rate of a stage. The failure rate of a stage is the number of attempts that failed at the stage divided by the number of attempts that reached it. Attempts that never reached the stage are not counted, so the rate measures whether an attempt that gets to the stage fails there. For the rate of a stage type, each stage of that type counts separately, so an attempt that reaches three put-in stages adds three stages. A stage counts as failed when it is the attempt's failure stage, and as met when its condition held at some moment, even if it already held at the start or stopped holding later. To compare the two settings without mixing suites, we compute the rate within each suite and average over the suites in which attempts reach the stage at least eight times in each setting, weighted by the smaller of the two counts. Tables 12 and 13 give the results.

Table 12 | Failure rate of each stage type in percent, compared within suites. P is the `Privileged` and S the `Standard` Setting. The Suites column gives the number of suites compared, and the Weight column the sum of the smaller counts. Types compared in only one suite are less reliable. Close includes stages that already hold at the start, such as a closed fridge, which count as reached even when the attempt never acts on them.

Stage type	Suites	Weight	P	S	S − P
Adjust pose	2	25	31	83	+52
Fold	1	25	39	88	+49
Clean	2	56	36	82	+46
Assemble	2	39	39	82	+43
Strike	2	193	12	52	+40
Throw	1	47	60	100	+40
Catch	2	71	21	59	+39
Dribble	1	38	14	50	+36
Flatten	1	30	44	80	+36
Insert	2	39	28	59	+30
Draw/write	1	140	64	93	+30
Jump	1	109	37	62	+26
Thread/hang	3	162	25	49	+25
Disassemble	2	42	47	70	+23
Traverse	2	85	26	49	+23
Put in	11	587	28	51	+23
Follow path	3	421	21	43	+23
Open	4	211	30	51	+21
Place	8	354	22	36	+14
Pick up	3	56	37	50	+13
Close	5	210	5	8	+3
Press	1	56	0	0	0

The `Privileged` and `Standard` settings. The two settings differ in two conditions that matter for Section 3.2. In the `Privileged` Setting the agent reads the exact pose of every object and can reset the simulator without limit, so it can compute a precise motion from the full state, test it and try again within the time limit before it submits an answer. In the `Standard` Setting it must estimate poses from camera images and depth, the episode is graded as it runs, and the task limits the number of resets, so a precise stage is attempted on estimates and an error can remain in the graded episode. The comparisons of Section 3.2 concern the stages of a task. They do not concern how the agent controls the robot, which changes with the setting (Section 3.5). We split the stage types into two groups by their definitions in the taxonomy (Table 7). A stage type is precise if it needs a tight fit, a mated or threaded relation, a target pose or path, or contact with a moving object. The 13 precise types are insert, assemble, screw/unscrew, thread/hang, connect,

Table 13 | Failure rate by contact in percent, compared within suites as in Table 12. Bimanual stages are the hardest in both settings, and the failure rates of in-hand adjustment and of cradling or scooping rise the most.

	Value	Suites	Weight	P	S	S − P
Who	Legs/torso	5	697	27	61	+34
	Bimanual	10	576	40	70	+30
	Tool	10	669	24	53	+28
	Fingers/gripper	25	2,636	24	47	+24
	Mobile base	3	842	24	43	+19
	No contact	3	138	10	25	+14
How	In-hand adjustment	3	49	29	81	+51
	Cradle/scoop	7	187	24	61	+37
	Avoid contact	2	184	20	47	+28
	Support	8	1,488	26	51	+25
	Hook	4	149	31	55	+24
	Grasp	23	2,619	24	47	+23
	Press/wipe	5	289	36	59	+23
	Push/sweep	13	476	15	30	+15

adjust pose, follow path, stabilize, regrasp, throw, strike, catch and dribble. A stage type is coarse if it needs only to reach a region, a container, a grasp or the open or closed state of a mechanism. The 12 coarse types are place, put in, take out, gather, pick up, push, pull, open, close, press, switch on/off and navigate. All other types, such as the material types and the body types other than navigate, are in neither group. Section 3.2, Section E.1 and Table 14 use these groups.

Compared within suites as in Table 12, the failure rate rises from 22% to 54% at precise stages, in 8 suites, and from 22% to 40% at coarse stages, in 24 suites. With suite fixed effects and intervals clustered by task, the precise stages rise by 15 points more than the coarse stages (95% interval 9 to 22). These comparisons use different attempts in the two settings. When the same agent reaches the same stage of a task in both settings, the failure rate rises by 21 points at precise stages and by 17 points at coarse stages, and the difference between the two groups is not significant (95% interval −1 to +10 points). Follow-path stages, most of them waypoints of DexToolBench, make up 38% of the precise stages in this comparison, and their rate rises by 11 points. Without them, the rate at precise stages rises by 28 points, 11 points more than at coarse stages (95% interval 5 to 18). Among the types compared in at least two suites in Table 12, the rates of striking, assembling, adjusting a pose and catching rise by 31 to 39 points in this paired comparison, and those of closing, placing and picking up rise by 2 to 15 points.

The labels cannot show how much of the loss comes from perception and how much from control (see the limitations at the end of this section). The labels also cannot show how much comes from the missing state and how much from the missing resets. The controlled runs of Section G separate the two at the level of success rates: on these tests, adding either one alone closes a third to a half of the gap. Joined with the labels, they also separate the two by stage type.

State and resets at precise stages. We join the controlled runs of GPT-6 Astra to its labeled failures in the `Standard` and `Realtime` settings, using the final results of Section G. Of these failures, 249 have a located failure stage. Of them, 80 are at precise stages and 97 at coarse stages, split as above. Of the rest, 40 are at material stages and 32 at other stages. Table 14 gives the share of them that each run solves.

Each run makes one attempt per test, so some failures would turn into successes simply

by trying again. The run without the success signal approximates such a second attempt. It is not an exact repeat, since the agent must end the episode itself, and its success rate is within 4 points of the baseline's (Table 5). It solves 19% of the failures at precise stages and 15% of those at coarse stages.

At precise stages (Table 14), 21 failures are solved by exactly one of the state-query run and the reset run, and the reset run solves 17 of them (exact McNemar test, $p = 0.007$).

The logs show how the agent used each feature. Of the 29 precise failures that the resets solve, 28 succeed only after a reset or a restored state. The number of state-query calls was recorded for all 80 precise failures, and the agent called the query in every one of them. The run with both solves more of these failures than the run with resets alone (15 failures that only one of the two runs solves, 13 of them by the run with both, $p = 0.007$).

Let the recovery of a run be its share of solved failures minus that of the run without the success signal, both on the same failures. On the precise failures that have a `Privileged` test, the recovery of the run with both features is 57% of the recovery of the `Privileged` Setting, with a 95% interval from 37% to 74% over failures. The `Privileged` Setting solves 19 more precise failures than the run with both. Stages that follow a path, thread or assemble account for 14 of these 19 failures. At striking and catching stages, the run with both solves exactly the failures that the `Privileged` Setting solves.

These comparisons have limits. Precise and coarse failures come from different suites. Most precise failures are in Embodied Olympics - Fun, DexToolBench and RoboARC, and 35 of the 97 coarse failures are in BEHAVIOR-1K. Leaving out any one suite keeps the direction of each comparison between runs and between the two groups of stages. The join keeps a failure only if its labeled attempt is still the leaderboard attempt. This leaves out 29 labeled failures whose tests were rerun after the labels were made, 9 of them in RoboDojo, where the leaderboard attempts were rerun under a new episode rule. In a few runs the agent's session ended with an error, for example because the API account ran out of credits. Leaving out the 16 failures with such a run changes each share by at most 2.3 points. The intervals and tests treat the failed tests as independent, although a `Standard` and a `Realtime` test of the same task can both appear.

Table 14 | Share (%) of GPT-6 Astra's labeled failures in the `Standard` and `Realtime` settings that each controlled run solves, by type of failure stage, with 95% intervals over failures. Material stages include drawing, pouring and cleaning. The `Privileged` row uses the `Privileged` test of the same task, which exists for 80 precise, 86 coarse, 32 material and 32 other failures.

