Teleporting Bread and Changing Goals: Tracing a VLA and a Code-Writing Agent

Nivaldo Humberto Oliveira Tokuda
The University of Texas at Austin
ntokuda@utexas.edu

Abstract: Changing a robot's goal during execution can leave different parts of the task unfinished. We extend RoboCasa365 with scenarios that change the goal or move objects during an episode, without resetting the environment or controller. A shared simulator wrapper supports GR00T N1.5, a vision-language-action policy (VLA) that executes short action sequences, and Astra, a language-model agent that writes Python control programs and receives ground-truth state and evaluator feedback. The wrapper records when a change occurs, when updated information appears in controller inputs, and which task requirements hold afterwards. Our case study starts with placing a croissant and then a jar of jam on a cutting board. After the croissant is placed, the goal changes: return it to its original plate, but still place the jam on the board. Of 54 planned episodes, including controls, 12 were assigned this goal change and nine finished after it was applied and passed its checks. In these nine episodes, GR00T met the returned-croissant requirement at the endpoint in 0/4, although same-seed comparisons confirm that the new instruction reached the policy. Astra met this requirement at the endpoint in 5/5 and the jam requirement in 2/5, but met the full current goal in 1/5. The full-goal score requires both requirements to hold for ten consecutive simulation steps. Scoring the objects separately distinguishes failure to return the croissant from failure to place the jam. The supplement includes the episode records and scripts that regenerate every table.

Keywords: Robot simulation, continuing episodes, agent evaluation

1 Introduction

A robot is asked to place a croissant on a cutting board and then bring a jar of jam to the same board. Once the croissant is in place, the user changes the request: put the croissant back on its plate, but still bring the jam. The robot must reverse its first placement while retaining the jam-placement goal.

We study this situation in RoboCasa365 [1] using two controllers. GR00T N1.5 is a vision-language-action policy (VLA): it receives images, robot state and an instruction, then predicts a short sequence of actions, called an action chunk. Astra is a language-model agent that writes and runs Python control programs. We add a shared wrapper around the simulator so that the goal or scene can change while either controller is executing, without resetting the environment or controller.

Updating the task does not mean that the controller has received the update yet. GR00T finishes its current action chunk before requesting a new prediction. Astra's running program can read the updated task, but the language model receives it only at the next model call, after the current program segment ends. We therefore record when the task changes and when the updated information appears in the inputs to the policy, running program or language model. We also score the croissant and jam requirements separately.

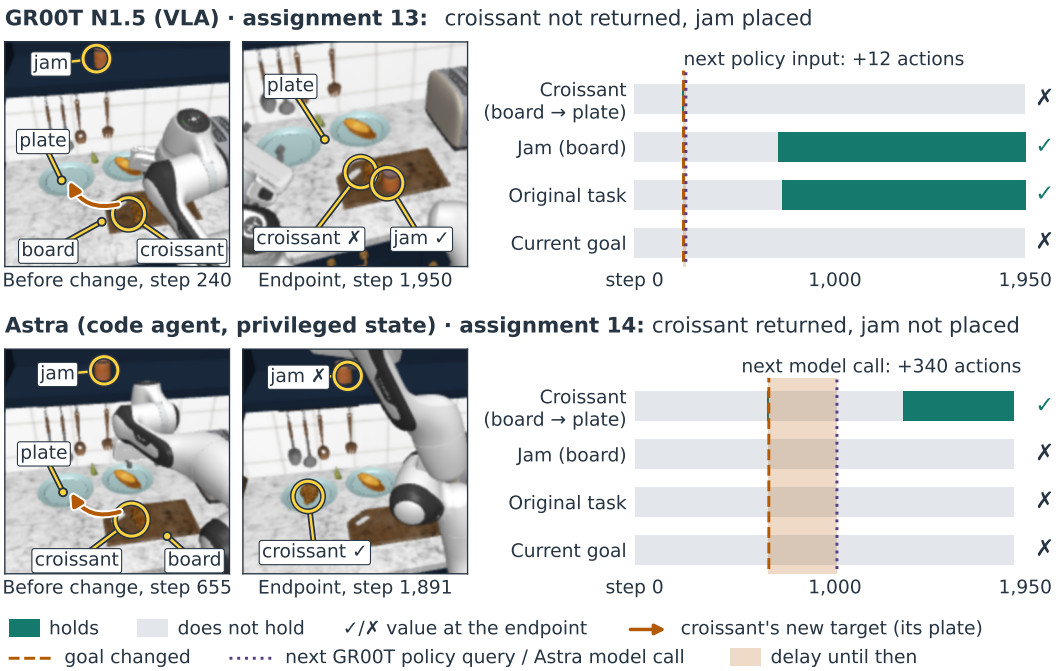

Figure 1: Assignments (episodes) 13 and 14, chosen to illustrate the two failure patterns and a long delay; Table 1 and Appendix A account for every episode. Frames: the last controller input before the change and the endpoint (step = action; left camera, cropped; labels, arrow and check/cross marks added). At the change, the croissant's target switches from the board back to its plate; the jam's stays the board. Timelines show when each requirement and score holds. GR00T's next policy query came 12 actions after the change; Astra's model saw it 340 actions later, because its running program segment continued (Section 4.3). Neither meets the current goal.

