SPOTTER: LET THE EMBODIED MODEL LEAD, AND THE VLM REFLECT FOR IT

Anonymous authors
Paper under double-blind review

ABSTRACT

Current embodied models do not respond to their own failures, although what just went wrong could inform a small adjustment on the next attempt; this is the mechanism behind the gains language models obtain from thinking, where a model reflects on an error and revises. We test whether they can repair a known error, which requires producing a correction and judging whether it is right. When an external interrupt stops them at a failure and lets them retry, their own randomness seldom repairs the error, a language description of the error changes little, and best-of-N selection cannot pick the successful candidate in post-failure states. We attribute this to training on successful demonstrations, which contain no failure or recovery, and to inputs too narrow to show what went wrong, and conclude that reflection must be supplied by a vision-language model (VLM), which can take in far more information, such as the history of the episode and text, and is more general. Prior VLM-led work has the VLM plan every step and invoke the embodied model as a tool, placing the VLM on the critical path. We propose Spotter, which reverses the roles: the embodied model leads and executes continuously, while the VLM runs in parallel, monitors through a lightweight local screener, intervenes only when an error is detected, reflects on and corrects it, and returns control. We run Spotter with Qwen and with GPT as the VLM, and both improve the embodied models; with GPT, Spotter improves Cosmos Policy and $\pi_{0.5}$ by 5.6 and 7.5 percentage points on RoboCasa, and raises $\pi_{0.5}$ from 47.2% to 57.0% on the Hard setting of RoboTwin 2.0 and from 53% to 83% on a real robot. Because the VLM steps in only when an error is confirmed, a successful episode with Qwen takes only 13 to 16 s longer than with the embodied model alone and about 70% less time than with a VLM-led baseline using the same model.

1 INTRODUCTION

Embodied models have advanced rapidly, from generalist policies that map images and language to actions (Zitkovich et al., 2023; Kim et al., 2025) to models built on vision-language backbones or pretrained video models that succeed on standard manipulation benchmarks (Black et al., 2025b;a; Bjorck et al., 2025; Gemini Robotics Team et al., 2025; Kim et al., 2026b). Yet they cannot correct their own errors. In Figure 1 the grasp misses the lime, and the policy carries nothing toward the plate as if the object were in hand. Independent evaluations agree: failures are dominated by missed grasps and premature releases (Zhang et al., 2025), policies rarely recover from them (Yu & Qiu, 2026), and they keep executing actions that no longer solve the task (Zhao et al., 2026a; Agia et al., 2025). In manipulation this is costly, because whether a grasp holds or a placement is stable is known only after the fact (He et al., 2026), so errors must be caught and corrected during execution.

Correcting an error takes two abilities, producing a correction and judging whether it is right; current embodied models have neither (Section 3.1). Stopped at a failure and given fresh attempts, they repair fewer than one in ten, whether they resample, are told what went wrong, or are fine-tuned on recovery data; and test-time selection, which samples several candidates and lets a scorer keep one, fails because after a failure the scorer cannot tell which will succeed. Two things are missing. The policy and its scorers learn from successful demonstrations, in which failures never appear (Lin et al., 2025; Hu et al., 2026), so naive retries repeat the original error (Hao et al., 2026), and fine-tuning does not help because recovery data are scarce in the policy's own rollouts and cover few kinds of failure. And they see only the current moment, whereas telling a recovery from a

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

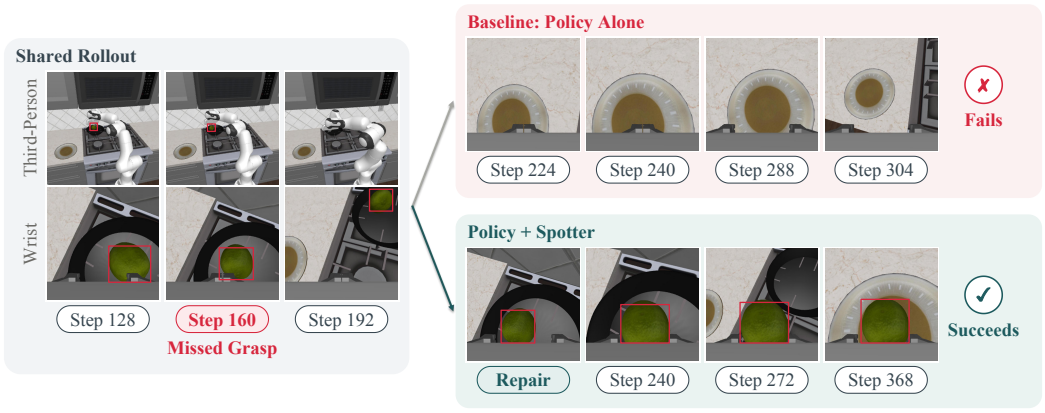


Figure 1: The embodied model alone and with Spotter after the same missed grasp (task: move the lime from the pan to the plate). The grasp misses at step 160; by step 192 the arm heads for the plate. Alone (top), the model never notices, replaying the shape of a successful trajectory. With Spotter (bottom), the VLM notices the miss only after the arm has moved on: by step 192 it has already left the pan. This delay can be undone: the repair returns the arm to its pose before the miss and, since the failed attempt was off to the left, shifts the gripper right to face the lime.

repeated failure needs the history of the episode: even a vision-language model (VLM) shown only the current images judges no better than random. Given that history, a VLM can supply both: it reads earlier attempts and text, and generalizes beyond the demonstrations, routinely correcting its own errors (Shinn et al., 2023; Madaan et al., 2023); Section 5 shows that it intervenes where it should and repairs errors the embodied model does not.

Prior work that pairs the two is VLM-led (Liang et al., 2023; Zhang et al., 2026b; Wang et al., 2026): the VLM plans every step and calls the embodied model as a tool. With the embodied model as a fast System 1 and the VLM as a slow System 2 (Bjorck et al., 2025), control switches wholesale: while the VLM deliberates, the embodied model waits, and while it acts, nothing watches. The VLM must also fix in advance how long each stretch runs: short stretches waste time on checks the policy does not need, and long ones let it go far down a failure.

We propose Spotter, which reverses the roles, much as a person lets a skilled action run and calls in deliberate thought only when something seems off. The embodied model leads and executes continuously. A lightweight local screener screens every chunk and forwards suspicious windows to the VLM, which runs in parallel; how often the VLM looks thus depends on events rather than a fixed schedule, and execution pauses when an error is confirmed or the embodied model gets K chunks ahead of the supervisor. The VLM then repairs the error with a short plan over motion primitives and reflects on it: it states what it expects to see and returns control, then checks the next supervised window and, if the expectation failed, tries a different kind of repair. Our contributions are threefold:
(1) an evaluation of the error-correction ability of current embodied models, testing whether they can produce a correction and judge whether it is right;
(2) Spotter, an embodied-led framework in which a VLM supervises in parallel and intervenes only on failure, without training the policy;
(3) experiments with Cosmos Policy and $\pi_{0.5}$ on RoboCasa and $\pi_{0.5}$ on the Hard setting of RoboTwin 2.0 and a real robot, with Qwen or GPT as the VLM, showing that Spotter raises success rates in far less time than a VLM-led design, with successful episodes taking time of the same order as with the embodied model alone.

2 Related Work

Robot Agents. Pairing a language model with an embodied model is an emerging paradigm for open-vocabulary manipulation, and existing systems are VLM-led, with the language model decid-

ing. It may write programs over perception and motion APIs (Liang et al., 2023), as benchmarked by CaP-X (Fu et al., 2026); issue a language command every second, or whenever the user speaks, to an embodied model fine-tuned to follow such commands, as in Hi Robot (Shi et al., 2025), which targets open-ended instructions rather than failures; invoke a frozen embodied model as one primitive among several, as in Harness VLA (Zhang et al., 2026b); or run an evaluator after every action to diagnose failures, as in Towards the Harness of Embodied Agents (Wang et al., 2026). RACER (Dai et al., 2025) queries a VLM supervisor at every step for a language hint, training both supervisor and actor on failure-recovery data. In all of these, execution waits on the language model at every decision, so latency grows with task length rather than with the number of failures. Spotter runs the VLM in parallel with a leading embodied model, pauses execution on a confirmed error or at the bounded lead limit, and lets the VLM repair the failure itself with a short primitive plan (Table 6, Appendix D).