Run	Precise (80)	Coarse (97)	Material (40)	Other (32)
No success signal	19 (10–28)	15 (8–23)	30 (18–45)	16 (3–28)
+ object-state query	20 (11–29)	34 (25–43)	38 (22–52)	28 (12–44)
+ unlimited resets and saved states	36 (26–46)	46 (36–57)	48 (32–62)	31 (16–47)
+ both	50 (40–61)	67 (58–76)	58 (40–72)	41 (25–59)
`Privileged`	74 (64–82)	60 (50–71)	88 (75–97)	72 (56–88)

Concentration of failure stages. The concentration of a test measures how much the agents that failed it agree on the stage at which they failed. We measure it with the Simpson index: for a test that $n \geq 2$ agents failed at located stages, with n_k of them at stage k, it is $D = \sum_k n_k(n_k - 1)/\big(n(n-1)\big)$, the share of pairs of these agents that failed at the same stage. We average D over the tests of a setting whose tasks have at least two counted stages, and resample tests 2,000 times for the 95% interval. If each agent's failure stage is sampled independently with probabilities q_k,

the expected value of D is $\sum_k q_k^2$. The random-stage value, the gray bars of Figure 13a, assumes that each agent fails at a random stage but keeps one property of the observed failures: where in a task they usually fall. First, we order the K stages of each task graph topologically, breaking ties by their order in the task file, so that the i-th stage covers the interval $[(i-1)/K, i/K]$. Second, over all failures in the other tasks of the same setting that have at least two stages, we build a ten-bin histogram of the relative position $(i-1/2)/K$ of the failure stage. Third, each stage receives the mass of this histogram over its interval, and these masses are the probabilities q_k. A simpler uniform baseline, $q_k = 1/K$ for a task with K stages, gives 28% in the `Privileged` Setting, 29% in the `Standard` Setting and 30% in the `Realtime` Setting. This baseline ignores that early stages fail more often, so the main text does not use it. The value of D for the actual failure stages exceeds the random-stage value in 73% of `Standard` and 65% of `Privileged` tests, compared with about 43% and 38% if the failure stages were sampled from the random-stage probabilities q_k. The 95% interval of the mean difference is 21 to 28 points in the `Standard` Setting and 14 to 23 points in the `Privileged` Setting. In the `Realtime` Setting, which Section 3.3 excludes because only 27 tests qualify, the concentration is 52%, compared with 45% for random failure stages, and the interval of the difference, −3 to +17 points, includes zero. On tasks with at least three stages, the concentration in the `Standard` Setting is 51% over 191 tests, compared with 27% for random failure stages and 18% for the uniform baseline. The comparison with random failure stages assumes that, apart from where in a task failures usually fall, the failure stages of different agents are independent. Because each agent makes one attempt per test, every pair counted in D compares two different agents. For the three strongest agents alone, we use the `Standard` tests that at least two of them failed at located stages, and we build the histogram from their failures on the other tasks. Over these 189 tests, the concentration is 57% (95% interval 50% to 63%), compared with 31% for random failure stages, and the 95% interval of the difference is 20 to 32 points. With the histogram of all seven agents, the random-stage value is 36%. Pairs that agree at a stage after the first contribute 30 points of the 57%, compared with 13 for random failure stages. In 57% of these tests, exactly two of the three agents have located failures, so the concentration on each such test is either 0 or 1. Among the located `Standard` failures, 65% of those of the four weaker agents are at the first stage, compared with 40% for the three strongest. In the `Privileged` Setting, 63 tests qualify, and 81% of them have exactly two located failures among these agents. The concentration is 38%, compared with 25% for random failure stages, and the 95% interval of the difference is 3 to 24 points.

Shared failure stages. We count the `Standard` tests with at least three stages that four or more agents failed. In 68 of these 175 tests, at least 80% of the failures fall on one stage. Table 15 lists one example per suite, chosen among the 68 so that most failures came after another stage had been reached. The `Realtime` test of the same RoboDynamicExam task as Figure 14 shows the same shared stage: the content filter stopped Gemini 4 Argon's attempt, and the other six agents all failed at the recatch.

The common failure stage. A test has a single most common failure stage when one stage holds more of its located failures than any other. In all, 195 `Privileged` and 250 `Standard` tests have one. For the statements of Section 3.3 built on it, the stages of a task are ordered by their depth in the task graph, with ties broken by the order in the task file. To ask whether stage types predict this stage, we estimate, for each setting, how often each stage type fails once an attempt reaches it, using the tasks other than the test's own. Each rate is shrunk toward the setting's mean rate with the weight of ten attempts. A second variant uses the rate of each type within the test's own suite, shrunk toward the pooled rate of the type in the same way. In a sequential

Table 15 | Examples of shared failure stages in the `Standard` Setting, one per suite. The Stages column gives the number of stages of the task. The column "Failed here" gives the failed attempts that failed at this stage over all failed attempts with a located failure stage, without the attempts excluded in Section H.3. The RoboDynamicExam row leaves out three attempts: the filter cutoff of Gemini 4 Argon and the unlabeled reruns of Claude Opus 5.5 and GLM 5.3 Flash. All three also fail at the recatch (Section H.6), so all seven agents fail there in Figure 14.

Suite	Task	Stages	Shared stage	Type	Failed here
BEHAVIOR-1K	clean up branches and twigs	5	first branch into the bin	put in	7/7
Embodied Safety Exam	three spice jars onto a tray	3	second jar onto the tray	place	7/7
RoboARC	thread a fiber bundle through rings around a post	18	first fiber through its ring	thread/hang	7/7
RoboCasa365	defrost meat in the microwave	4	container into the microwave	put in	6/6
DexHandExam	install the front guard of a fan	5	seat the front guard	assemble	5/6
RoboDynamicExam	catch, return through a slot, deliver to a moving cart	3	recatch after the rebound	catch	4/4
RoboDojo	sweep small objects into a dustpan	6	first earphone into the dustpan	put in	4/5

model with failure rates $h_1, \ldots, h_K$ in stage order, the chance that the failure stage is stage k is proportional to $h_k \prod_{j<k}(1 - h_j)$. The predicted stage is the one with the highest chance, and the concentration expected under these rates is $\sum_k q_k^2$ for the normalized chances q_k. To measure how strongly getting past the common stage predicts passing the test, we take for each attempt the most common failure stage of the other agents' failed attempts on the same test, when at least two of them failed there and no other stage ties with it. We count the attempt as past this stage when its progress map shows the stage met at some moment. Two agents share a failure phenomenon when their labels name at least one phenomenon in common, and pairs in which either attempt has no phenomenon are excluded. Agents that fail at the same stage need not fail there in the same way: in the `Standard` Setting, two agents that failed at the same stage share a failure phenomenon in 40% of pairs, as in the RoboDynamicExam task of Figure 14 (Section H.6).

Table 16 compares the common failure stage with what the stage types predict, with rates from the other tasks and with rates from the same suite, and Section 3.3 reports the rates from the other tasks. A prediction from the stage types finds the common stage in 78% of the tests, and always naming the first stage finds it in 70%. On tasks with at least two types of stages, the two rules find it in 71% and 60%. The concentration it predicts is 36% in the `Privileged` and 48% in the `Standard` Setting, against 51% and 61% observed. Table 17 shows how strongly getting past the common stage predicts passing the test. Part of this link follows from the order of the stages, since an attempt that reached more stages is closer to the end. The settings still differ: in the `Privileged` Setting most attempts that pass the common stage go on to pass the test, while in the `Standard` Setting fewer than half do, and most of the others fail at a later stage.

The common failure stage across settings. In all, 176 tasks have a single most common failure stage in both the `Privileged` and the `Standard` Setting, and it is the same stage in 143 of them, or 81%. For a baseline, we sample, for each of these tasks and each setting, as many failure

Table 16 | Stage types and the common failure stage, in percent. Left: how often a rule names the most common failure stage of a `Standard` test, over the 250 `Standard` tests that have one and over those among them whose tasks have at least two types of stages. Right: the concentration of failure stages, observed and expected if each agent failed independently at the given rates.

Naming rule	All	Two or more stage types
Stage-type rates, other tasks	78	71
Stage-type rates, same suite	79	73
Always the first stage	70	60

Concentration	`Privileged`	`Standard`
Observed	51	61
Stage-type rates, other tasks	36	48
Stage-type rates, same suite	40	52
Random failure stages	33	37

Table 17 | Getting past the common failure stage, in percent. For each attempt, the common stage is the most common failure stage of the other agents on the same test. The table covers the 977 `Privileged` and 1,335 `Standard` attempts that have such a stage and a readable status at it.