In Figure 1, GR00T completes the original task but does not return the croissant, while Astra returns the croissant but does not place the jam on the board. Both fail the updated task, but leave different parts unfinished. Scoring only the original task would count the GR00T episode as a success, even though it does not meet the new request.

The records also show a delay in the Astra episode: its language model receives the new goal 340 actions, or 17 seconds of simulated time, after the goal changes. The program it had already written continues running during this interval without checking the task. Section 4.3 traces this delay in detail (it does not establish that the delay caused the failure).

We contribute:

1. **Mid-episode scenarios for RoboCasa365.** The package supports three changes: the simulator moves a placed object back so the robot must place it again, the instruction changes so the robot must return the object itself, or the simulator relocates an object needed later. The case study evaluates the first two.
2. **A shared simulator wrapper for both controllers.** The wrapper checks for and applies changes between actions, updates the current task and records progress. The code that schedules and applies changes is separate from the controller-specific execution loops (Section 3.4).
3. **A recorded 54-episode case study.** We record when changes occur, when each controller receives updated information and which requirements hold afterwards. We account for every assigned episode, including those whose changes never triggered and one interrupted by a model-provider error.

2 Related work

Recovery and goal switching. LIBERO-Recover [8] and RoboRecover [9] evaluate whether a robot can recover from a failure state and complete the original task. Other work studies changes to the task itself. ReSteer measures whether multitask policies follow a new instruction issued during execution [16], and SwitchVLA proposes execution-aware task switching for VLAs [17]. RACER includes experiments in which the task goal changes during execution [6]. CoRe's LangSwitch evaluation scores completion of the task requested after a switch [5], while InterruptBench evaluates web-navigation agents against the user's final updated intent [7].

These evaluations also make choices about which episodes to count. CoRe's LangSwitch protocol replaces episodes whose switch never occurs or whose new goal is already satisfied when the switch occurs [5]. We retain every assigned episode in our accounting and report separately those in which the conditions for a goal change were never met. These episodes are not counted as responses to a changed goal.

Agents and evaluation tools. Code as Policies generates robot control programs from language instructions [10]. CaP-X studies coding agents that combine perception and control primitives, or reusable functions for sensing and acting [11]. ASPIRE refines robot programs through repeated execution and validation [14]. ENPIRE uses a loop of execution, outcome verification and policy improvement on real robots [15].

Language feedback can also guide execution without requiring the agent to write a program. Inner Monologue provides natural-language feedback to an LLM planner during a task [18]. In YAY Robot, user corrections can temporarily replace the high-level policy's commands to the low-level policy [19]. Hi Robot uses a high-level vision-language model to interpret instructions and user feedback, then issues commands to a lower-level VLA [20].

Harness VLA combines an agent with analytic control primitives and primitives backed by a VLA [2]. Our Astra controller uses Python programs with motion primitives adapted from its RPent implementation [2]. Astra and GR00T run as separate controllers in our study.

Inspect Robots supports LLM agents and VLAs, provides logs for inspecting their execution, and can deliver operator feedback at the next policy inference [3]. vla-eval separates model prediction from benchmark execution through reusable interfaces [4]. Goal switching and post-switch scoring are therefore already studied in prior work. Our extension applies scene and goal changes through the same RoboCasa365 wrapper for both controllers. It checks whether each change is valid, records when updated information appears in the inputs to the policy, running program or language model, and scores each task requirement separately.

3 Continuing episodes

3.1 Changes and controller inputs

A continuing episode contains one scheduled change, without resetting the environment or the controller. We record the simulator step at which the change is applied. A **valid** change passes the trigger check for a goal change or the physical checks below for a moved object. We record receipt when the updated information appears in a controller input. For Astra we distinguish the running program, which can read the update at any step, from the model, which reads it at its next call. Task scores use the saved state. Receipt alone does not establish understanding or successful control.

Both controllers act through the same simulator step (Figure 2). One **action** is one 20 Hz control step of the PandaOmron mobile manipulator, i.e. 0.05 s of simulated time. Every count in this paper uses this unit. After each action, the scenario checks its trigger, applies any due change, updates the current task and refreshes the observation. Actions already queued keep running.

The package implements three change types. **Undo** teleports the released croissant from the board back to its original place on its plate. The goal is unchanged, so the robot must place it again.