Failure Explanation and Recovery. Language models have long been used to explain and recover from robot failures. Inner Monologue (Huang et al., 2023) feeds success detection and scene descriptions back to a language-model planner, which replans in language, and DoReMi (Guo et al., 2024) has the planner also write constraints that a VLM checks continuously during execution, triggering recovery when one is violated. Both are VLM-led: the language model plans over a fixed set of skills, and recovery means replanning at the skill level. Another line learns from failures: RoboFAC (Ye et al., 2025) builds a failure-centric dataset of erroneous trajectories and question-answer pairs and fine-tunes a multimodal model that acts as an external supervisor of a vision-language-action model (VLA), and AIC-MLLM (Xiong et al., 2025) fine-tunes a multimodal model to correct its own SE(3) contact poses on articulated objects from interaction feedback. Spotter needs neither failure data nor training: the embodied model leads, a VLM checks it in parallel, and the VLM repairs an error itself with motion primitives rather than handing a plan or instruction back to it.

Test-Time Scaling. Spending extra inference compute is standard in language modeling, through beam search (Xie et al., 2023), majority voting (Wang et al., 2023), and verifier-based best-of-N selection (Cobbe et al., 2021). The recipe transfers to embodied models: a stochastic policy can be sampled repeatedly at one state, and a model that also predicts future observations attaches an imagined outcome to each candidate. Candidates are ranked by a learned value, as in Cosmos Policy (Kim et al., 2026b); by a separately trained vision-language progress critic, VLAC (Zhang et al., 2026a); or without training, by Consistency-Consensus (Ruan et al., 2026) (with its non-deployable upper bound Consistency-Exploring), GeoBoN (Zhao et al., 2026b), and RCS from τ_0-WM (Zhou et al., 2026) (definitions in Appendix A). All assume that the scorer can tell good candidates from bad ones; since policy and scorers alike learn from successful demonstrations, we test this right after a failure, asking which candidate repairs it (Section 3.1), and find that the assumption does not hold.

3 Method

3.1 Error-Correction Ability of Current Embodied Models

We first evaluate the error-correction ability of current embodied models: an oracle marks when an error has occurred and interrupts execution there, and we measure how well the model then corrects it. Correction involves two abilities, producing a corrected action and judging whether a correction is right. We pose each as a question and answer it with a dedicated experiment using the world action model Cosmos Policy and the VLA $\pi_{0.5}$ on the pick-and-place tasks of RoboCasa, where an oracle tells unambiguously whether the object has been grasped.

Question

Q1: Once an error is caught by the oracle, can the embodied model produce a correction?

We interrupt execution at each failure and give the model three attempts to correct it under each of three protocols, counting the failure as corrected if any attempt lifts the object (Appendix A).

Resampling. We restore the simulator to one chunk before the failed grasp and let the model act again under three random seeds, executing all three continuations; with no signal that anything went wrong, this tests whether its own randomness alone can repair the error.

Language feedback. Execution stops at the failed grasp without rewinding; a VLM shown the current camera views is told that the grasp has failed and asked what went wrong and what to do differently, and its answer becomes the model's instruction for another attempt, up to three rounds.

Recovery fine-tuning. Following Hu et al. (2026); Hao et al. (2026); Zhao et al. (2026a), we fine-tune the policy on recovery data collected from its own rollouts and repeat the resampling protocol; in our runs only roughly 3% of ordinary rollouts yield recovery data meeting our requirements, so the data are scarce and cover few of the many kinds of failure.

Figure 2(a) reports the outcome. None of the three works: fewer than one failure in ten is corrected, even though execution is interrupted right at the failure and three attempts are allowed. Language feedback is only slightly better than resampling: being told what went wrong leaves the model about as likely to succeed as drawing a new seed, because current embodied models do not take in an instruction and revise their behavior the way a language model does, and Cosmos Policy does not even support instructions other than the official ones it was trained with. Fine-tuning on recovery data does not help either (8.13% for Cosmos Policy, 8.44% for $\pi_{0.5}$), and in our runs the overall success rate stays close to the baseline.

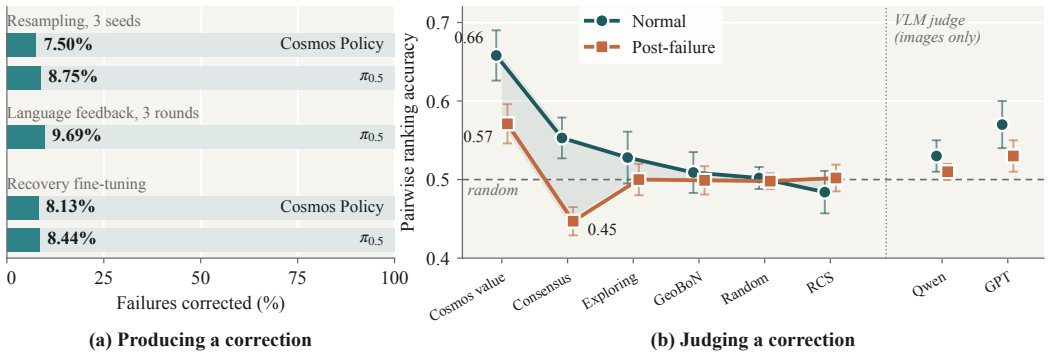


Figure 2: Error-correction ability of current embodied models. (a) Fraction of failed grasps that the model corrects when given three more attempts, before and after fine-tuning on recovery data, over 320 failed grasps per setting. (b) Pairwise ranking accuracy of test-time scorers and of the VLM judge given only images around the saved state, at normal states and after a failed grasp; error bars are 95% bootstrap confidence intervals.

QUESTION

Q2: Among several candidate corrections, can the successful one be identified?

This is precisely what recent test-time scaling methods for embodied models set out to do: sample several candidates from the embodied model at the same state and let a scorer keep the best one. We find that, after a failure, none of them can reliably pick the successful candidate.

Protocol. Using Cosmos Policy, we save the simulator at a chosen state, restart the policy from that save eight times with different random seeds, and run each candidate for 32 steps, labeling it by whether the object ends up grasped. Within a state we pair every successful candidate with every failed one and report the fraction of pairs in which the scorer ranks the successful one higher, so 0.5 is random. We evaluate the value head of Cosmos Policy, Consistency-Consensus, Consistency-Exploring, GeoBoN and RCS, together with a random baseline that should land at 0.5. As a reference from outside the embodied model, we also score the same candidates with the VLM judge of Spotter (Section 3.2), Qwen or GPT-6 Astra, given only images around the saved state (Appendix A).

Post-failure and normal states. In the setting the scorers would be used in, the save is taken the instant a grasp has failed, so the candidates are attempts to recover, some of which succeed. As a control showing what the scorers can do when conditions are in their favor, the save is taken one chunk before a grasp the policy completed successfully, on an ordinary trajectory, where some candidates grasp the object as usual and others miss.

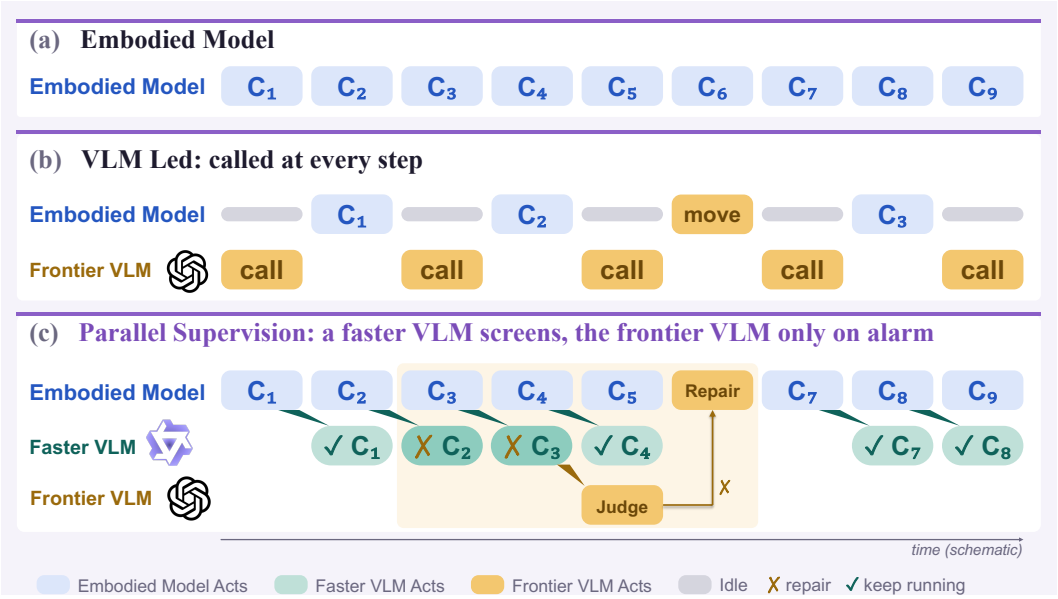


Figure 3: Three ways to combine an embodied model with a VLM over time (block widths are schematic). (a) The embodied model acts alone. (b) VLM-led: the frontier VLM is called before every step, and the embodied model idles while it waits. (c) Spotter: a faster VLM screens each chunk in parallel, one chunk behind execution, and, on the routine path illustrated here, the frontier VLM is called after two consecutive flags. Once C_2 and C_3 are both flagged, the judge deliberates while the embodied model executes C_5; it is interrupted only when the error is confirmed, at C_6, for the repair, then resumes. For clarity each drawn check covers one chunk and the lead limit K is not drawn; in practice a delayed check takes in every completed chunk not yet checked, so none is discarded.