	`Privileged`	`Standard`
Passed the test, all attempts covered	30	9
Passed the test, attempts past the common stage	67	34
Passed the test, past a common stage that is not the first	82	40
Failed exactly at the common stage, attempts that never met it	80	84

stages as the task has located failures in that setting. The samples are independent, with the stage probabilities q_k^{P} and q_k^{S} of the random-failure-stage value above and the stages in depth order. We keep the samples in which both settings have a single most common stage. Averaged over the tasks, the two most common stages then agree with probability 51%. In 102 of the 176 tasks the common stage is the first stage in both settings, so much of the agreement comes from tasks that fail at their first stage. On the other 74 tasks the common stage is still the same in 55% (95% interval 45% to 66% over tasks). Under the same baseline, leaving out the samples in which both common stages are the first stage, the average over these 74 tasks is 9%. Where the two common stages differ, the one of the `Standard` Setting lies earlier in the task in 26 tasks and later in 7. These counts describe where failures fall. They do not show that an agent follows the same plan in the two settings.

Pooled failure rates. Table 18 gives the rates behind Figure 13b, for each setting and for both together: every attempt that reached a stage counts, whatever its suite. The stage categories are pooled over all their stage types, and the rows list the stage types found in at least three suites and reached at least 30 times in the two settings together.

Interpreting the stage rates. The rates of Figures 13b and 28 show an association between kinds of stages and failure. They cannot be read as causal effects, for three reasons. First, each kind of stage occurs mostly in particular suites and with particular robots: the part that makes contact depends on the robot, so legs and torso belong to legged robots, and most material stages come from RoboPaint and Embodied Safety Exam. Second, stages differ in how much the robot must do in them. A relation stage usually covers grasping, carrying and releasing its object, because the task graphs list picking up as a stage of its own in only 10 of the 229 tasks with relation stages. A rate per stage therefore compares units of different size. Relation stages are 41% of the failure stages counted here, against their 38% share of the stages that attempts reach. Third, the taxonomy records what a stage changes but not how much contact it needs.

Table 18 | Failure rate (%) of each stage category and of the stage types found in at least three suites, in the `Privileged` Setting, in the `Standard` Setting and in both together, with the stages at which attempts failed over the stages that attempts reached. The Suites column gives the number of suites with the stage type.

Category	Stage type	Suites	`Privileged`	`Standard`	Both
Mechanism	all	18	14 (102/704)	28 (180/638)	21 (282/1,342)
	Press	6	2 (3/144)	4 (3/72)	3 (6/216)
	Close	9	5 (12/242)	8 (21/258)	7 (33/500)
	Switch on/off	4	11 (4/38)	28 (7/25)	17 (11/63)
	Open	12	29 (75/259)	52 (140/271)	41 (215/530)
Pose	all	15	16 (209/1,283)	49 (304/616)	27 (513/1,899)
	Follow path	6	14 (151/1,066)	47 (212/453)	24 (363/1,519)
	Pick up	9	28 (27/95)	45 (42/93)	37 (69/188)
	Adjust pose	9	25 (27/106)	74 (46/62)	43 (73/168)
Dynamic	all	3	18 (106/590)	60 (215/356)	34 (321/946)
	Strike	3	13 (45/354)	54 (107/200)	27 (152/554)
Relation	all	26	26 (523/2,000)	46 (827/1,804)	35 (1,350/3,804)
	Place	19	22 (122/557)	33 (231/697)	28 (353/1,254)
	Thread/hang	6	22 (64/290)	49 (85/172)	32 (149/462)
	Put in	20	28 (228/821)	49 (334/678)	37 (562/1,499)
	Screw/unscrew	3	20 (10/51)	78 (18/23)	38 (28/74)
	Insert	5	25 (14/56)	60 (36/60)	43 (50/116)
	Stack	4	23 (7/31)	67 (22/33)	45 (29/64)
Body	all	7	29 (123/418)	65 (219/339)	45 (342/757)
	Traverse	4	22 (37/167)	53 (48/91)	33 (85/258)
	Navigate	5	30 (6/20)	69 (45/65)	60 (51/85)
Material	all	7	43 (165/386)	84 (353/419)	64 (518/805)
	Clean	3	31 (23/75)	78 (47/60)	52 (70/135)
	Pour	3	22 (2/9)	76 (96/126)	73 (98/135)

Explaining the rates by contact therefore requires a judgment of what each kind of stage asks of the robot. Opening a door or a lid fails at 41% of the stages that attempts reach. Closing one fails at 19% when it does not already hold at the start, and at 7% if the doors and fridges that are closed from the start count too. Relation stages fail at 36% of the stages that attempts reach with one hand or gripper, at 51% with two arms and at 66% with the legs or torso. Rates with tool contact can be lower or higher than with direct contact: stages on moving objects fail less often when the robot acts through a cup or a racket, while pose stages fail more often when it moves an object through a tool. The same mode of contact can be easy or hard: stages done by pressing or wiping fail at 8% of the stages that attempts reach when they press on a mechanism, such as a button, and at 68% when they wipe or press on a material. The pose, relation and dynamic categories lie between these extremes, and their order differs between the two settings, so Figure 13b, which counts both settings together, does not rank these three reliably.

What predicts the failure stage. We predict whether an attempt that reaches a stage fails there from the setting plus one other attribute, such as the stage type or the contact. We use five-fold cross-validation over tasks, so that the tasks of the held-out fold are not seen in training, and report the relative drop in log loss compared with the setting alone. Contact as who × how gives 6.9%, the stage type 6.6%, the stage category 5.1%, how alone 4.4%, the suite 4.3%, the robot's body, grouped into eight kinds, 2.7% and who alone 1.2%. The agent gives 15%. With the setting

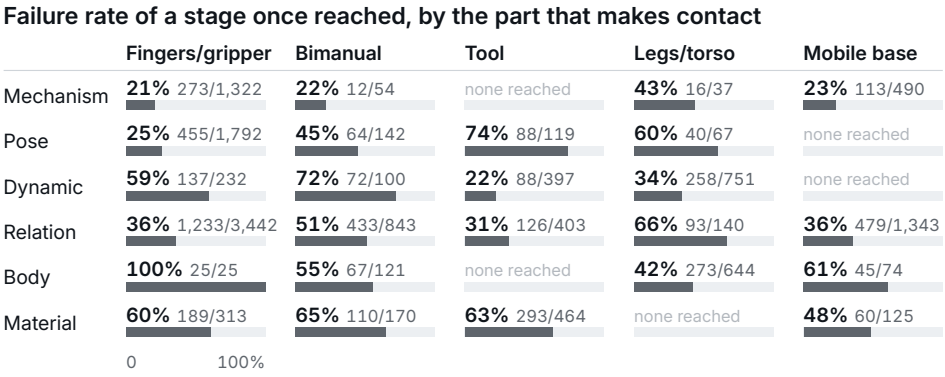


Figure 28 | Failure rate of a stage once reached, by stage category and by the robot part that makes contact, with the `Privileged` and `Standard` settings pooled (Table 18). Each cell gives the rate, the stages failed over the stages reached, and a bar on a scale from 0 to 100%. The rows are those of Figure 13b, at the same heights.

and the suite as the baseline, the stage type adds 8.1%, who × how 7.4%, how 7.1%, the category 6.5% and who 1.4%. To test whether difficulty rankings hold across suites, we hold out one suite at a time, rank each pair of values by their failure rates on the other suites, where each value must be reached at least 20 times, and check the ranking inside the held-out suite, where it must be reached at least 10 times. The rankings agree in 51% of 203 pairs for stage types, 57% of 155 pairs for how, and 61% of 59 pairs for stage categories.

Failure profiles of the agents. Table 19 gives the share of each agent's failed attempts in the `Standard` and `Realtime` settings with each label. A failed attempt can have two phenomena, so the columns do not sum to 100%. The last two rows compare each agent with GPT-6 Astra on the tests with at least three stages that both failed, with ties excluded. Ties are common. Of the 128 tests that GPT-6 Astra and Claude Opus 5.5 both failed with readable progress, GPT-6 Astra reached more stages on 60 tests, Claude Opus 5.5 on 17, and the two tie on 51. Between the other two strongest agents, Claude Opus 5.5 reached more stages than Gemini 4 Argon in 48% of the 52 such tests. In the `Standard` Setting, two of the three strongest agents that failed the same test failed at the same stage in 54% of pairs, and two of the four weaker agents did so in 77%. When they did, that stage came after the first stage in 53% of the 192 such pairs of the strongest agents and in 26% of the 1,017 such pairs of the weaker agents.