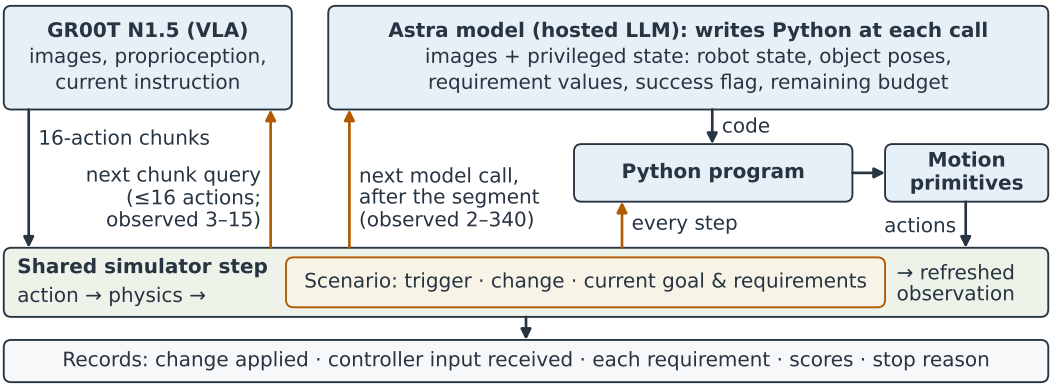

Figure 2: Both controllers act through one simulator step; the scenario owns changes and scoring. GR00T sees an update at its next chunk query, at most 16 actions later. Astra's running program can read the current task after every step, but its model sees the update only at its next call, after the current program segment ends. Orange arrows mark when an update reaches each layer, with the gaps observed after goal changes (Table 1). Offline auditors reconstruct event validity and outcomes from the records.

Goal change replaces the requested croissant destination with its original plate while keeping the jam's destination on the board (without physics changes). **Move an object needed later** relocates the untouched jam jar from the cabinet to the counter. The case study evaluates the first two. The jar scenario is a development example (Section 3.4). Physical changes are checked for support, clearance and settling, and must leave the rest of the simulator state unchanged.

The goal change fires when the croissant's original requirement (on the board, released and settled) has held for ten consecutive actions while the jam's requirement is not yet met. A **no-opportunity** episode ends without meeting this trigger. Three controls complete the design: **No change** keeps the original task. **Same text** re-sends the original wording under a new goal-version number. **From start** gives the replacement instruction from step 0, while the croissant is still on its plate.

3.2 Two scores and their requirements

The **original-task score** is RoboCasa's BreadSelection success check, evaluated after every action: both objects on the board and the gripper away from the croissant. The **current-goal score** checks the current destinations with stricter release and settling conditions, and requires both requirements to hold for ten consecutive actions (0.5 s). State is measured once per action (one **sample**), so we call this the **ten-sample rule**. The **settling rule** requires no robot contact and object speeds within fixed limits. The returned-croissant requirement also requires its original plate to stay in place. We report both scores at the endpoint, without latching earlier success (thresholds in Appendix C).

The two scores differ in both goal and completion rule. To separate the two, we re-scored the original goal with the current-goal checks and the ten-sample rule. This strict re-score agrees with the original-task score in 52/54 episodes. The two exceptions (assignments 4 and 7) had no goal change. It still credits all four finished GR00T goal-change episodes, so GR00T's goal-change failures come from the changed goal, not the stricter rule.

3.3 Controllers and protocol

GR00T N1.5 receives camera images, proprioception and the current instruction. It predicts 16-action chunks, executes each chunk fully and then queries again until the horizon. The checkpoint was post-trained on RoboCasa365's composite-unseen target data, which includes BreadSelection [13]. We do not know the instruction wording in its training data.

Astra is our harness around a hosted language model (`gpt-6-astra` at low reasoning effort, chosen to keep per-call latency and API cost manageable). At each call the model receives three camera images, robot state, ground-truth object poses, grasp flags, the written requirements with their current truth values, the current-goal success flag and the remaining action budget. It replies with a Python **program segment** that runs in a persistent namespace, or with a final answer, which ends the episode. Segments call motion primitives adapted from RPent [2]: end-effector steps, position servoing, pitch rotation, gripper commands, a scripted top-down grasp and base motion. The system prompt warns that the task can change during any action and tells the model to check object state and the success flag. Each episode starts a fresh context and allows at most 20 model calls. Harness VLA similarly shows its agent a benchmark success signal, but withholds object poses [2].

Protocol. The protocol fixes 54 **assignments** (planned episodes), numbered 1–54 in launch order. We cite episodes by this number. They cover 2 controllers, 9 condition–configuration cells and 3 repeats per cell. The assignments, their order, the scoring code and the requirement predicates were fixed before the first launch. Configurations 0 and 3 (scene seeds 0 and 3) are two object configurations in one kitchen layout, differing in the croissant, jam and distractor instances and in their initial placements. Undo covers configuration 0 only, because configuration 3 failed the development settling check. GR00T seeds 101–103 are matched across conditions. Astra's provider sampling is unseeded. Every episode allows 1,950 actions (97.5 s), which is BreadSelection's original horizon.