Figure 2(b) reports the outcome: at normal states the scorers have some skill, and after a failure they lose it. GeoBoN and RCS hover around random in both settings, while the value head of Cosmos Policy and Consistency-Consensus, clearly above random at normal states, show a large gap between the two settings, dropping sharply once the policy has failed.

Both results trace back to the embodied model: learned from successful demonstrations, which lack the states that follow a failure, it rarely produces a correction there, and recovery data collected from it are too scarce to supply one. Judging a correction, however, is hard even for a model that did not learn from these demonstrations when it sees only the current state: given the same images, the VLM judge orders success above failure 53% of the time after a failure with GPT-6 Astra and 51% with Qwen, against 57% and 53% at normal states. Candidates from the same state differ only slightly, and a single view does not show what has already been tried. Telling a recovery from a repeated failure takes the history of the episode: the failed attempt, what came before it, and text such as the robot's telemetry and the actions so far. We therefore bring in a vision-language model that reads a much longer stretch of the episode, including the failed attempt, together with this text, and that generalizes well enough to correct the embodied model's errors: instead of instructing the embodied model or choosing among its samples, it composes a repair plan over motion primitives and checks in the next supervised window whether it worked (Section 3.2); Section 5 measures how often it corrects an error.

3.2 Spotter

Problem definition. An embodied model π performs a manipulation task described by a language instruction ℓ: at each step t it receives RGB images I_t and the robot state q_t (end-effector pose and gripper state), and outputs the next H actions, $a_{t:t+H} = \pi(I_t, q_t, \ell)$. The task succeeds when a condition $\mathcal{G}$, such as "the object is on the plate", becomes true; the episode ends then, or else at the step limit T set for the task. We keep π frozen and add a supervisor that sees the same images and

robot state and is never told whether $\mathcal{G}$ holds. The depth maps D_t of the same cameras are used only by the executor to lift the pixel targets of a repair to 3-D; the VLM itself never sees them. The goal is to raise the success rate of π within the same budget of T steps, while overlapping supervision with policy execution.

Parallel supervision. Figure 3 compares three designs. Run alone (a), π fails unnoticed; in VLM-led designs (b) such as Harness VLA (Zhang et al., 2026b), the VLM plans every step, so its latency is paid throughout the episode. Spotter (c) runs π (System 1) and a VLM supervisor in parallel, with continuous attention between them. Most actions, such as moving the arm to a location, are routine and handled well by π, so the supervisor only has to catch the few that go wrong: a fast, high-recall *screener* checks every chunk, and a frontier VLM (System 2), the *judge*, examines flagged windows and verifies the outcome of a repair. The VLM receives only what the robot's own sensors provide at deployment: the camera images and robot state that π sees and the actions π has commanded, never privileged simulator state such as object poses or the success signal. The embodied model keeps running during supervision until an error is confirmed or the bounded lead limit is reached.

Screening, repair, and reflection. Let C_i be the i-th chunk executed by π and z_i its record, the camera views and telemetry of the chunk. When π finishes C_i it moves straight on, and z_i goes to the screener. With j the latest chunk the supervisor has checked, π may start chunk i only if

$$i - j \le K, \tag{1}$$

that is, it runs at most K chunks ahead of the check; at this lead limit it stops at the chunk boundary until the check catches up. Apart from a confirmed error, this is the only time π pauses, and it bounds staleness: when the judge reaches a verdict, the robot has moved at most K chunks beyond what the judge saw. Because the screener can run slower than π, chunks may queue up, so each check takes in all chunks finished since the previous one, $z_{j+1:i}$, not only the latest. The screener S, a locally deployed Qwen model, gives a risk score and flags the chunks when the score reaches a threshold τ:

$$r_i = S(z_{j+1:i}), \qquad f_i = \mathbf{1}[\, r_i \ge \tau \ \vee \ E_i \ \vee \ F_i \,]. \tag{2}$$

Here E_i denotes empty-gripper evidence and F_i a failed or invalid screening call (Appendix B). We tune it to favor recall: τ is low, because a miss costs far more than a false alarm, and any screener failure counts as a flag. To limit false alarms, the judge J is called after two consecutive screening checks are flagged, or immediately for empty-gripper evidence under the rule in Appendix B. The next supervised window after a repair also receives an observation-only judge check. In a decision call the judge returns

$$(v, \delta, e, p) = J(z_{j+1:i}, h_i), \qquad v \in \{\textit{ok}, \textit{intervene}, \textit{giveup}\}, \tag{3}$$

where h_i is the episode history, v the verdict, δ a short free-text diagnosis, and, when it intervenes, e an expectation and $p = (c_1, \ldots, c_n)$, $c_k \in \mathcal{P}$, a plan of primitives (Table 5, Appendix C). If nothing is wrong, it returns *ok* and π continues undisturbed. Otherwise the plan runs segment by segment, and the executor lifts each image target, a pixel in a named camera view, to 3-D with its depth in D_t (Appendix C). Unless the episode has ended, after each segment the judge sees a new image and the execution log and decides to continue, declare the problem fixed, or give up; π then resumes from the repaired state, with no rollback. Budgets cap the segments per intervention and the interventions per episode, and every motion counts toward T. Every intervention also states an expectation, an observable fact for the next supervised window such as "the gripper stays open wider than 20 mm while the arm lifts". In that verification window the judge may only observe, and must first say whether the expectation held; if it did, it must not repeat the repair, and if not, its next intervention must try a different kind of repair rather than a new offset. Without this step, the judge tends to repair and let go, unaware of whether it worked.

Issues raised by asynchrony. The judge examines what the model produced up to K chunks ago (Eq. (1)); does the lag make a difference? Three cases arise. (i) *No damage*: the faulty action has not changed the environment; it is passed to the VLM, which uses it to return the arm to its earlier position, undoing the action. (ii) *Repairable damage*: an object is knocked over but still on the table, or pushed out of its original region on contact; the VLM can locate it again and complete the task, just as it could on a real robot. (iii) *Irreversible damage*: the task cannot continue, for example because the object falls to the floor out of view. This is not a limitation of our framework: if an action ruins the environment the first time it is executed, paradigms (a) and (b) cannot recover either; such cases call for a more advanced approach, possibly a world model simulating the future.

Physical reset and learning. In the first case above, Spotter returns the arm, not the scene: Spotter records the arm pose at each chunk boundary, and to redo an action the judge uses the Retreat primitive to return to the recorded takeover pose or the pre-grasp anchor, without rolling back the simulator, so the whole method runs on a real robot. The simulator's reset is used only while learning a library of general lessons, free of privileged or task-specific information (Appendix B); at test time the lessons are only retrieved, and nothing is reset.

4 EXPERIMENTS

We evaluate Spotter without an in-context demonstration (**zero-shot**, all table rows other than *1-shot*) and with a single learned demonstration (**1-shot**). In the RoboCasa implementation, both settings retrieve from the same frozen library of general lessons; 1-shot additionally retrieves one demonstration from separate learning episodes (Section 3.2). Neither resets at test time.

4.1 EXPERIMENTAL SETUP

Benchmarks and embodied models. We run three sets of experiments. (i) On RoboCasa (Nasiriany et al., 2024), a simulated benchmark and our main testbed, we use the released checkpoints of Cosmos Policy (Kim et al., 2026b) and $\pi_{0.5}$ (Black et al., 2025a) on all 24 official tasks (8 pick-and-place tasks and 16 that operate a mechanism such as a door, a drawer, or a knob), with 50 episodes each (1,200 episodes in total). (ii) On the Hard setting of RoboTwin 2.0 (Chen et al., 2026), a simulated bimanual benchmark that tests transfer to a different simulator and embodiment, we use $\pi_{0.5}$ trained on clean data only, on 50 tasks with 10 episodes each. (iii) On a real Franka Research 3 robot, we use $\pi_{0.5}$ fine-tuned on teleoperated demonstrations for three tasks (Section 4.4). Episode seeds are fixed in advance. Qwen3.8-27B (Qwen Team, 2026), served locally, is the screener, and either Qwen itself or GPT-6 Astra is the judge, denoted **Spotter (Qwen)** and **Spotter (GPT)**. Every episode is run with and without Spotter under the same scene, instruction, and step budget, raised equally for a matched comparison; repair actions consume the same episode budget as policy actions. The supervisor learns from episodes disjoint from the test episodes and never sees the success signal. Task lists, seeds, hyperparameters, the settings of Harness VLA (run with Qwen), and the sources of Figure 5 are in Appendix B.