The shares of failed attempts depend on how far an agent gets: an agent that reaches more stages that need a hold has more chances to lose one. The two rows "per stage reached" of Table 19 therefore divide by the stages that the attempts reached, using the attempts with readable stage progress. Of the reached stages that need contact, the weaker agents fail without contact at 11 to 18%, and the strongest agents do so at 2 to 4%. The pooled difference, 12 percentage points, has a 95% interval from 10 to 14 points over tasks, and the weaker agents' rate is the higher one in 17 of the 18 suites with at least 20 such stages per group. Over the four pairs of GPT-6 Astra with a weaker agent, on the tests that both failed, 8 to 9% of GPT-6 Astra's failures lack contact, compared with 18 to 22% of the weaker agent's. Lost holds show a different pattern. They make up more of GPT-6 Astra's failed attempts than of any other agent's, but per stage reached that needs a grasp, GPT-6 Astra loses the hold in 5.5% of cases, compared with 8.3% for the other six agents pooled, a difference of −2.8 points with a 95% interval from −4.8 to −0.3 points over tasks. Per stage reached, GPT-6 Astra therefore loses holds less often than the other agents pooled, although the upper end of the interval is close to zero. Among

single agents, only Gemini 4 Argon has a lower rate. Constraint violations, too, make up more of GPT-6 Astra's failures than of any other agent's. Per 100 attempts, GPT-6 Astra has 4.9 and Qwen3.8 Max 4.5.

Table 19 | Failure profile of each agent, given as the share in percent of its failed attempts in the `Standard` and `Realtime` settings with each label. Zero progress, an attempt that met no stage of the task, counts only `Standard` attempts whose progress shows whether they met a stage, on tasks with at least two stages. This rule excludes 8 to 36% of each agent's failures, the most for Qwen3.8 Max. The highest value of each row is in bold. The rows before "Failed while exploring" are phenomena, that row gives the position of the failure stage, and the two rows after it are root causes. The two rows "per stage reached" divide by stages instead of failed attempts: among the reached stages that need contact (or a grasp), the share at which the attempt failed without contact (or with a lost hold). The last two rows compare each agent with GPT-6 Astra on the tests with at least three stages that both failed, excluding ties: the share of these tests on which GPT-6 Astra reached more stages, and their number. Gemini 4 Argon's "not attempted" may still include filter cutoffs whose summaries did not name the interruption.

	Astra	Opus	Argon	G-3.8 Flash	Qwen	DeepSeek	GLM
Failed attempts	279	345	237	429	433	447	442
Zero progress, `Standard`	25	35	28	46	**63**	57	59
Failed to make contact	8.6	9.0	10.5	17.5	20.8	20.1	**22.9**
Lost hold	**17.9**	13.0	10.5	11.9	6.9	11.9	11.5
Inaccurate pose	19.4	20.6	**23.2**	22.6	10.2	13.0	12.7
Obstruction/collision	18.3	15.1	12.2	10.0	11.5	**19.5**	14.3
Constraint violation	**7.5**	4.3	5.1	1.9	4.6	3.8	3.6
Loss of balance	13.6	18.0	8.9	12.8	19.9	**20.4**	19.0
Not attempted	0.4	2.6	**12.2**	3.3	8.1	4.9	3.8
Failed while exploring	3.6	7.8	5.1	3.5	8.1	**14.5**	8.6
False completion belief	0.7	0.3	1.3	**8.2**	1.6	1.3	3.6
Feedback control or trajectory/IK	5.4	**14.2**	5.1	4.7	9.7	12.1	7.5
No contact, per contact stage reached	2.4	3.4	3.8	11.2	17.5	15.2	**17.6**
Lost hold, per grasp stage reached	5.5	6.8	4.8	9.3	8.2	**11.5**	10.3
GPT-6 Astra reached more stages	–	78	79	96	99	100	99
Tests compared	–	77	42	81	71	108	100

Limitations of the labels. The labels used here were written and reviewed by agents and have not been checked manually in a systematic way (Section H.1), and the labeling model is from the same family as GPT-6 Astra. We plan to check about 150 to 200 failed attempts, sampled by agent and setting, against the replays, starting with failure stages and phenomena, and report the agreement. Labels come from the replay video and the logs, without the simulator state, so a wrong estimate and a wrong motion often cannot be distinguished. Perception is named in 5% of `Standard` failures, and 54% of the analyzed failures have a root cause given at the category level, "Control". The root-cause layer therefore supports few conclusions, and the analyses of Sections 3.2 to 3.4 rest mainly on failure stages, stage types and phenomena. The 8 IKEA assembly tests of RoboARC in the `Standard` Setting have no video and were labeled from the logs alone. Each agent makes one attempt per test, so per-agent shares on a few dozen tests have wide uncertainty.

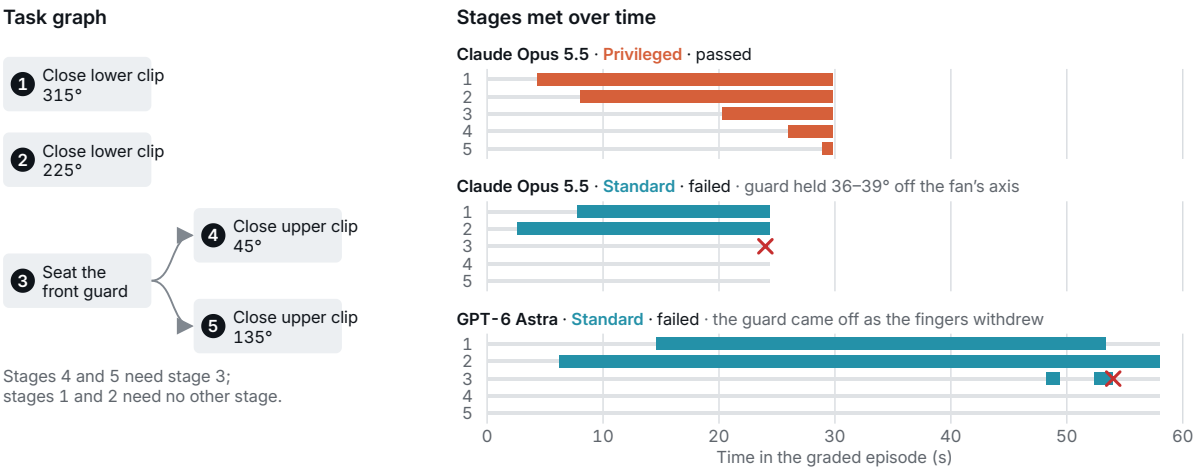


Figure 29 | Task graph of a DexHandExam task, installing the front guard of a fan, and the stages that three attempts met over time. A bar shows when a stage's condition held, and a cross marks the failure stage.

H.6. Case Studies

This section gives in full the attempts behind the examples of Section 3, each with its figure. Times are seconds on the replay video of the graded episode. Three of the attempts described here are reruns in the final results that replaced labeled attempts: those of Claude Opus 5.5 and GLM 5.3 Flash in the ball task, and that of GPT-6 Astra in the mug task. They have no labels yet, so we read them from the replay and the agent's log, and the statistics of Section H.5 do not include them.

Carrying in groceries (Figure 25). In this BEHAVIOR-1K task a mobile two-armed robot must carry a tomato and a carton of milk from a car into the fridge and then close the car and the fridge. The robot must open the garage door and the fridge before either food can go in, and close the fridge only after both are in, while closing the car does not depend on the other stages. Stage 5, the closed fridge, is met from the start and stops holding when the robot opens the fridge. GPT-6 Astra in the `Privileged` Setting reaches all six stages and passes. Claude Opus 5.5 in the `Standard` Setting opens the garage door and the fridge, closes and reopens the fridge twice, puts the tomato in and falls while carrying the milk through the door, so its failure stage is stage 4. DeepSeek V4.1 Flash in the `Standard` Setting lifts the tomato but drops it, so it fails at stage 3, and only the closed fridge is still met.