3.4 Adding a scenario or a controller

A scenario is called after every simulator step and reports the current instruction, goal version, requirement values and whether physics changed. The shared runtime wraps the environment's step, blocks resets, refreshes observations after a change and records every step. Two shared schedulers handle checked physical moves and goal replacements. They use a task-specific **binding** containing measurement, relocation and scoring code.

Both evaluated scenarios reuse the existing BreadSelection binding. The jar-relocation development example also reuses the runtime and controller execution logic (Appendix C). A new controller can call the wrapped environment directly, as GR00T does, or exchange step, observe and current-task messages over a local socket, as Astra's programs do. We have not measured the effort required to support a new task.

4 Results

4.1 Goal changes: accounting and outcomes

Table 1 accounts for all twelve assigned goal changes, then reports endpoints for the nine finished episodes with a valid change. GR00T had two no-opportunity episodes: in assignment 24 it placed the jam before the croissant, and in 53 it placed neither object on the board. In Astra's assignment 23 a valid change reached the model before a provider error stopped the run. Its endpoint appears in Appendix A, outside the finished-episode denominator.

GR00T: instruction received, return requirement unmet. At the endpoint, none of the four finished GR00T goal-change episodes met the returned-croissant requirement, while all four met the original-task score. Because GR00T seeds are matched, in configuration 0 each goal-change run takes the same actions as the "no-change" run with the same seed until the first action computed from the updated input (actions 257, 273 and 337; Appendix B), and differs at every action from then on. With "same text", every action stays identical. The instruction therefore reached the policy and changed its saved outputs. In the "from-start" control, where the croissant starts on its plate and the instruction asks for it there, GR00T still ended with it on the board in 3/6 episodes (4/6 under No change).

Table 1: Goal-change accounting (top) and endpoints of the finished valid changes (bottom). Requirement rows are booleans at the last sample. The current-goal score also needs the ten-sample rule. The last row counts actions from the change to GR00T's next policy input or Astra's next model call. Columns do not rank the controllers (Section 6).

Goal-change accounting and endpoints	GR00T	Astra
Assigned a goal change	6	6
No opportunity (trigger never met)	2	0
Change applied and valid	4	6
Provider interruption after the change	0	1
Finished with a valid change	4	5
Endpoints of finished valid changes	**n = 4**	**n = 5**
Original-task score	4/4	0/5
Changed requirement: croissant on plate	0/4	5/5
Kept requirement: jam on board	4/4	2/5
Both requirements at the last sample	0/4	2/5
Current goal (ten-sample rule)	0/4	1/5
Actions to next policy input / model call	3–15	2–340

Astra: croissant returned at the endpoint, jam usually unfinished. Astra met the returned-croissant requirement at the endpoint in all five finished episodes, placed the jam in two, and completed the current goal in one. Three unsuccessful episodes (5, 14 and 32) ended with a final answer near the action limit, with 93, 59 and 2 actions left and the jam never on the board. Assignment 39 reached the horizon: the jam first reached the board at action 1,941 and met its requirement at 1,948, so only three of the ten required samples fit before the horizon. Counting only the last sample would credit this episode (2/5 in Table 1). In assignment 5 the croissant requirement holds at the last sample but never for more than five consecutive samples under the replacement goal, because of settling-threshold crossings.

Budget. The 1,950-action horizon was not extended for the return trip. Astra's "no-change" runs already used 1,523–1,875 actions, and its finished unsuccessful goal-change episodes had 1,141–1,459 left at the change (the successful one had 1,793). No goal-change episode reached the 20-call limit (the most was 18). These runs do not isolate the effect of the action budget; changing the goal later also leaves less time to complete it.

4.2 Controls and Undo

Table 2: Controls and "undo", pooled over both configurations (Undo: configuration 0 only). Entries are successes / episodes under the current-goal score and the original-task score. Same-text results include valid events only, excluding two GR00T no-opportunity episodes. Its pooled GR00T subset therefore differs from No change, while the trajectory-identity check in Section 4.1 covers only configuration 0. From start begins with the croissant already on its plate and does not test returning it. The "undo" denominator keeps assignment 42, an execution error at the action limit (Appendix A).

Condition	GR00T current	GR00T original	Astra current	Astra original
No change	3/6	4/6	5/6	5/6
Same text (valid events)	2/4	3/4	5/6	5/6
From start	3/6	3/6	4/6	0/6
Undo (config 0)	3/3	3/3	2/3	2/3

In the "no-change" and "same-text" controls (Table 2), GR00T's two scores differ only because of assignments 7 and 4: both requirements hold at the end, but a one-sample angular-speed spike

restarts the count seven samples before the horizon. Counting only the last sample would credit both. "Undo" completes in all three GR00T episodes and two of three Astra episodes.