4.2 MAIN RESULTS

Table 1 reports success rates on RoboCasa. A longer budget alone helps little, adding 1.2 points for Cosmos Policy and 1.8 for $\pi_{0.5}$ at $1.8\times$ the step limit, and detecting the failure is not enough either: with a fixed retry, in which the arm retreats and the embodied model tries again, success stays at the level of the embodied model even with the oracle as detector, consistent with Section 3.1. Spotter improves both models: with the GPT judge and one learned example it reaches 73.5 and 71.8, 5.6 and 7.5 points above the embodied model under the same budget, mainly on pick-and-place (10.3 and 9.0 points, against 3.3 and 6.7 on the other tasks), where a missed grasp is common and can be retried. Harness VLA with Qwen, which consults the VLM at every round and is given 5,000 steps, reaches 71.3 and 69.1 when its VLM can also see the task's success predicate, as in its released code, but without it falls by 6.1 and 8.6 points, most sharply on the tasks that operate a mechanism (9.2 and 11.7 points), to 65.2 and 60.5, below the embodied model alone. A VLM-led design relies on the language model to drive every step: the original work uses the Codex and Claude Code agents as planners, and a weaker model such as Qwen works only with privileged information. Spotter needs neither: with the same Qwen model and no privileged information it gains 3.8 and 4.1 points, and with one learned example it exceeds privileged Harness VLA on both models with either judge (71.8 and 70.4 with Qwen, 73.5 and 71.8 with GPT-6 Astra). Removing the screener, so that the judge examines every supervision window, adds about two points (70.0 to 72.0 and 68.8 to 70.8), the price of keeping the frontier VLM off the common path. On RoboTwin 2.0, with longer bimanual tasks, the gap between the two judges widens (Figure 5(b), Appendix E): from $\pi_{0.5}$ at 47.2%, Spotter reaches 51.2% with Qwen and 57.0% with GPT-6 Astra, a 5.8-point gap against 1.7 and 1.4 on RoboCasa.

Table 1: Success rates (%) on RoboCasa, 24 tasks with 50 episodes each (PnP: the 8 pick-and-place tasks; Non-PnP: the 16 tasks that operate a mechanism). *Fixed retry*: on a flagged failure the arm retreats and control returns to the embodied model, with no judge. Δ: gain over the embodied model at $1.8\times$ the step limit, the budget of Spotter. Harness VLA uses 40 rounds and 5,000 steps (Appendix B); with *privileged info* its VLM can also see the task's success predicate, which Spotter never receives. Best per column in bold.

	Cosmos Policy						$\pi_{0.5}$					
Method	**PnP**	**Δ**	**Non-PnP**	**Δ**	**All**	**Δ**	**PnP**	**Δ**	**Non-PnP**	**Δ**	**All**	**Δ**
Embodied model												
-1.0× step limit	53.2		73.4		66.7		56.8		65.4		62.5	
-Screener + fixed retry	52.2		74.2		66.9		56.2		65.0		62.1	
-Oracle + fixed retry	53.5		73.9		67.1		56.0		66.6		63.1	
-1.8× step limit	55.2		74.2		67.9		58.0		67.5		64.3	
Harness VLA (Qwen)												
-privileged info	57.5	+2.3	**78.2**	+4.0	71.3	+3.4	62.2	+4.2	72.5	+5.0	69.1	+4.8
-no privileged info	57.5	+2.3	69.0	−5.2	65.2	−2.7	60.0	+2.0	60.8	−6.7	60.5	−3.8
Spotter (Qwen)												
-short context	59.0	+3.8	74.5	+0.3	69.3	+1.4	63.5	+5.5	68.9	+1.4	67.1	+2.8
-full context	59.3	+4.1	77.9	+3.7	71.7	+3.8	63.2	+5.2	71.0	+3.5	68.4	+4.1
-1-shot	63.5	+8.3	76.0	+1.8	71.8	+3.9	65.2	+7.2	73.0	+5.5	70.4	+6.1
Spotter (GPT)												
-with screener	58.8	+3.6	75.6	+1.4	70.0	+2.1	65.0	+7.0	70.8	+3.3	68.8	+4.5
-no screener	61.3	+6.1	77.4	+3.2	72.0	+4.1	**67.8**	+9.8	72.2	+4.7	70.8	+6.5
-1-shot	**65.5**	+10.3	77.5	+3.3	**73.5**	+5.6	67.0	+9.0	**74.2**	+6.7	**71.8**	+7.5

Table 2: Time and tokens per episode, overall and on successful and failed episodes. *Async*: wall-clock time with parallel supervision; *Sync*: time if supervision blocked execution, as in Harness VLA (with privileged info) at every round. Cosmos Policy and $\pi_{0.5}$: the models alone at $1.8\times$ the step limit. Details in Appendix B.

	Per episode			Successful episodes			Failed episodes		
Method	Async (s)	Sync (s)	Tokens (k)	Async (s)	Sync (s)	Tokens (k)	Async (s)	Sync (s)	Tokens (k)
Cosmos Policy	63	–	–	34	–	–	125	–	–
Spotter (Qwen)									
-short context	158	196	82	54	77	18	395	467	227
-full context	181	217	87	47	71	9	521	587	283
-1-shot	173	210	218	62	86	33	454	524	689
Spotter (GPT)									
-with screener	–	–	156	–	–	48	–	–	411
-no screener	–	–	456	–	–	232	–	–	1030
-1-shot	–	–	226	–	–	54	–	–	705
Harness VLA (Qwen)	–	260	135	–	146	46	–	545	356
$\pi_{0.5}$	37	–	–	23	–	–	62	–	–
Spotter (Qwen)									
-short context	70	99	43	27	43	9	158	212	113
-full context	126	156	77	39	58	11	313	370	220
-1-shot	126	156	170	41	61	26	328	382	514
Harness VLA (Qwen)	–	253	170	–	164	84	–	451	362

4.3 Cost of Supervision

Table 2 reports time and tokens per episode (the judge calls and repairs behind them are in Table 4, Appendix B). On successful episodes Spotter stays close to the embodied model alone: with Qwen, the judge is consulted less than once per episode and almost never repairs, so such an episode takes only 13 s longer on Cosmos Policy and 16 s on $\pi_{0.5}$, 3% and 6% of the extra time of a failed episode, which takes $4.2\times$ and $5.0\times$ as long as without Spotter. The small local screener does much of the work: execution continues while chunks are checked, hiding 36 to 38 s of supervision per episode on Cosmos Policy and 29 to 30 s on $\pi_{0.5}$ (about 60% and 80% of the policy's own execution time), and with GPT-6 Astra it cuts judge calls per episode by 37% and tokens by 66% at a cost of 2.0 points of success rate. Appendix E analyzes the cost of the single demonstration.

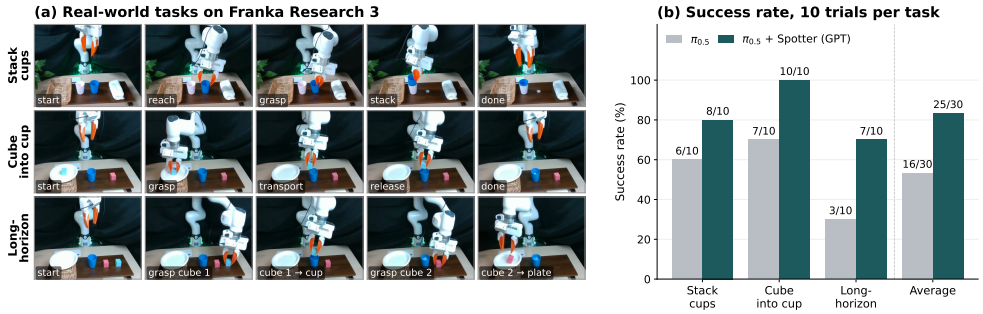


Figure 4: Real-robot validation on a Franka Research 3. (a) The three tasks (Section 4.4). (b) Success rate over 10 trials per task for $\pi_{0.5}$ alone and with Spotter (GPT), 1-shot.