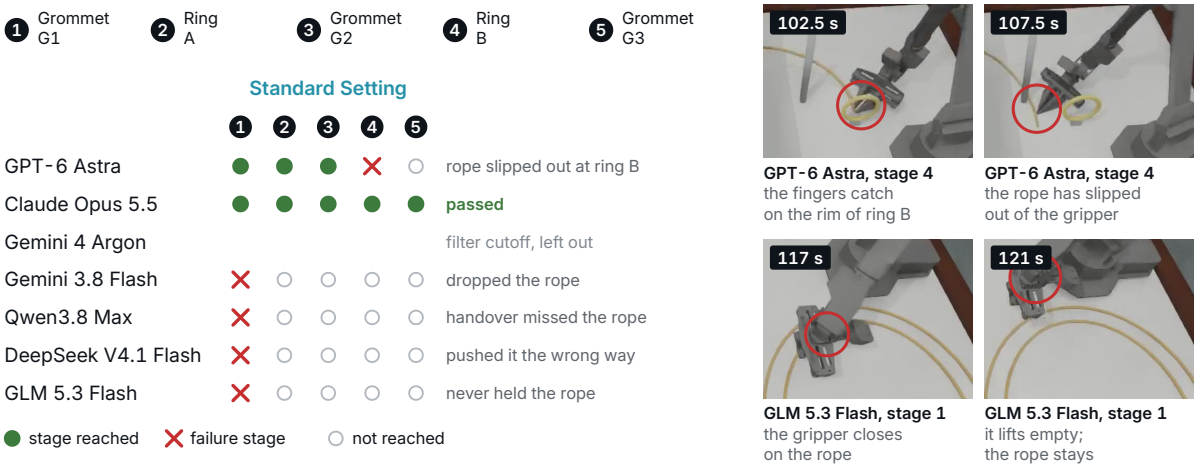
Installing a fan guard (Figure 29). In this DexHandExam task a dexterous hand must seat the front guard of a fan on its rear guard and close four clips. The two upper clips need the seated guard, and the two lower clips do not. In the `Privileged` Setting, Claude Opus 5.5 closed the lower clips, seated the guard at 20 s and closed the upper clips. In the `Standard` Setting it closed the lower clips in the other order, but it held the guard 36 to 39 degrees off the fan's axis, whereas seating allows at most 5 degrees. When it pushed the guard on, the fan moved, and the simulator service ended the episode at 24 s. The labels attribute the failure to a wrong estimate of the guard's pose in the hand. In the same `Standard` test, GPT-6 Astra seated the guard twice, each time for less than 2 s. As the hand withdrew its fingers, the guard came off with it and reopened a lower clip, and the attempt ended when the guard dropped.



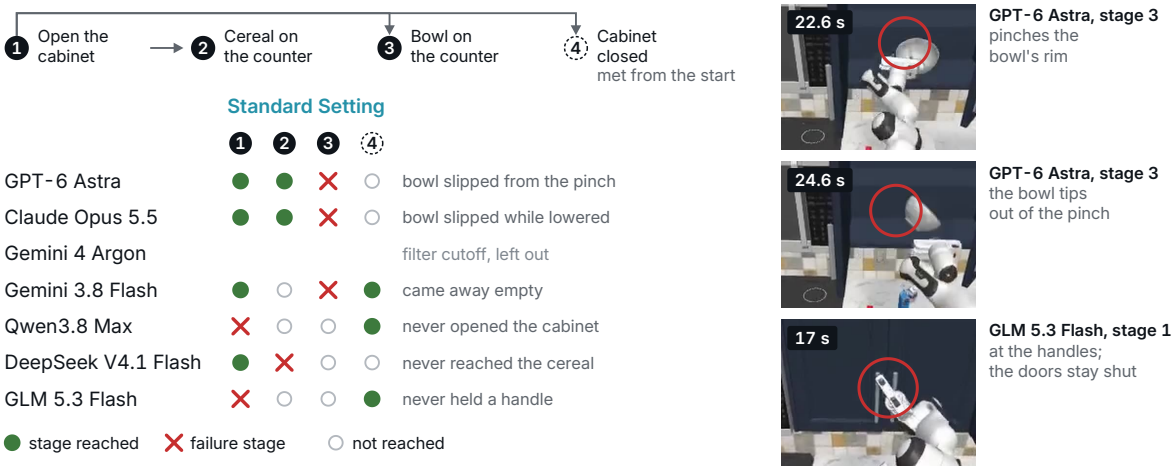
Figure 30 | Progress maps of all seven agents on *preparing a lunch box* (BEHAVIOR-1K, `Standard` Setting). Crosses mark the failure stages.

Catching a ball twice (Figure 14). In this RoboDynamicExam task the robot must catch a thrown ball in a cup, pour it into a slot, catch it again after it rolls down and rebounds off a ramp, then pour it into a moving cart. In the `Standard` Setting, all seven agents caught the thrown ball and failed at the recatch stage. In three attempts, the episode ended because the cup touched the slot or the rebound ramp, which the task forbids. GPT-6 Astra, Gemini 3.8 Flash and Qwen3.8 Max hit them while pouring the ball out or moving the empty cup downstream. The cups of Claude Opus 5.5 and Gemini 4 Argon were still too high when the ball came off the ramp. Claude Opus 5.5 reported that the ball can be caught for only about 1.3 to 1.6 s after it leaves the cup, while straightening the cup after the pour alone takes about 1.4 s. In the runs of DeepSeek V4.1 Flash and GLM 5.3 Flash the ball never left the tilted cup. GLM 5.3 Flash then carried the ball to the cart, where it stayed in the cup again. In the `Privileged` Setting, GPT-6 Astra and Claude Opus 5.5 completed all three stages, and Gemini 4 Argon, Gemini 3.8 Flash and Qwen3.8 Max submitted no answer.

Preparing a lunch box (Figure 30). In this BEHAVIOR-1K task a mobile two-armed robot must put two apple halves, a sandwich and a cookie from a cutting board, and a tea bottle from the closed fridge, into a lunch box, and leave the fridge closed. The closed fridge is met from the start. All seven agents failed the `Standard` test. GPT-6 Astra put both apple halves, the cookie and the sandwich into the box by 250 s. Driving to the fridge with empty hands, the robot tipped over at 284 s, and the episode ended with the fall. Gemini 4 Argon packed the same four foods by 85 s, then pulled open a tall dark cabinet that it had mistaken for the fridge, and it had not opened the fridge when the episode ended. The labels name a perception error. Claude Opus 5.5 packed an apple half and the cookie. At 210 s, the sandwich slipped from the gripper and landed behind the box, and placing the sandwich is its failure stage. After packing the second apple half at 242 s, the robot tipped over on its way to the fridge. Gemini 3.8 Flash opened and closed the fridge but never lifted an apple half off the board, and Qwen3.8 Max never grasped one in its 845 s episode. DeepSeek V4.1 Flash and GLM 5.3 Flash tipped the robot over within 8 s while exploring the kitchen and packed nothing. No agent passed this task in the `Privileged` Setting either, where the three strongest agents reached five or six of the seven stages.



(a)



(b)

Figure 31 | `Standard` Setting. (a) RoboARC, *threading a rope*. (b) RoboCasa365, *fetching cereal and a bowl*; the column of stage 4 shows the end of the episode. Empty rows are attempts left out.

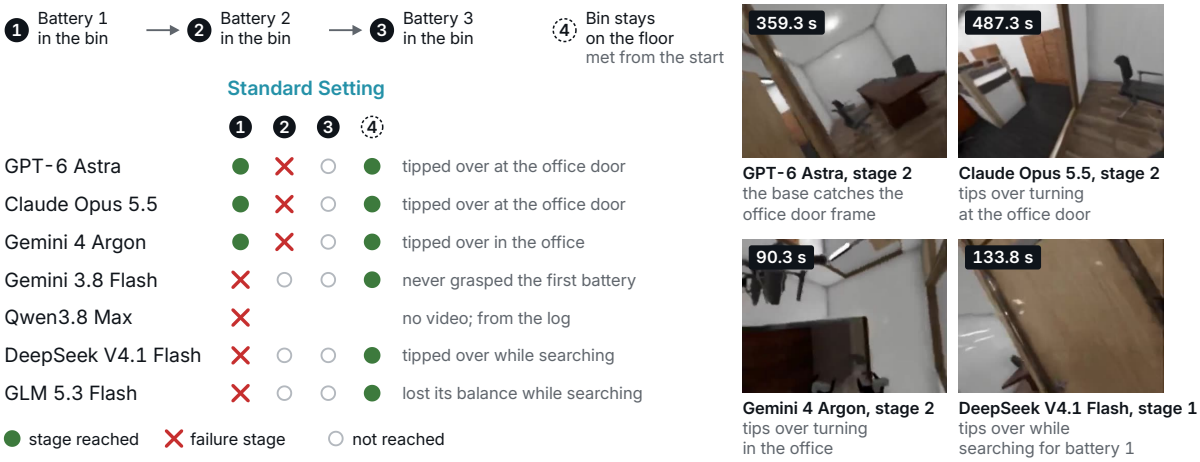


Figure 32 | BEHAVIOR-1K, *disposing of batteries*, `Standard` Setting. Left: the task graph and the stages of each agent; the wastebasket on the floor holds from the start. Right: head-camera frames as the robot tips over.