4.3 When does each layer see the change?

After a goal change, GR00T's next input came 3–15 actions later, bounded by its 16-action chunk. Astra's model received it 2–340 actions later, because the model is called only between program segments. Assignment 14 (Figure 1) shows the long case. At action 655 the model wrote a five-line segment with a servo helper it had defined earlier: three moves aimed at the jam, a gripper close and a lift. The goal changed at 670. The segment made no task query; the arm moved only about 6 cm while each servo loop ran to its step limit, and the segment ended at 1,010, so the model saw the new goal 340 actions (17 s of simulated time) after it was applied. The segment was still aiming at the jam, which the new goal kept, so the records do not show that the delay changed the outcome. In all six Astra goal-change episodes, programs read the task only to print it for the model's next call; none branched on it. For a code-writing agent, the delay depends on segment length and on whether the running program checks the task. Astra's model–program split is one small instance of the reasoning–motor boundary (open problem 1), and the goal change is one form of mid-trajectory feedback (open problem 3) [12]; unlike [19], [20], no route here places an agent over a learned policy.

5 Reporting implications

For the same nine episodes, the original-task score (4/4 vs 0/5), the changed requirement (0/4 vs 5/5) and the current goal (0/4 vs 1/5) describe different outcomes. Reporting only one would hide which part of the task remained unfinished. A change timestamp also does not establish receipt: gaps span 3–15 actions for GR00T and 2–340 for Astra's model. Finally, four scheduled goal updates never triggered (two goal changes, two Same-text re-sends), and three unsuccessful Astra episodes ended near the action limit. These cases require separate counts for assigned and applied changes, provider interruptions, and the actions left at the change and endpoint.

Two questions remain: should a running program stop when the goal version changes, and how long does an update take to reach each layer when an agent calls a VLA as a primitive?

6 Limitations

The case study covers one task, one kitchen layout, two object configurations that were also used during development (not held out), and three repeats per cell. It supports neither reliability estimates nor a controller ranking: the controllers differ in training, inputs, change timing, stopping rules and whether they are told the task may change. It does not isolate why either controller failed, whether a longer horizon would help Astra, or whether GR00T can return an already placed croissant. Astra sees evaluator feedback and ground-truth poses, so it cannot be credited with independent perception or completion recognition. The runtime supports one change per episode, and each new task needs its own binding (geometry, validity checks and scoring). The settling rule can reject negligible jitter (assignments 4 and 7), which is why requirement values are reported beside the ten-sample score. Physics pauses during inference (Appendix D), and the hosted model alias could not be pinned across collection sessions. The package checks ran only on the development host, not on an independent machine.

7 Artifact and conclusion

The supplementary material contains the evaluation source as frozen before launch, all 54 episode records with per-step measurements and controller-input receipts, the protocol and assignment list, the auditors and the analysis scripts. One standard-library Python command regenerates every table

without a simulator, assets or model calls. The included measurements and receipts also support outcome checks, such as re-scoring the original goal under the ten-sample rule. Re-executing episodes requires the RoboCasa365 assets, the GR00T checkpoint and API access. Full camera and state recordings are available on request.

In the finished goal-change episodes, GR00T completed the original task without returning the croissant, while Astra returned it but usually left the jam unfinished. Further experiments should test whether interrupting programs after goal updates or extending the post-change budget improves completion, and whether these patterns recur on other tasks and controllers.

AI-use statement. Coding agents and language models assisted with implementation, experiment orchestration, analysis scripts, literature search and manuscript drafting. The author checked the reported numbers against the episode records and the cited sources against the claims they support, and traced three episodes (13, 18 and 32) from code through records to table cells. This assistance is separate from Astra's role as an evaluated controller.

References

[1] Soroush Nasiriany, Sepehr Nasiriany, A. Maddukuri, and Y. Zhu. **RoboCasa365: A Large-Scale Simulation Framework for Training and Benchmarking Generalist Robots**. ICLR, 2026. arXiv:2603.04356.

[2] Y. Zhang et al. **Harness VLA: Steering Frozen VLAs into Reliable Manipulation Primitives via Memory-Guided Agents**. 2026. arXiv:2607.08448v5. RPent primitive source: revision e6c585dd.

[3] Robocurve. **Inspect Robots: The open-source evaluation framework for physical AI**. 2026. Documentation, live feedback.

[4] S. Choi et al. **vla-eval: A Unified Evaluation Harness for Vision-Language-Action Models**. 2026. arXiv:2603.13966v3.