4.4 Real-Robot Validation

We fine-tune $\pi_{0.5}$ on teleoperated demonstrations for three tasks on a Franka Research 3: stacking cups, placing a cube into a cup, and a long-horizon task that chains two subtasks (one cube into the cup, then another onto the plate). Adding Spotter (GPT) raises average success from 53% (16/30) to 83% (25/30), and the largest gain is on the long-horizon task (30% $\rightarrow$ 70%), where a single mistake in either subtask is enough to fail the task without intervention (Figure 4). Videos of the real-robot runs are provided in the supplementary material.

5 Analysis

Why VLM-led supervision is slow where it is not needed. The gap to Harness VLA is largest where supervision matters least: on successful Cosmos Policy episodes, Harness VLA with Qwen takes 146 s and 46k tokens, against 47 to 62 s and 9k to 33k for Spotter (Qwen) and 34 s for the policy alone (Table 2). The embodied model could have finished these episodes alone, but a VLM-led planner must fix how long each call runs before seeing how it turns out, so it stops to check even when nothing is wrong, spending so many steps that under the $1.8\times$ limit its success rate falls to about 50%, with 1,802 steps per episode on average (Appendix B). Checking less means running the embodied model longer between calls, so an error early in a call goes unnoticed until the call returns, by which time the robot may have gone far down a failure. Spotter avoids this choice: every chunk is included in a screening check, with the supervisor trailing execution by at most K chunks (Eq. (1)), while frontier VLM calls are reserved for flagged windows and repair verification, keeping successful episodes only 13 to 28 s longer than with the policy alone.

How often does the judge correct an error? Asked the same question as the embodied model was in Section 3.1, the judge, which finds the error itself, intervenes where it should, and a single repair corrects about 10% to 20% of the errors, against 7.5% to 9.7% for the embodied model with three attempts and an oracle-found error (Appendix E, which also analyzes the single demonstration).

6 Conclusion

Recovering from a failure takes two abilities, producing a correction and recognizing the right one, and embodied models learned only from success have neither: they rarely repair a failure even when it is caught for them, and the scorers built on them lose their skill once the policy has failed. What is missing is not more samples from the embodied model but a different reader of the episode: a general vision-language model that reads a longer stretch of it with telemetry and past attempts, adjusts from the failed attempt, and checks whether its repair worked. This reflection need not take control away from the embodied model: Spotter adds continuous, lightweight attention between the two, keeping the embodied model running while a cheap screener watches and calling the general model for flagged windows and repair verification, so it helps most where one mistake would end the episode and costs little elsewhere. Its lag is undone by returning the arm rather than resetting the scene, so the approach carries over to a real robot. Errors that ruin the scene the first time they occur remain out of reach; anticipating them, perhaps with a world model, is the next step toward embodied models that recover from their own mistakes.

AI USE STATEMENT

We used generative AI tools to help run the experiments, produce the figures, translate and polish the text, and search the literature; every cited paper was checked against its source by the authors. We carefully checked all code to ensure that it is correct. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not raise ethical concerns.

REPRODUCIBILITY STATEMENT

The experimental parameters are listed in detail in the appendix: the error-correction evaluation in Appendix A, the experimental setup in Appendix B, and the primitive library in Appendix C. The RoboCasa Spotter implementation and evaluation drivers are provided in the supplementary material.

REFERENCES

Christopher Agia, Rohan Sinha, Jingyun Yang, Ziang Cao, Rika Antonova, Marco Pavone, and Jeannette Bohg. Unpacking failure modes of generative policies: Runtime monitoring of consistency and progress. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 689–723. PMLR, 06–09 Nov 2025. URL https://proceedings.mlr.press/v270/agia25a.html.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, brian ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In Joseph Lim, Shuran Song, and Hae-Won Park (eds.), *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 27–30 Sep 2025a. URL https://proceedings.mlr.press/v305/black25a.html.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert 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, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A Vision-Language-Action Flow Model for General Robot Control. In *Proceedings of Robotics: Science and Systems*, LosAngeles, CA, USA, June 2025b. doi: 10.15607/RSS.2025.XXI.010.

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. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=itonej9GIV.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Yinpei Dai, Jayjun Lee, Nima Fazeli, and Joyce Chai. RACER: Rich language-guided failure recovery policies for imitation learning. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 15657–15664, 2025. doi: 10.1109/ICRA55743.2025.11127799.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Li Fei-Fei, Guanya Shi, Jiajun Wu, S. Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi Fan. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=4JRO9plGAI.

Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, Steven Bohez, Konstantinos Bousmalis, Anthony Brohan, Thomas Buschmann, Arunkumar Byravan, Serkan Cabi, Ken Caluwaerts, Federico Casarini, Oscar Chang, Jose Enrique Chen, Xi Chen, Hao-Tien Lewis Chiang, Krzysztof Choromanski, David D'Ambrosio, Sudeep Dasari, Todor Davchev, Coline Devin, Norman Di Palo, Tianli Ding, Adil Dostmohamed, Danny Driess, Yilun Du, Debidatta Dwibedi, Michael Elabd, Claudio Fantacci, Cody Fong, Erik Frey, Chuyuan Fu, Marissa Giustina, Keerthana Gopalakrishnan, Laura Graesser, Leonard Hasenclever, Nicolas Heess, Brandon Hernaez, Alexander Herzog, R. Alex Hofer, Jan Humplik, Atil Iscen, Mithun George Jacob, Deepali Jain, Ryan Julian, Dmitry Kalashnikov, M. Emre Karagozler, Stefani Karp, Chase Kew, Jerad Kirkland, Sean Kirmani, Yuheng Kuang, Thomas Lampe, Antoine Laurens, Isabel Leal, Alex X. Lee, Tsang-Wei Edward Lee, Jacky Liang, Yixin Lin, Sharath Maddineni, Anirudha Majumdar, Assaf Hurwitz Michaely, Robert Moreno, Michael Neunert, Francesco Nori, Carolina Parada, Emilio Parisotto, Peter Pastor, Acorn Pooley, Kanishka Rao, Krista Reymann, Dorsa Sadigh, Stefano Saliceti, Pannag Sanketi, Pierre Sermanet, Dhruv Shah, Mohit Sharma, Kathryn Shea, Charles Shu, Vikas Sindhwani, Sumeet Singh, Radu Soricut, Jost Tobias Springenberg, Rachel Sterneck, Razvan Surdulescu, Jie Tan, Jonathan Tompson, Vincent Vanhoucke, Jake Varley, Grace Vesom, Giulia Vezzani, Oriol Vinyals, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Fei Xia, Ted Xiao, Annie Xie, Jinyu Xie, Peng Xu, Sichun Xu, Ying Xu, Zhuo Xu, Yuxiang Yang, Rui Yao, Sergey Yaroshenko, Wenhao Yu, Wentao Yuan, Jingwei Zhang, Tingnan Zhang, Allan Zhou, and Yuxiang Zhou. Gemini robotics: Bringing ai into the physical world, 2025. URL https://arxiv.org/abs/2503.20020.

Jun Guo, Qiwei Li, Peiyan Li, Zilong Chen, Nan Sun, Yifei Su, Heyun Wang, Yuan Zhang, Xinghang Li, and Huaping Liu. Unified 4D world action modeling from video priors with asynchronous denoising. *arXiv preprint arXiv:2604.26694*, 2026.

Yanjiang Guo, Yen-Jen Wang, Lihan Zha, and Jianyu Chen. DoReMi: Grounding language model by detecting and recovering from plan-execution misalignment. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 12124–12131, 2024. doi: 10.1109/IROS58592.2024.10802284.

Haoran Hao, Shahram Najam Syed, Jeffrey Ichnowski, and Jeff Schneider. FAR: Failure-aware retry for test-time recovery and continual policy improvement. *arXiv preprint arXiv:2607.01111*, 2026.

Haotian He, Zeyu Yan, Qipeng Liu, Ning Guo, and Wenzhao Lian. FAWAM: Force-aware world action models for closed-loop contact-rich manipulation. *arXiv preprint arXiv:2606.08555*, 2026.

Zheyuan Hu, Robyn Wu, Naveen Enock, Jasmine Li, Riya Kadakia, Zackory Erickson, and Aviral Kumar. RaC: Robot learning for long-horizon tasks by scaling recovery and correction. *IEEE Transactions on Robotics*, 42:2843–2860, 2026. doi: 10.1109/TRO.2026.3706552.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and brian ichter. Inner Monologue: Embodied reasoning through planning with language models. In Karen Liu, Dana Kulic, and Jeff Ichnowski (eds.), *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782. PMLR, 14–18 Dec 2023. URL https://proceedings.mlr.press/v205/huang23c.html.