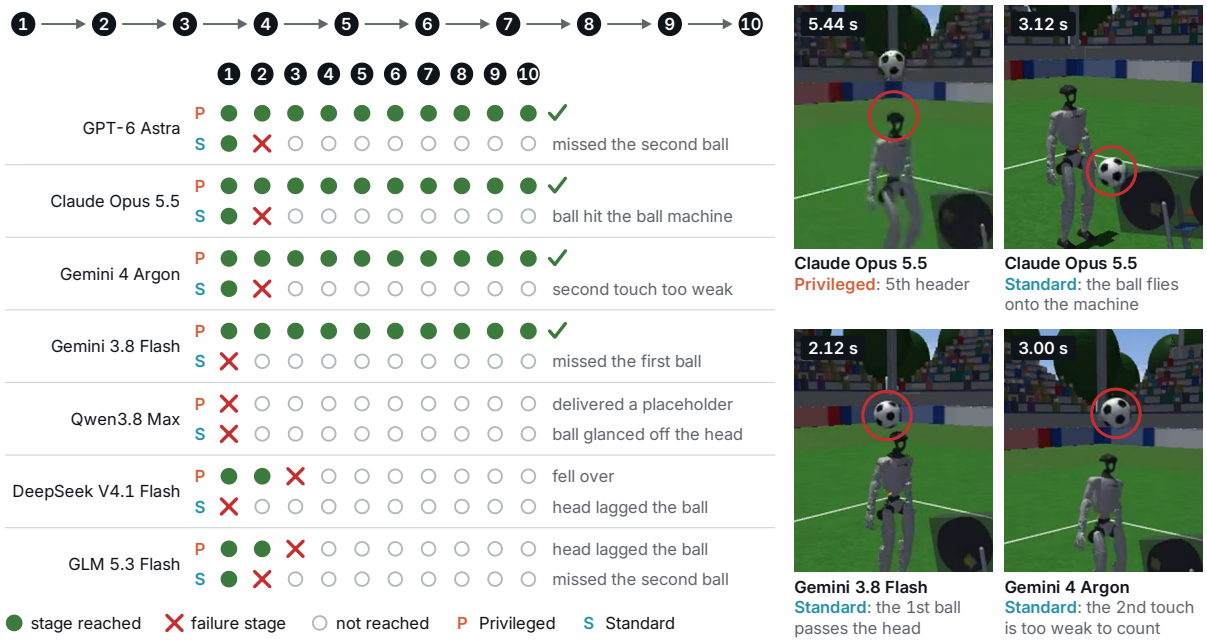
Threading a rope through five passages (Section 3.4, Figure 31a). In this RoboARC task two arms with parallel grippers must thread a rope, in order, through a grommet in a wall, a ring, a second grommet, a second ring and a third grommet. Each passage is a stage. In the `Standard` Setting, Claude Opus 5.5 passed. GPT-6 Astra threaded the first three passages by 100 s. At the second ring, the fingers caught on the rim. At 106 s, the rope slipped out of the gripper, and the time limit ended the episode soon after. None of the four weaker agents got the rope through the first grommet, and two of them never got the hold they needed. GLM 5.3 Flash closed its gripper on the rope at 116 s and again later, and each time lifted the gripper without the rope. Qwen3.8 Max pushed the rope end into the grommet, but at 115 s the gripper on the other side closed without catching it, and the rope slid back. Gemini 3.8 Flash lifted the rope and dropped it at 16 s, and DeepSeek V4.1 Flash tried to thread the first grommet in the wrong direction. Gemini 4 Argon's attempt was a filter cutoff (Section H.3).

Fetching cereal and a bowl (Section 3.4, Figure 31b). In this RoboCasa365 task a one-armed robot on a mobile base must open a cabinet, put a cereal box and a bowl from it on the counter, and close the cabinet. The closed cabinet is met from the start. In the `Standard` Setting, GPT-6 Astra and Claude Opus 5.5 opened the cabinet, put the cereal on the counter and failed at the bowl after holding it. GPT-6 Astra tried to pick up the bowl by its rim from 16 s. At 23 s, the bowl slipped out of the pinch and stayed on the shelf, and later pinches tipped it onto its side. Claude Opus 5.5 lifted the bowl out of the cabinet, and at 30 s, as the arm lowered it, the bowl slipped from the fingers and fell out of reach. Three of the four weaker agents failed without the contact that their failure stage needed. Gemini 3.8 Flash closed its gripper at the bowl at 2 s and came away empty. GLM 5.3 Flash never got hold of a door handle, and DeepSeek V4.1 Flash opened a door but never reached the cereal box. Qwen3.8 Max never opened the cabinet. Gemini 4 Argon's attempt was a filter cutoff.

Disposing of batteries (Section 3.3, Figure 32). In this BEHAVIOR-1K task a mobile two-armed robot must put three batteries into the wastebasket of a copy room, one from a cabinet in the copy room and two from a desk in an office along the corridor, and leave the wastebasket on the floor. The task graph has four stages, one for each battery and one for the wastebasket on the



(a)



(b)

Figure 33 | `Privileged` and `Standard` settings. (a) RoboTwin 2.0, *hanging a mug*: progress maps of GPT-6 Astra. (b) Embodied Olympics - Fun, *heading a ball ten times*: the headers of each agent in the `Privileged` (P) and `Standard` (S) settings.

floor, which holds from the start. In the `Standard` Setting, the three strongest agents each put the cabinet battery into the wastebasket and then failed at the second battery, when the robot tipped over. GPT-6 Astra dropped the first battery at 90 s. At 359 s, as it drove into the office, the base caught the door jamb and the robot tipped onto its side. It righted the robot by 542 s and picked up a second battery at 592 s, and it still held the battery when the episode ended. Claude Opus 5.5 dropped the first battery at 102 s, and the robot tipped over at 486 s as it turned at the office door. Gemini 4 Argon dropped the first battery at 23 s and tipped the robot over at 90 s while turning in the office. In both of these runs the simulator service ended the episode because the robot had fallen. The four weaker agents failed at the first battery. Gemini 3.8 Flash never lifted it from the cabinet, DeepSeek V4.1 Flash tipped the robot over while searching for it, and GLM 5.3 Flash lost the robot's balance while searching. Qwen3.8 Max has no video, and its log reports that the robot fell.

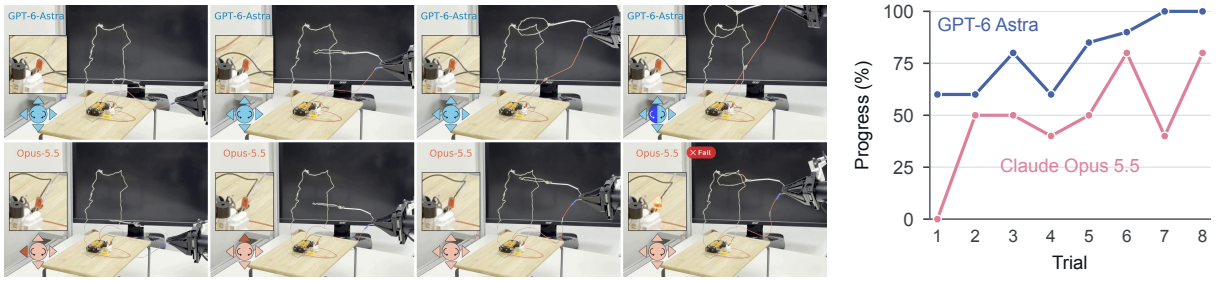
Hanging a mug (Section 3.2, Figure 33a). In this RoboTwin 2.0 task a two-armed robot must pick up a mug with the left arm, turn it and put it down in the middle of the table, and then hang it on a rack with the right arm. In the `Privileged` Setting, GPT-6 Astra put the mug down at 9 s and hung it at 23 s. In the `Standard` Setting it put the mug down at 4 s and carried it to the rack with the right arm. At 9 s, having reported that the peg was through the handle, it opened the gripper, and the mug slipped off as the gripper withdrew and fell to the table. Two more tries also failed, and the episode ended with the mug not hung. Claude Opus 5.5 passed both the `Privileged` and the `Standard` test.

Heading a ball ten times (Section 3.2, Figure 33b). In this Embodied Olympics - Fun task a humanoid robot must head a ball from a ball machine ten times in a row, and each header is a stage. In the `Privileged` Setting, GPT-6 Astra, Claude Opus 5.5, Gemini 4 Argon and Gemini 3.8 Flash all made the ten headers. In the `Standard` Setting the agent sees the ball only in camera images, the task grants 5 resets, and only the last episode counts. There, the graded episodes of the same four agents reached one, one, one and zero headers. Claude Opus 5.5 first watched the lob twice while standing still. It then tracked the ball in the venue camera, fitted its flight, and timed a crouch and a push up of the legs to the ball's arrival. From its third episode on, this approach made the first header every time, and it spent the remaining resets aiming that header with the waist and ankles. In its graded episode the header sent the ball onto the ball machine at 3.1 s. GPT-6 Astra made its first header in its fourth episode; in its graded episode the second ball missed the head and hit the ground at 3.1 s. Gemini 4 Argon's second touch, at 3.0 s, was too weak to count as a header. Gemini 3.8 Flash made no header in its first six episodes, and in its graded episode the first ball passed the head and hit the ground at 2.8 s.