[5] Y. Zhang et al. **Imagining Recovery: Inference-Time Counterfactual Realignment for Vision-Language-Action Models**. 2026. arXiv:2608.14822.

[6] Y. Dai, J. Lee, N. Fazeli, and J. Chai. **RACER: Rich Language-Guided Failure Recovery Policies for Imitation Learning**. 2024. arXiv:2409.14674.

[7] H. P. Zou et al. **When Users Change Their Mind: Evaluating Interruptible Agents in Long-Horizon Web Navigation**. 2026. arXiv:2604.00892.

[8] L. Liu et al. **LIBERO-RECOVER: Beyond Task Success Towards Failure Recovery in Robotic Manipulation Models**. 2026. arXiv:2609.05178v2.

[9] Y. Li et al. **RoboRecover: Benchmarking Robot Policy Recovery under Execution Deviations**. 2026. arXiv:2609.28952.

[10] J. Liang et al. **Code as Policies: Language Model Programs for Embodied Control**. 2022. arXiv:2209.07753.

[11] L. Fu et al. **CaP-X: A Framework for Benchmarking and Improving Coding Agents for Robot Manipulation**. 2026. arXiv:2603.22435v2.

[12] Workshop on Agentic Robotics organizers. **Open problems: the agent–policy seam and evaluating agentic robotic systems**. CoRL, 2026. Workshop call.

[13] RoboCasa core team. **Foundation model learning and target datasets**. 2026. Benchmark documentation, task registry, checkpoint repository.

[14] R. Lu et al. **ASPIRE: Agentic /Skills Discovery for Robotics**. 2026. arXiv:2607.00272.

[15] W. Xiao et al. **ENPIRE: Agentic Robot Policy Self-Improvement in the Real World**. 2026. arXiv:2606.19980.

[16] Z. Chen et al. **ReSteer: Quantifying and Refining the Steerability of Multitask Robot Policies**. 2026. arXiv:2603.17300.

[17] M. Li et al. **SwitchVLA: Execution-Aware Task Switching for Vision-Language-Action Models**. 2025. arXiv:2506.03574.

[18] W. Huang et al. **Inner Monologue: Embodied Reasoning through Planning with Language Models**. 2022. arXiv:2207.05608.

[19] L. X. Shi et al. **Yell At Your Robot: Improving On-the-Fly from Language Corrections**. 2024. arXiv:2403.12910.

[20] L. X. Shi et al. **Hi Robot: Open-Ended Instruction Following with Hierarchical Vision-Language-Action Models**. ICML, 2025. arXiv:2502.19417.

A Complete outcome accounting

Table 3 lists every supported controller/configuration/condition cell. Endpoint denominators exclude no-opportunity episodes and the provider interruption, which have their own columns, and keep execution errors. No-opportunity outcomes are not successes under a goal change.

Table 3: Outcomes per configuration. N is assigned episodes; No opportunity counts episodes whose trigger never fired; Provider counts provider interruptions. Current-goal and original-task entries are successes / included episodes. Undo exists only in configuration 0.

Controller	Config	Condition	N	No opportunity	Provider	Current goal	Original task
GR00T	0	No change	3	0	0	2/3	2/3
GR00T	0	Same text	3	0	0	2/3	2/3
GR00T	0	From start	3	0	0	2/3	1/3
GR00T	0	Goal change	3	0	0	0/3	3/3
GR00T	0	Undo	3	0	0	3/3	3/3
GR00T	3	No change	3	0	0	1/3	2/3
GR00T	3	Same text	3	2	0	0/1	1/1
GR00T	3	From start	3	0	0	1/3	2/3
GR00T	3	Goal change	3	2	0	0/1	1/1
Astra	0	No change	3	0	0	3/3	3/3
Astra	0	Same text	3	0	0	2/3	2/3
Astra	0	From start	3	0	0	2/3	0/3
Astra	0	Goal change	3	0	0	0/3	0/3
Astra	0	Undo	3	0	0	2/3	2/3
Astra	3	No change	3	0	0	2/3	2/3
Astra	3	Same text	3	0	0	3/3	3/3
Astra	3	From start	3	0	0	2/3	0/3
Astra	3	Goal change	3	0	1	1/2	0/2

Table 4 gives one row per goal-change and undo assignment. Event and input are action counts: the change, and the first model or policy input carrying it (or following the physical change). No-opportunity rows leave current-goal fields empty because no goal change was applied; their original-task outcome is shown. C / J are the croissant and jam requirements at the endpoint. Actions left are given at the event and at the endpoint. Final is a model final answer; Provider and Exec. error are recorded error classes.

Assignment 23 had a valid goal change at 175 and a model input at 202, before a provider error at action 1,321. It is one of ten valid changes but not one of the nine finished ones. Assignment 42 reached the action horizon inside a primitive; it is recorded as an execution error, not an ordinary horizon stop.