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

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R 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. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 06–09 Nov 2025. URL https://proceedings.mlr.press/v270/kim25c.html.

Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, and Jinwei Gu. Cosmos Policy: Fine-tuning video models for visuomotor control and planning. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=wPEIStHxYH.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as Policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500, 2023. doi: 10.1109/ICRA48891.2023.10160591.

Junbang Liang, Pavel Tokmakov, Ruoshi Liu, Sruthi Sudhakar, Paarth Shah, Rares Ambrus, and Carl Vondrick. Video generators are robot policies. *arXiv preprint arXiv:2508.00795*, 2025.

Zijun Lin, Jiafei Duan, Haoquan Fang, Dieter Fox, Ranjay Krishna, Cheston Tan, and Bihan Wen. FailSafe: Reasoning and recovery from failures in vision-language-action models. *arXiv preprint arXiv:2510.01642*, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-Refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. RoboCasa: Large-Scale Simulation of Household Tasks for Generalist Robots. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024. doi: 10.15607/RSS.2024.XX.050.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026. URL https://qwen.ai/blog?id=qwen3.8.

Bo-Kai Ruan, Teng-Fang Hsiao, Ling Lo, and Hong-Han Shuai. Is the future compatible? Diagnosing dynamic consistency in world action models. *arXiv preprint arXiv:2605.07514*, 2026.

Lucy Xiaoyang Shi, Brian Ichter, Michael Robert Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, Adrian Li-Bell, Danny Driess, Lachy Groom, Sergey Levine, and Chelsea Finn. Hi Robot: Open-ended instruction following with hierarchical vision-language-action models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 54919–54933. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/shi25d.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

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.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

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

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie. Self-evaluation guided beam search for reasoning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 41618–41650. Curran Associates, Inc., 2023. doi: 10.52202/075280-1802. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/81fde95c4dc79188a69ce5b24d63010b-Paper-Conference.pdf.

Chuyan Xiong, Chengyu Shen, Xiaoqi Li, Kaichen Zhou, Jiaming Liu, Ruiping Wang, and Hao Dong. Autonomous interactive correction MLLM for robust robotic manipulation. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 3139–3156. PMLR, 06–09 Nov 2025. URL https://proceedings.mlr.press/v270/xiong25a.html.

Zewei Ye, Weifeng Lu, Minghao Ye, Tao Lin, Shuo Yang, Junchi Yan, and Bo Zhao. RoboFAC: A comprehensive framework for robotic failure analysis and correction. *arXiv preprint arXiv:2505.12224*, 2025.

Yi Yu and Xinchuan Qiu. Benchmarking vision-language-action models on SO-101: Failure and recovery analysis. *arXiv preprint arXiv:2606.08881*, 2026.

Qi Zhang, Shaopeng Zhai, Shengzhe Zhang, Litao Liu, Tianyi Zhang, Fuxian Huang, and Ming Zhou. A generalist pair-wise progress critic model for vision-language-action robots. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=i7mfaYYLDf.

Yihao Zhang, Yuankai Qi, and Xi Zheng. Experiences from benchmarking vision-language-action models for robotic manipulation. *arXiv preprint arXiv:2511.11298*, 2025.

Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, et al. Harness VLA: Steering frozen VLAs into reliable manipulation primitives via memory-guided agents. *arXiv preprint arXiv:2607.08448*, 2026b.

Ganlong Zhao, Zijia Tang, Xingping Chen, Zhanghui Kuang, Ye Tian, and Guanbin Li. FLARE: A failure-aware framework for autonomous correction and recovery in visual-language robotic manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22391–22401, June 2026a.

Zesen Zhao, Minkyoung Cho, Hui Shen, Boyuan Zheng, Kunxiao Gao, Yulong Cao, and Z. Morley Mao. Test-time scaling for world action models via zero-shot geometric evaluation, 2026b. URL https://arxiv.org/abs/2607.17454.

Ruijie Zheng, Jing Wang, Scott Reed, Johan Bjorck, Yu Fang, Fengyuan Hu, Joel Jang, Kaushil Kundalia, Zongyu Lin, Loïc Magne, Avnish Narayan, You Liang Tan, Guanzhi Wang, Qi Wang, Jiannan Xiang, Yinzhen Xu, Seonghyeon Ye, Jan Kautz, Furong Huang, Yuke Zhu, and Linxi Fan. FLARE: Robot learning with implicit world modeling. In Joseph Lim, Shuran Song, and Hae-Won Park (eds.), *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 3952–3971. PMLR, 27–30 Sep 2025. URL https://proceedings.mlr.press/v305/zheng25a.html.

Pengfei Zhou, Shengcong Chen, Di Chen, Jiaxu Wang, Rongjun Jin, Bingwen Zhu, Yike Pan, Songen Gu, Kuanning Wang, Shufeng Nan, et al. τ_0-WM: A unified video-action world model for robotic manipulation. *arXiv preprint arXiv:2606.01027*, 2026.

Chuning Zhu, Raymond Yu, Siyuan Feng, Benjamin Burchfiel, Paarth Shah, and Abhishek Gupta. Unified World Models: Coupling Video and Action Diffusion for Pretraining on Large Robotic Datasets. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2025. doi: 10.15607/RSS.2025.XXI.015.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. RT-2: Vision-language-action models transfer web knowledge to robotic control. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

A Implementation Details of the Error-Correction Evaluation

This appendix gives the details of the two experiments in Section 3.1. Both use the official checkpoints of Cosmos Policy (2B, fine-tuned on RoboCasa) and $\pi_{0.5}$, run closed-loop in RoboCasa. All privileged quantities below (object poses, gripper–object distance, grasp state, and the task success predicate) are read from the simulator by the evaluation harness only. They are never shown to the embodied model or to the Spotter supervisor; the evaluation harness uses the success predicate only for scoring and episode termination.

A.1 Producing a correction

Failure detection. After every chunk the oracle checks whether a grasp was attempted and failed. A grasp counts as attempted when the gripper has come within 5 cm of the target object and its aperture has changed by at least 0.008 within the chunk. It counts as failed when the object is not held at the end of the chunk, or was held at the midpoint of the chunk and has been dropped by its end.

Resampling. The simulator state is saved at every chunk boundary. When a failure is detected, the state at the start of the failed chunk is restored and the model is queried again from that state with a new random seed; the resulting rollout is executed to the end of the episode under the original step budget. This is repeated for three seeds, and all three rollouts are executed. The failure counts as corrected if the object is lifted in any of the three.

Language feedback. When a failure is detected, execution stops without rewinding. A VLM is shown the current camera images and the task instruction, is told that the grasp has failed, and is asked for the cause of the failure and for a short imperative description of what to do differently. This text is given to the embodied model through its language input, and execution resumes. If the next grasp attempt also fails, the procedure repeats, for at most three rounds; success in any round counts as corrected. We run this protocol with $\pi_{0.5}$, whose language input accepts free-form text.

Metric. Let $\mathcal{F}$ be the set of oracle-detected failures, with $|\mathcal{F}| = 320$ in every setting, drawn from 1,200 episodes per model, and let $s_k(f) \in \{0, 1\}$ indicate that attempt k after failure f lifts the object. The correction rate is

$$\mathrm{CR} = \frac{1}{|\mathcal{F}|} \sum_{f \in \mathcal{F}} \max_{k \le 3} s_k(f). \tag{4}$$

A.2 Judging a correction

Saved states and candidates. While the policy runs, the harness evaluates a trigger condition at every chunk boundary. When it fires, the full simulator state is saved. From each saved state we restore the simulator and sample eight candidates from the same policy and checkpoint with different random seeds. Each candidate is executed in the real simulator for 32 steps: the 16 steps of the sampled chunk, followed by 16 steps of the continuation of the same plan. A candidate is labeled by the benchmark's official grasp predicate at step 32, and a success observed at any earlier step is kept.

Triggers. In the *post-failure* regime the trigger is the failure condition defined above: the policy attempted a grasp in the chunk that just ended and did not succeed, so the state is saved at the instant of the failure. In the *normal* regime the trigger is the first chunk of an episode in which the target becomes grasped, and the state is saved at the start of that chunk, that is, just before the successful grasp; it fires at most once per episode. The saved state therefore lies on an ordinary successful trajectory. The two regimes share the candidate count, the horizon, the labeling rule, and the scorers, and differ only in the trigger.