I. Real-Robot Demonstration

Because the `Standard` Setting gives the agent only the robot's sensor readings and its control interface (Section 2.1), an interface of the same kind can connect a coding agent to a physical robot. As a first test, we connected two coding agents to a physical robot arm, and each agent played a buzz wire game for eight trials in a row (Figure 34). This demonstration is not part of the exam, and its results are not included in any success rate in this paper.

Task. In a buzz wire game, a player guides a metal loop along a bent wire from one end to the other without touching the wire. The loop and the wire belong to one electric circuit, so



(a) (b)

Figure 34 | Buzz wire game on a physical robot arm. (a) Each row shows four frames of one trial in time order, with GPT-6 Astra in the top row and Claude Opus 5.5 in the bottom row. The inset shows a close-up of the light and the buzzer, and the pad at the lower left shows the command being executed. In the last frame of the bottom row, the loop has touched the wire and the light is on. (b) The plot gives the progress score of each agent in each of its eight trials. The frames and the scores are taken from the project video.

a contact closes the circuit and turns on a light and a buzzer. In our setup, both ends of the bent wire are fixed to a wooden board, and the robot holds the handle of the loop in its gripper (Figure 34a). A contact counts as a failure, as in the buzz wire tasks of RoboARC (Section A.1), in which a simulated Franka Panda arm guides a metal loop along a wire.

Commands and video. The agent controls the robot with discrete commands. Each command moves the gripper up, down, left or right or rotates it in either direction. In each panel of Figure 34a, the pad at the lower left highlights the command being executed, and the inset shows a close-up of the light and the buzzer, so a contact is visible in the video.

Agents and trials. The two agents use GPT-6 Astra and Claude Opus 5.5, the two models with the highest `Standard` success rates in Figure 7. Each agent ran one sequence of eight trials on the same wire.

Progress score. Each trial receives a progress score, the percentage of the wire that the loop covers before the trial ends. A score of 100% means that the loop passed along the whole wire without a contact.

Results. Both agents score higher in the last four trials than in the first four, and GPT-6 Astra scores higher than Claude Opus 5.5 in every trial (Figure 34b). Averaged over the first four trials, the progress is 65.0% for GPT-6 Astra and 35.0% for Claude Opus 5.5. Averaged over the last four trials, the progress is 93.8% and 62.5%. GPT-6 Astra scores at least 80% in five of its eight trials, and Claude Opus 5.5 does so in two. GPT-6 Astra starts at 60% and passes along the whole wire in trials 7 and 8. Its progress falls once, from 80% in trial 3 to 60% in trial 4. Claude Opus 5.5 scores 0% in trial 1 and between 40% and 80% in trials 2 to 8, and it does not pass along the whole wire in any trial. Its progress varies more from trial to trial. It rises to 80% in trial 6, falls to 40% in trial 7 and returns to 80% in trial 8. In the trial shown in Figure 34a, Claude Opus 5.5 touches the wire, and the light in its close-up turns on. The two agents rank in the same order as in the `Standard` Setting in simulation, where GPT-6 Astra solves 34.2% of the `Standard` tests and Claude Opus 5.5 solves 25.0% (Figure 7).

Scope. The demonstration covers one task, one robot and two agents, with one sequence of eight trials per agent. It shows that a coding agent can control a physical robot through an interface of this kind, and that the progress of both agents rose over the eight trials. With one sequence per agent, the demonstration cannot separate this rise from variation between runs. It also does not measure how well success rates in simulation predict results on hardware (Section J). A systematic evaluation on physical robots, with a grading program and several runs per agent, remains future work.

J. Limitations

Each result holds for one model in one harness. Each agent also makes a single attempt per test, so the results cannot separate an agent's ability from the variation between runs. The core set excludes tasks that we judged easy and tasks that GPT-6 Luna solved in a screening run, so its success rates are lower than those of the full exam by an amount that differs between suites. Because GPT-6 Luna belongs to the same family as GPT-6 Astra, the screening may also bias the core set against tasks that this family solves. Many per-suite rates are based on few tests. In a suite with five or six tests, a single test changes the rate by 17 to 20 percentage points, and a single `Realtime` test changes an agent's success rate over all `Realtime` tests by about two points. The stage-level and control-approach labels come from agents and have not yet been checked manually in a systematic way. The model that wrote the stage-level labels, GPT-6.1 Sol, belongs to the same family as GPT-6 Astra. This shared family could bias the label-based comparisons between agents. A test without a verdict counts as failed. Of the 1,156 tasks, 392 have no reference solution, so we have not shown that each of them can be solved in every setting. Time limits differ between suites, the tasks of Embodied Olympics - Fun run with open network access, and results on tasks adapted from public benchmarks may partly reflect memorization of published solutions (Section B). Finally, all results outside Section I are in simulation. The demonstration in Section I covers one task on a physical robot, so we have not yet measured how well the simulation results predict results on hardware.

K. Extended Related Work

This appendix extends the review in Section 4. We first review benchmarks for coding agents and benchmarks for robot learning, and then work that uses language models for robots. We then review recent work that runs general-purpose agents on robots, together with the harnesses and benchmarks built for these agents.

Benchmarks for coding agents. Coding agents are evaluated mostly on digital tasks. These tasks include writing programs from a problem description [12, 52], resolving issues in real repositories [58, 99, 169], completing long software tasks and tasks in a terminal [67, 97], engineering algorithms for hard optimization problems [51], machine learning engineering and research [11, 137, 158], writing efficient GPU kernels [111], cybersecurity [183], and operating websites, desktop applications and other tools [85, 98, 163, 174, 192]. Most of these agents interleave reasoning with tool calls [173] and revise their work from feedback [131]. SWE-agent [168] shows that the interface between the model and the computer affects how often an agent succeeds, and OpenHands [154] provides an open platform for building and evaluating such agents. Other work measures how often agents exploit the tests that grade them [190] and proposes practices for building rigorous agentic benchmarks [197]. The Embodied First Exam keeps each coding agent's harness unchanged and extends these evaluations from digital tasks to robotics

tasks. In its `Standard` and `Realtime` settings, the agent must interpret sensor observations and often cannot undo an action, and in the `Realtime` Setting the environment keeps running while the agent reasons. In every setting, a program that the agent cannot change grades each attempt.

Benchmarks for robot learning. Most robotics benchmarks evaluate learned policies, from imitation-learning policies [19, 189] to generalist policies and vision-language-action models [7–9, 31, 61, 62, 105, 108, 114–116] trained on large robot datasets [60, 109]. World action models build policies on pretrained video generation models such as Cosmos [104] and predict future observations together with actions [63, 89, 176], extending earlier work that extracts actions from generated videos [23] or learns a joint representation of video and actions [75]. They are built on physics simulators and robot learning frameworks [34, 36, 93, 100, 106, 117, 138, 140, 141, 143, 196]. They cover tabletop and bimanual manipulation [5, 13, 14, 53, 57, 81, 94, 95, 105, 118, 166, 178, 184, 187], household activities [65, 70, 101, 102, 132, 136], humanoid and legged whole-body control [3, 126, 181, 188], dexterous and tool manipulation [17, 59, 149], musculoskeletal control [10], physical reasoning and creative problem solving [49, 79], aerial manipulation [156], laboratory automation [68, 73] and safety [113]. Policies trained on real data are also evaluated in simulated replicas of real setups [76], on real robots at scale [6, 195] and in learned world models [33, 37, 120], which can also generate training data for policies [54, 171]. These benchmarks do not need to restrict what the evaluated system can access, because a learned policy receives only a fixed set of observations. A coding agent can run any program on its computer and could read the simulator's files, so the exam adds rules about what the agent may access (Section 2.1). The exam adapts 24 of these benchmarks and two others, EmbodiedSWE [128] and ABC Sim, and keeps their scenes and success rules, so that coding agents can attempt their tasks (Section A.2). Its `Standard` Setting gives the agent observations of the kind that a learned policy receives, so coding agents and learned policies can be compared with the same observations in future work.