Assignment 29's strict audit failed because the physical change occurred at the end of a program segment, not within it. Separate checks of the event, restoration and scores passed. This episode does not test a change within an active program segment.

Assignment 6 carries a recording-precision correction for a contact near minus one billionth of a meter, with no score change. Neither exception changes the reported task results.

B Protocol and matched controls

GR00T uses `target_posttraining/composite_unseen/checkpoint-60000`, with recorded checkpoint revision `c484448aba1a9b60a04c9b0ca117241518ea69f3`.

The protocol, ordered assignments and scoring code were fixed before the September 17 launch. Collection paused after the 32nd episode on September 17, was closed by author decision on September 18, before the protocol's own stop time, and resumed on September 25 under two written

Table 4: Every goal-change and undo assignment. After Undo the original task stays active. The interrupted row shows the last available state, not completion under the full horizon.

ID	Controller / config	Change	Event / input	C / J	Current / original	Actions left	Stop
5	Astra/3	Goal	809/887	T/F	F/F	1141 / 93	Final
6	GR00T/3	Goal	168/176	F/T	F/T	1782 / 0	Horizon
13	GR00T/0	Goal	244/256	F/T	F/T	1706 / 0	Horizon
14	Astra/0	Goal	670/1010	T/F	F/F	1280 / 59	Final
17	Astra/0	Undo	632/633	T/T	T/T	1318 / 116	Final
18	GR00T/0	Undo	244/256	T/T	T/T	1706 / 0	Horizon
23	Astra/3	Goal	175/202	F/F	F/F	1775 / 629	Provider
24	GR00T/3	No event	—	—	—/T	— / 0	Horizon
29	Astra/0	Undo	579/579	T/T	T/T	1371 / 112	Final
30	GR00T/0	Undo	333/336	T/T	T/T	1617 / 0	Horizon
31	GR00T/0	Goal	333/336	F/T	F/T	1617 / 0	Horizon
32	Astra/0	Goal	645/647	T/F	F/F	1305 / 2	Final
39	Astra/0	Goal	491/626	T/T	F/F	1459 / 0	Horizon
40	GR00T/0	Goal	257/272	F/T	F/T	1693 / 0	Horizon
41	GR00T/0	Undo	257/272	T/T	T/T	1693 / 0	Horizon
42	Astra/0	Undo	620/916	F/T	F/F	1330 / 0	Exec. error
53	GR00T/3	No event	—	—	—/F	— / 0	Horizon
54	Astra/3	Goal	157/180	T/T	T/F	1793 / 1128	Final

amendments (the second covering assignments 50–54) that kept the frozen sources and the original launch order. No failure or no-opportunity episode was replaced. Astra called the hosted alias `gpt-6-astra` on September 17 and September 25, 2026 (UTC); contexts were fresh in every episode, but the alias could not be pinned to a fixed model version. The amendment does not claim an identical host: a sandbox package was upgraded on September 19 and the September 17 GPU driver was not recorded; the seed-103 pair, identical before receipt across the gap, is one direct check. As the amendment planned, Table 5 splits outcomes by collection session. Goal and physical auditors reconstruct the saved scenarios; controller auditors check the chain from delivered information to controller input. Fixed-action runs required by the undo auditor are audit work, not extra controller episodes.

The matched-action comparison uses configuration 0, seeds 101–103. For each seed, the No-change and Goal-change trajectories have identical action arrays through the first updated input, and the Same-text run matches the No-change run throughout. The first differing actions are 257, 273 and 337, and 1,614–1,694 of the 1,950 actions differ in total. The seed-103 pair spans the collection gap; equality before receipt is checked rather than assumed. These comparisons concern saved outputs and do not diagnose internal language processing.

The original-goal check reads the saved measurements, applies the frozen board-placement, no-contact, speed and gripper predicates, excludes initialization as a sample, and counts ten consecutive samples. It changes no official result. Its only endpoint disagreements with the original-task predicate are assignments 4 and 7. The requirement-maximum diagnostic resets when the goal version changes; otherwise a maximum can mix distinct destinations. This corrected diagnostic leaves every official score unchanged.

C Instructions, thresholds and the goal-change scenario

The original BreadSelection instruction is: “From the different types of pastries on the counter, select a croissant and place it on the cutting board. Then retrieve a jar of jam from the cabinet and place it alongside the croissant on the cutting board.” The replacement instruction is: “Place the croissant back on its original plate. Place the jam on the cutting board. Release the objects and move

Table 5: Outcomes by collection session: assignments 1–32 (September 17) and 33–54 (September 25). N is assigned episodes; current-goal and original-task entries are successes / included episodes, excluding no-opportunity episodes and the provider interruption as in Table 3. Cells are too small to compare sessions.