Metric. Only states whose eight candidates contain both a success and a failure can be used, since ranking is undefined otherwise. For these states we form the set $\mathcal{P}$ of every (successful, failed) pair of candidates (c^+, c^-) from the same state and report, for a scorer r, the fraction of pairs in which it scores the successful candidate higher, with ties counted as one half:

$$\mathrm{Acc}(r) = \frac{1}{|\mathcal{P}|} \sum_{(c^+,c^-)\in\mathcal{P}} \Big(\mathbf{1}\big[r(c^+) > r(c^-)\big] + \tfrac{1}{2}\,\mathbf{1}\big[r(c^+) = r(c^-)\big] \Big). \tag{5}$$

Confidence intervals are 95% intervals from 1,000 bootstrap resamples over states. The post-failure regime yields 991 usable states and 10,974 pairs, and the normal regime 432 states and 4,853 pairs. Usable states are rarer in the normal regime because candidates drawn from a state that is about to complete a grasp almost always succeed together.

Scorers. Every scorer assigns one number to a candidate at the saved state, before the candidate is executed, using only what is available at deployment, with one exception noted below. *Random* draws a uniform score and verifies that the pipeline is unbiased. *Cosmos value* is the output of the policy's own value head for the candidate. *Consistency-Consensus* is the agreement between the candidate's imagined future and the mean imagined future of the eight candidates. *GeoBoN* is the cross-view depth reprojection consistency between the imagined third-person and wrist frames, computed with a frozen geometry model. *RCS* re-noises the imagined future at a noise level from the training distribution and measures how closely the same model denoises it back. *Consistency-Exploring* compares the imagined future with the frames actually observed after the candidate is executed. It needs the real outcome and cannot be deployed; we include it as a diagnostic upper bound on what Consistency-Consensus approximates, which our design provides at no extra cost because every candidate is executed. *VLM judge* is the judge of Spotter, Qwen3.8-27B or GPT-6 Astra, asked for each candidate separately how likely it is to grasp the object, from 0 to 1. It sees only images, those of the chunk leading to the saved state and the candidate's imagined future frames, with no earlier history of the episode, no telemetry, and no retrieved lessons or demonstration.

B Detailed Experimental Setup

Tasks and seeds. On RoboCasa we use the 24 official tasks. The eight pick-and-place tasks are PnPCounterToCab, PnPCabToCounter, PnPCounterToSink, PnPSinkToCounter, PnPCounterToMicrowave, PnPMicrowaveToCounter, PnPCounterToStove and PnPStoveToCounter. The sixteen others operate doors (OpenSingleDoor, CloseSingleDoor, OpenDoubleDoor, CloseDoubleDoor), drawers (OpenDrawer, CloseDrawer), the stove (TurnOnStove, TurnOffStove), the sink (TurnOnSinkFaucet, TurnOffSinkFaucet, TurnSinkSpout), the coffee machine (CoffeeSetupMug, CoffeeServeMug, CoffeePressButton) and the microwave (TurnOnMicrowave, TurnOffMicrowave). Formal RoboCasa evaluation uses 1,200 episodes: base seed 500 with episode indices 0–49 of each task and object instances from the held-out split. The released code also retains a fixed 96-episode subset (four episodes per task at seed 500) for implementation debugging only; it does not replace the formal evaluation set. The environment seed is derived as $256(s + e)$ from the base seed s and episode index e. Example episodes are mined from separate learning runs with base seed 195; the example library is read-only during evaluation. On RoboTwin 2.0, which is simulated, we test on the official 50 tasks in the Hard setting, with seed 0 and the first 10 episodes of each task; $\pi_{0.5}$ is trained on clean data only.

Step budget. The official RoboCasa limit is 500 steps for most tasks, 300 for CoffeePressButton, 600 for PnPCounterToMicrowave, CoffeeSetupMug and CoffeeServeMug, 700 for PnPCounterToSink and CloseDoubleDoor, and 1,000 for OpenDoubleDoor. On RoboTwin 2.0 the official limits range from 400 to 1,700 steps. The main RoboCasa comparison applies the same $1.8\times$ multiplier to the official limit for the baseline and Spotter. After initial scene stabilization, every executed

environment step counts toward this shared budget, including repair motions, gripper actions and controller calibration.

Chunks and screening. The default is bounded asynchronous supervision. Cosmos Policy executes 16 steps per inference, one chunk, and the screener checks every two chunks, or 32 steps. $\pi_{0.5}$ outputs 50 steps and executes the first 25, one chunk, and the screener checks every chunk. The lead limit is $K = 4$ chunks on Cosmos Policy and $K = 2$ on $\pi_{0.5}$, so the supervisor trails execution by at most 64 and 50 policy steps. Delayed checks combine the images and telemetry of completed, unchecked windows, including windows accumulated during judge deliberation. Policy chunks executed within a repair are also screened before the next repair decision. The screener flags a check if its risk score, from 0 to 10, is at least 3, if screening fails or returns an invalid score, or if the gripper closes below 12 mm in any chunk (including a fully closed, 0 mm empty grasp; the threshold is 4 mm on the mug tasks, where holding a handle leaves 6 to 12 mm). The judge is called after two consecutive flags, or at once on newly screened empty-gripper evidence if it has not intervened in the previous two checks. The next supervised window after a repair is checked by the judge even when the screener does not flag it; this check cannot trigger another repair. A repair has at most 8 primitives and 3 plan segments. Table 3 lists what differs between the three configurations.

Table 3: The three Spotter configurations. Telemetry: recent chunks shown as rows; older chunks are summarized. Turns: recent Qwen judge turns kept in full, in addition to the initial briefing; older turns are compacted. Context restart: the token threshold for restarting with a text recap. GPT uses server-side conversation continuation.

	Short context	**Full context**	**1-shot**
Telemetry	last 3	last 40 (Cosmos) / 10 ($\pi_{0.5}$)	as full
Turns	last 3	last 20	as full
Context restart	22k tokens	110k tokens	as full
Interventions per episode	3	8	8
Judge called after	1 flag	2 flags	2 flags
Example	none	none	1 episode

Inputs to the supervisor. The supervisor receives time-ordered montages from the left and right third-person cameras and the wrist camera, with pending windows combined when supervision is delayed. Repair decisions also receive a fresh checkpoint image. The text includes the instruction, action history, remaining budget, proprioceptive telemetry, empty-grasp events, retrieved general lessons, the primitive interface and the expectation from the previous repair. With one learned example, the prompt adds one example episode of at most 12 images. No object pose or success signal is given.

Serving. Qwen3.8-27B-FP8 runs on vLLM with prefix caching, one replica on each of four NVIDIA H20 GPUs, with a context of 131,072 tokens and up to 60 images per request (32,768 tokens and 20 images in the short-context runs). The judge uses temperature 0.2 and at most 3,000 output tokens, the screener temperature 0 and at most 120, both with thinking disabled. GPT-6 Astra is called through the Responses API with medium reasoning effort, and the conversation continues on the server side. In Spotter (GPT) the screener is still the local Qwen.

General lessons. The simulator's reset is used only while learning: on training data or on seeds other than the test seeds, we let GPT reset and retry, and summarize what it finds into a library of general lessons. General means physical knowledge that holds across tasks, for example that a long object such as a cola bottle should be grasped in the middle rather than at an end, where it tends to slip. Privileged or task-specific information, such as how many degrees to rotate in a particular scene, is removed.

Learning. The released general-lesson library contains 77 entries and is frozen during evaluation. A single demonstration, when enabled, is retrieved from a separately prepared example library. Selection prefers the same policy and task, then the same task, then a similar task, then the whole library, preferring a repair example without overriding task relevance.

Timing. Table 2 excludes scene construction. Times are omitted for GPT-6 Astra because they include network latency. Qwen token counts include prefix-cache hits. The $\pi_{0.5}$ short-context configuration screens every two chunks, or 50 steps.

Table 4: Judge calls and executed repairs per episode for every configuration, on successful and failed episodes. The $\pi_{0.5}$ short-context row screens every two chunks, or 50 steps. For Harness VLA every interaction round counts as a repair.