Language models for robot planning and control. Language models have been used to plan tasks for robots, write robot programs and design training signals. SayCan [1], Inner Monologue [46], ProgPrompt [135], Text2Motion [80] and SayPlan [121] sequence learned or scripted skills into plans, PaLM-E [22] and Gemini Robotics-ER [31, 32] train multimodal models for embodied reasoning, and embodied chain-of-thought reasoning [182] trains a vision-language-action model to reason about the task before it acts. AutoRT [2] uses foundation models to propose and assign tasks to a fleet of robots, and SOAR [194] uses them to propose tasks and practice them autonomously. Code as Policies [78], ChatGPT for Robotics [145], Instruct2Act [45] and RoboTool [165] turn instructions into programs that call perception and control primitives, and Code-as-Monitor [191] applies the same idea to failure detection, with a model that writes code to check spatio-temporal constraints during execution. VoxPoser [47], ReKep [48] and MOKA [82] have a model specify value maps, keypoint constraints or visual marks, which a planner or a set of motion primitives then turns into motions. Language to Rewards [179], Eureka [90], Text2Reward [162], DrEureka [91] and RDA [69] have a model write and refine reward functions, which a controller or a reinforcement learning algorithm then optimizes. RoboGen [155] and GenSim [150] generate simulated tasks and training data, and Voyager [148] builds a library of code skills in an open-ended game. Most of these systems pair a model with an interface designed for it, such as a set of primitives, a planner or a reward template. Such interfaces contribute the kind of classical engineering that Goldberg [35] argues robotics still needs, because robot data are far scarcer than the text on which language models are trained. The Embodied First Exam builds on this line of work and evaluates unchanged

general-purpose coding agents on the whole engineering loop of reading an interface, writing a controller, running the controller and revising it.

Embodied benchmarks for language models. ALFWorld [133] pairs text-based household tasks with their embodied versions and has become a common testbed for language-model agents [131, 173]. Embodied Agent Interface [72], EmbodiedBench [170] and VisualAgentBench [86] evaluate language models as decision makers in simulated households and other embodied environments, and MineDojo [25] provides open-ended tasks in Minecraft. Embodied Agent Interface divides embodied decision making into goal interpretation, subgoal decomposition, action sequencing and transition modeling, and EmbodiedBench evaluates both high-level tasks and low-level navigation and manipulation. In these benchmarks the model chooses from an action space that the benchmark defines, which ranges from high-level skills such as picking up an object to discretized end-effector poses. The Embodied First Exam gives the agent the robot's own programming interface, so the agent also decides how its actions are produced.

Agentic robots. A growing body of recent work runs general-purpose agents on robots, many with GPT-6 Astra [110] or Claude Fable as the model. Recent surveys organize this work by model, data, environment and harness [119] and compare agents that use robots with agents that use computers [16]. This work differs in two respects: the control approach that the agent uses, and the harness that connects the model to the robot. This paragraph groups the work by the three control approaches of Figure 2, and the next paragraph reviews harnesses.

Direct control. In this approach, the agent observes the scene and outputs each action itself. Several studies use GPT-6 Astra as the policy in this way. They evaluate it on RoboDojo, RoboTwin 2.0 and other benchmarks [124, 185, 193], on memory-dependent manipulation tasks [55], on contact-rich tasks with tactile feedback [157], and with demonstrations and its own interaction history in context [18, 83].

Code to control. In this approach, the agent writes programs that produce the robot's actions. Several studies give a general-purpose coding agent the robot's programming interface and add few tools for the task. FAEA [144] runs the unmodified Claude Agent SDK on simulated manipulation benchmarks with access to the simulator state, and Agent as Policy [56] has a general-purpose agent interpret camera images, write programs and send motion commands on real robots. Claude Code (Opus 5) and Codex (GPT-5.6 Sol and GPT-6 Astra) write task and motion planners that generalize to unseen problem instances [96]. With Claude Fable, a coding agent writes a controller that solves Push-T without demonstrations [161], writes force-feedback programs for contact-rich tasks [139] and develops its own tools on a real robot [30]. In Anthropic's Project Fetch, human teams programmed a quadruped robot with and without Claude [4], and Claude Opus 4.7 later did the same tasks on its own, about 20 times faster than the fastest human team [50]. Other systems add procedures that help the agent write better programs. They search for repositories of perception, planning and control code [24], build libraries of skills or programs from execution feedback [66, 88, 103], reuse body descriptions, past experience and skills derived from the agent's own routines [39], write a shared code library that makes every decision without a model at test time [43], infer a program from a single human demonstration [87], let the agent check and retry primitives within Python cells [134], or let the agent develop its tools in simulation before it controls a real robot [42]. RAX [44] has a coding agent explore online without demonstrations or hand-designed primitives and turn the successful action sequences into programs for single-arm, bimanual, dexterous, mobile and humanoid systems, both in simulation and on real robots.

Train a policy. In this approach, the agent trains a policy and then uses it to control the robot. ENPIRE [160] gives coding agents a real-world loop of resetting the scene, running a policy, verifying the outcome and refining the policy. With this loop, frontier coding agents train policies that reach a 99% success rate on dexterous tasks such as organizing a pin box. HARBOR [77] has a team of agents set up simulated environments, design rewards and tune reinforcement learning. The reward design methods above automate one step of this approach.

Each of these studies examines one control approach or a fixed combination of approaches. The Embodied First Exam supports all three in one framework and leaves the choice to the agent, so it also measures which approach agents choose (Section 3.5).

Harnesses for agentic robots. The harness is the program between the model and the robot. Harnesses for robots differ in the action interface they provide, in whether the agent can call learned policies, and in the support they add for tracking state and checking outcomes.

Action interfaces. The success rates of coding agents depend on the level of abstraction of the interface [27]. Show-Harness [15] gives vision-language models a small set of semantic action units, which robot-specific interpreters convert into robot actions, and VIA [41] gives off-the-shelf agents a visual interface with tools for probing pixels and setting end-effector targets. HexaShell [29] shares 14 general tools for motion, perception and state across all tasks and robots, because code refined for one benchmark tends to overfit it. OpenRUA [20] gives an unmodified coding agent only terminal access to the robot's ROS 2 interface and its documentation. The uni-robot-agent harness [180] supports open-weight models that can be trained further.

Learned policies as tools. Hi Robot [130] trains a high-level vision-language model that sends language commands to a vision-language-action policy. RoboClaw [74] and Harness VLA [186] let the agent call vision-language-action policies as tools, Recursive Harness Distillation [64] distills a strong agent's interventions in a vision-language-action policy into a playbook for a lighter agent, Spotter [71] has a vision-language model correct a policy after a confirmed failure, and Gemini Robotics 1.5 [32] pairs an orchestrating reasoning model with a vision-language-action model. The Physical Agency orchestrator [28] plans subgoals, calls vision-language-action policies and parametrized skills, checks the outcomes and recovers from failures. Yao et al. [175] evaluate GPT-6 Astra on EBench both as the policy and as an agent that can call vision-language-action and world action model policies, and compares both with the policies alone.

State tracking and verification. Other harnesses add a persistent representation of the scene and a check of each action's outcome [151], calibrated 3D measurements and a preview of each motion [147], or sub-agents for perception, verification and memory [26].

The Embodied First Exam runs any harness and any model without modification, so different harnesses can be compared on the same tasks.

Benchmarks for agentic robots. Several benchmarks evaluate agents that control robots. CaP-X [27] evaluates coding agents that compose perception and control primitives at several levels of abstraction and finds that their success rates depend on the abstractions that designers provide. RobotWorld [172] evaluates agents on 84 tasks in manipulation, mobile manipulation, locomotion, driving and aerial control with executable success checks. EmbodiedSWE [128] evaluates coding agents on long-horizon dexterous tasks and turns successful agent solutions into demonstrations for training vision-language-action policies. RoboQuest [84] evaluates whether agents explore to find the information a task requires, BiGym 2.0 [188] compares learned policies with programs that coding agents develop on humanoid household tasks, and

Xu et al. [164] evaluate whether coding agents avoid an obstacle they are told not to touch and propose obstacle-aware harnesses for route planning and contact. Earlier safety studies jailbreak robots that language models control into harmful actions [122] and test whether models recognize undesirable situations in images and text [125] and whether agents that plan with high-level actions reject hazardous household tasks [177], while the exam's Embodied Safety Exam decides safety from what physically happens when the agent controls the robot in a physics simulation. Several of the agentic-robot studies above report results on RoboDojo, RoboTwin 2.0 and KinDER, and the exam adapts all three suites. The Embodied First Exam covers 47 robots of eight types, which include humanoids, quadrupeds, dexterous hands and aerial robots. It also separates what an agent can do with full access to the simulator from what it can do through the robot's sensors alone.