Controller	Condition	N 1–32	Current	Original	N 33–54	Current	Original
GR00T	No change	3	2/3	3/3	3	1/3	1/3
GR00T	Same text	4	2/3	3/3	2	0/1	0/1
GR00T	From start	3	2/3	1/3	3	1/3	2/3
GR00T	Goal change	4	0/3	3/3	2	0/1	1/1
GR00T	Undo	2	2/2	2/2	1	1/1	1/1
Astra	No change	3	2/3	2/3	3	3/3	3/3
Astra	Same text	4	4/4	4/4	2	1/2	1/2
Astra	From start	3	1/3	0/3	3	3/3	0/3
Astra	Goal change	4	0/3	0/3	2	1/2	0/2
Astra	Undo	2	2/2	2/2	1	0/1	0/1

the gripper away." The Same-text control increments the goal version while keeping the original text and goal.

Thresholds. Each placed object must have no robot contact and speeds of at most 0.01 m/s and 0.1 rad/s. The gripper must be more than 0.25 m from the croissant, measured from the end-effector to the croissant's origin. The returned croissant must rest on its original plate and not on the board. The plate must stay on its original support within 0.005 m translation, 0.02 rad rotation and 0.05 rad tilt, with speeds of at most 0.005 m/s and 0.05 rad/s. Both current requirements must hold for ten consecutive actions; the score is read at the endpoint and never latched.

Where the thresholds come from. The 0.25 m distance is the one used by RoboCasa's own BreadSelection check. The speed limits were set before the evaluation, from configuration-0 development runs: across 424 contact-based release candidates, median croissant speeds were about 10^{-9} m/s but maxima reached 0.18 m/s and 5.3 rad/s, so contact alone does not show that an object has come to rest. These development measurements motivate a settling check but do not uniquely determine the chosen speed limits. The return target follows the plate's current pose rather than a stored coordinate, and the plate limits ensure that the plate itself stays where it was.

The goal-change scenario. The scenario-specific part of the evaluated goal change is one 103-line file: a configuration check (not shown) and the two classes below. It builds on the existing BreadSelection binding. An excerpt of the frozen source follows (some lines elided; comments ours):

```
class BreadGoalTask:
    object_name = "croissant"
    replacement_instruction = (...)                    # quoted above

    def __init__(self, env):
        self.original = UndoTask(env, "BreadSelection")  # existing binding
        self.binding = self.original.binding
        ...                                            # plate region, checks

    def definitions(self, replacement): ...            # requirement text

    def measure(self):
        row = self.original.measure()                  # Bread measurements
        row["source_plate_geometry_reasons"] = ...     # plate support check
        return row

    def requirements(self, row, replacement):
        result = self.original.requirements(row)       # original requirements
```

```
        if replacement:
            result["croissant"] = bool(
                not row["croissant_on_board"]
                and not row["source_plate_geometry_reasons"]
                and ...        # no contact; gripper distance; object and
            )                  # plate speeds; plate displacement, rotation, tilt
        return result

class BreadGoalScenario(PlacementGoalScenario):          # shared trigger, scoring

    def __init__(self, env, config, output_dir):
        super().__init__(env, config, output_dir, BreadGoalTask(env))
```

Its configuration sets the trigger type and the two ten-action windows. The shared scheduler applies the change once, increments the goal version, swaps the instruction and requirements, and restarts the ten-sample completion count.

The jar addition. Its preserved source inventory separates scenario code and configuration (222 + 10 lines), new auditors (380 + 204), new test files (96 + 54), dispatch and runner inventory changes (+4 and +2) and a shared audit helper (+46 / -18). It reuses the existing Bread score and relocation lifecycle. An audited GR00T development run did not place the jar. The hand-written assisted program that retrieved the relocated jar in development is separate, task-specific controller work: it used the observed jar pose and a stored grasp pose; three attempts failed, and a fourth, which repeated the third and added a release rule declared after its 5 mm miss (4 mm tolerance), completed the task. These jar runs are development examples outside the 54-episode evaluation. Three separate one-episode information probes from September 25 (removing evaluator predicates, or vision only) are recorded separately in the supplement; they do not isolate the effect of removing information and are not part of the 54 episodes.

D Recorded inference timing

The timing analysis covers only the 32 episodes collected on September 17, not the final 54. Table 6 pools successful warm calls, excluding each episode's first call and the failed provider request. GR00T's instrumented timer includes input-archiving work. Astra's API-worker timer excludes process startup, SDK import and robot execution, so it is a lower bound on the full decision delay. One 16-action GR00T chunk covers 0.8 s of simulated time. Neither timer predicts performance with continuously advancing physics.

Table 6: Recorded warm-call timers, with their original 32-episode scope. The world pauses during inference; these are not real-time execution results.

Recorded timer (September 17 only)	Warm calls	Median (s)
GR00T instrumented inference	1936	0.477
Astra API-worker timer	238	6.456