	Judge calls		Repairs	
Method	Succ.	Fail.	Succ.	Fail.
Cosmos Policy + Spotter (Qwen)				
-short context	1.1	10.9	0.0	1.5
-full context	0.6	10.9	0.0	3.7
-1-shot	0.9	9.3	0.1	3.1
Cosmos Policy + Spotter (GPT)				
-with screener	2.5	13.1	0.2	4.0
-no screener	5.5	17.8	0.1	2.8
-1-shot	1.4	14.2	0.2	3.2
Cosmos Policy + Harness VLA (Qwen)	–	–	7.7	39.6
$\pi_{0.5}$ + Spotter (Qwen)				
-short context	0.6	5.4	0.1	0.8
-full context	0.6	9.9	0.0	2.4
-1-shot	0.7	9.4	0.1	2.6

Compared methods. We also compare with other embodied models in Figure 5. On RoboCasa, RLDX-1 (Kim et al., 2026a) is taken from its technical report, and Cosmos Policy, GR00T-N1.5 (the N1.5 release of Bjorck et al., 2025), UWM (Zhu et al., 2025), π_0 (Black et al., 2025b), Video Policy (Liang et al., 2025), and FLARE (Zheng et al., 2025) from the Cosmos Policy paper, all under the official step limit. On RoboTwin 2.0, GR00T-N1.7 (the N1.7 release of Bjorck et al., 2025) and LingBot-VLA (Wu et al., 2026) are taken from Harness VLA (Zhang et al., 2026b), which evaluates 50 tasks with 5 seeds each, and X-WAM (Guo et al., 2026) is run by us. In the figure, Ours-Qwen and Ours-GPT are Spotter (Qwen) and Spotter (GPT) with one learned example under $1.8\times$ the step limit, on Cosmos Policy for RoboCasa and on $\pi_{0.5}$ for RoboTwin 2.0.

Harness VLA. To compare running cost, we also run Harness VLA with Qwen as its VLM. We keep its original settings but cut the maximum number of interaction rounds from 100 to 40 and raise its step budget to 5,000. Under the same step limit as the other methods its success rate falls to about 50%: its rounds spend many steps while the task is still under way, so the limit is reached halfway through. It uses 1,802 steps per episode on average, more than the $1.8\times$ limit of most tasks.

C Primitive Library

Table 5 lists the primitives from which the judge composes a repair plan. In the RoboCasa executor, a move target is either an explicit world-coordinate vector (x, y, z) or one pixel (r, c) in a selected camera view, with an optional vertical offset. For a pixel target, the executor reads the calibrated depth $d = D_t[r, c]$ and applies the inverse camera projection to $(cd, rd, d, 1)$, then adds the vertical offset. It uses that single depth sample, not a median over a bounding box; depth remains internal to the executor and is not shown to the VLM.

D Comparison with Prior Work

Table 6 summarizes how Spotter differs from the representative prior work discussed in Section 2.

Table 5: Primitives available to the judge.

Primitive	Effect
Move to point	Move the end effector to a back-projected target
Move by offset	Translate the end effector by a relative displacement
Lift	Raise the end effector
Rotate wrist	Rotate the wrist by a given angle
Set gripper	Open or close the gripper
Retreat	Return the arm to a recorded earlier pose of the judge's choice
Move both arms	Move the two arms synchronously
Act	Run the supervised embodied model π for a given chunk size
Hand back	End the takeover and return control to the embodied model

Table 6: Comparison with representative prior work. VLM-led designs wait on the external model; Spotter runs both in parallel. *Semantic*: corrects errors invisible in telemetry, such as reaching for the wrong object. *Training*: needs training beyond the off-the-shelf embodied model.

Method	Architecture	Repair	Semantic	Training
Hi Robot	VLM-led	none	–	Yes
Harness VLA	VLM-led	failure memory	Partial	No
Towards the Harness	VLM-led	diagnose, retry	Yes	No
RACER	VLM-led	language hint	Yes	Yes
Spotter (ours)	**Parallel**	**primitive plan**	**Yes**	**No**

E Additional Results

This appendix collects the comparison with other embodied models referred to in Section 4.2, the detection and correction rates of the judge summarized in Section 5, and an analysis of the single demonstration.

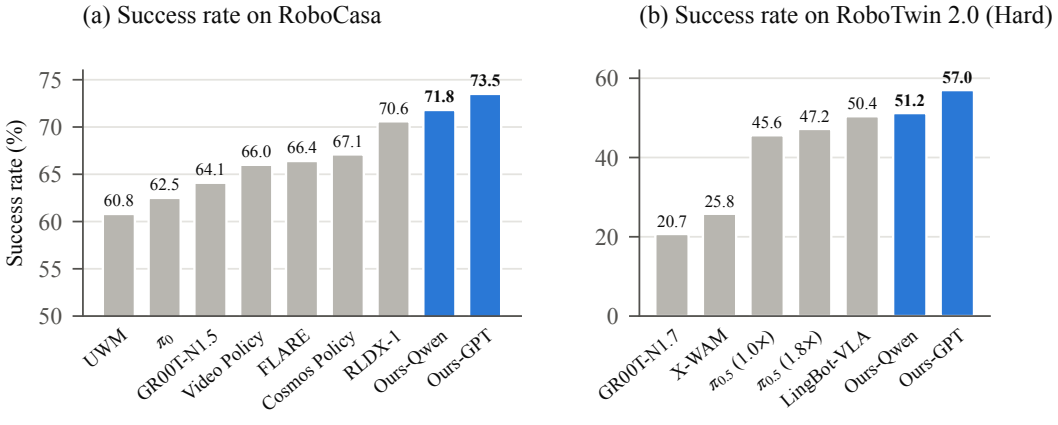


Figure 5: Success rate compared with other embodied models on (a) RoboCasa and (b) the Hard setting of RoboTwin 2.0. The compared methods are described in Appendix B.

E.1 How Often Does the Judge Correct an Error?

Section 3.1 found that, even once an oracle has caught the error, the embodied model corrects fewer than one failure in ten on its own. We ask the same of Spotter, which finds the error itself and repairs

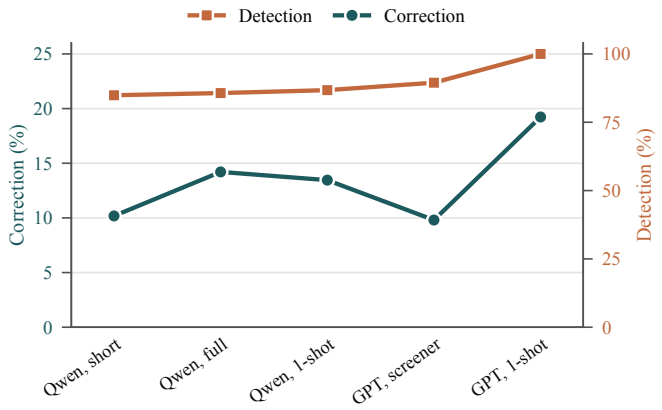


Figure 6: The judge on RoboCasa, with Cosmos Policy and $\pi_{0.5}$ pooled. Detection: share of the episodes it intervened in that the embodied model alone fails on the same scene. Correction: share of these errors corrected by a single repair, not by three attempts as in Section 3.1.

it with its own primitive plan. Let $\mathcal{I}$ be the episodes in which the judge intervened, $\mathcal{E} \subseteq \mathcal{I}$ those the embodied model alone fails on the same scene, and $\mathcal{C} \subseteq \mathcal{E}$ those in which a single repair makes the episode succeed. Figure 6 reports

$$\text{Detection} = \frac{|\mathcal{E}|}{|\mathcal{I}|}, \qquad \text{Correction} = \frac{|\mathcal{C}|}{|\mathcal{E}|}. \tag{6}$$

The judge intervenes where it should: with Qwen as both screener and judge, about 85% of the episodes it intervenes in are ones the embodied model alone fails, and GPT-6 Astra as the judge behind the Qwen screener raises this further. A single repair corrects about 10% to 20% of these errors, against 7.5% to 9.7% for the embodied model with three attempts and an oracle that had already found the error (Section 3.1); with one learned example each, GPT corrects 19.2% against 13.5% for Qwen.

E.2 The Single Demonstration

Why the single demonstration does not always show up in the average. On $\pi_{0.5}$ the demonstration helps uniformly, by exactly 2.0 points in all three columns. On Cosmos Policy the aggregate barely moves, from 71.7 to 71.8, but this hides a redistribution: the demonstration is worth $+4.2$ points on the eight pick-and-place tasks and -1.9 on the sixteen others, and since the latter carry two thirds of the weight, the two nearly cancel. It also costs $2.5\times$ the tokens (Table 2). For Cosmos Policy the demonstration therefore buys a different distribution of successes rather than more of them; for $\pi_{0.5}$ it simply buys more. The single demonstration does not lengthen episodes despite $2.5\times$ and $2.2\times$ the tokens: time per episode changes by -4% and 0%, and the judge is called 7% less often per episode on both models, mainly because the success rate rises by 2.0 points on $\pi_{0.5}$ and failed episodes need 15% fewer calls on Cosmos Policy.