BRIDGING FRONTIER REASONING AND ROBOT EXECUTION: FROM AUTONOMOUS DEMONSTRATION GENERATION TO DENSE LANGUAGE SUPERVISION

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in frontier models enable robot manipulation from only a few demonstrations, but high inference latency limits their use for real-time robot control. To bridge this gap, we study two complementary approaches that connect frontier reasoning with low-latency local execution. First, we use a frontier model to autonomously generate demonstrations that supplement human demonstrations for training a fast local policy. We augment its in-context examples with corrective demonstration segments that show how to recover from physical errors, improving generation reliability. Generation time and cost decrease as successful examples accumulate in context, suggesting a path toward more efficient data collection. At deployment, a harness combines frontier-generated instructions with a fast local policy, enabling efficient execution while preserving the frontier model's ability to guide and correct actions.

With low-latency execution delegated to the local policy, the bottleneck shifts to its capacity to reliably follow the frontier model's diverse instructions. Our second bridge introduces dense language supervision across three nested granularities—primitive, atomic, and composite—with multi-aspect descriptions at each level. Across long-horizon tasks in RoboCasa 365 and BEHAVIOR-1K, where instructions change as execution progresses, the combined supervision achieves the highest performance under both oracle and frontier-model instructors, demonstrating more reliable instruction following through the policy's language interface. Finally, we evaluate both bridges together on a crossword task that combines semantic planning and manipulation within a fixed time budget. These results support autonomous demonstration generation and dense language supervision as complementary components for connecting frontier reasoning to low-latency local execution.

1 INTRODUCTION

Robot learning has explored several approaches to translating frontier-level intelligence into physical robot actions. These approaches differ in how frontier models guide execution: they can issue instructions to a trained local policy (Shi et al., 2025), generate executable programs over control primitives (Liang et al., 2022), provide spatial guidance such as trajectories (Li et al., 2025), or produce robot commands directly (Menon et al., 2026). Recent advances in frontier models enable robot manipulation from only a few demonstrations through in-context learning, without task-specific fine-tuning. However, frontier-model inference and API latency limit the responsiveness of direct robot control, whereas practical manipulation requires high-frequency feedback and timely action adjustments for reliable physical interaction (Berman et al., 2026). This motivates our central question: how can we retain the benefits of frontier-model reasoning while delegating motor execution to a fast local policy?

The ability to manipulate objects from a few examples makes frontier models a potential source of additional robot demonstrations, complementing teleoperation-based data collection. Our first bridge explores this capability by generating manipulation trajectories and incorporating them into

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.

local-policy training. Successful demonstrations alone do not reliably support autonomous generation, particularly during precise contact operations. We augment the in-context examples with corrective demonstration segments: short demonstrations of adjustments at moments when execution encounters physical errors. As successful trajectories accumulate, we retain useful segments for reuse in subsequent episodes and examine how generation time and API cost evolve. At deployment, a harness combines frontier-generated plans with the policy's low-latency execution, allowing the frontier model to provide high-level direction while the policy generates motor actions within each execution segment.

Delegating motor execution reduces the need for frontier inference to generate actions, but makes the policy's instruction-following capabilities a central constraint. We propose dense language supervision as the next bridge, enabling the policy to interpret and execute diverse, fine-grained instructions generated by the frontier model. We annotate the same demonstrations at three nested granularities—primitive, atomic, and composite—and pair these annotations with multi-aspect descriptions to broaden instruction diversity.

We study two complementary bridges between frontier reasoning and robot execution: generating demonstrations for local-policy training and expanding the policy's language interface through dense supervision. Across four manipulation tasks, mixed human and generated demonstrations achieve the same aggregate policy success as an equally sized human-only corpus, supporting generated trajectories as a supplementary training source. Hierarchical and multi-aspect supervision together achieve the strongest results across both simulation benchmarks and instructor conditions. Finally, the crossword task demonstrates their integration, with dense supervision improving performance under dynamic spatial instructions. Together, these results support autonomous demonstration generation and dense language supervision as complementary bridges between frontier reasoning and low-latency local execution.

2 Related Work

Foundation Models for Robot Control. Foundation models guide robot execution through several interfaces, summarized in Figure 8 in Appendix F. Direct-control evaluations demonstrate manipulation capabilities alongside limitations in execution speed and precision (Menon et al., 2026; Berman et al., 2026). Hierarchical and dual-system architectures couple vision-language processing with low-level control (NVIDIA et al., 2025; Shi et al., 2025; Chen et al., 2026a; Cheng et al., 2025), while other approaches provide spatial guidance (Li et al., 2025; Shridhar et al., 2025; Lee et al., 2026) or synthesized goal images (Black et al., 2024). Hi Robot fine-tunes a high-level VLM using labeled demonstrations and synthetic language interactions, and a low-level VLA using robot demonstrations (Shi et al., 2025). Harness frameworks such as Thea support feedback-driven task execution (Wang et al., 2026). Our study retains a frontier model without task-specific fine-tuning and examines how to train a local policy that executes its instructions.

Autonomous Data Generation and Policy Learning. MimicGen adapts human demonstrations to new scene configurations and embodiments (Mandlekar et al., 2023). Manipulate-Anything uses VLM-generated actions and visual verification to collect physical demonstrations for policy learning (Duan et al., 2024), while Show-Harness trains models on human and frontier-generated trajectories (Chen et al., 2026b). Program-synthesis approaches generate executable programs over perception and control primitives (Liang et al., 2022; Fu et al., 2026), with ASPIRE accumulating reusable skills (Lu et al., 2026) and ENPIRE extending agent-driven improvement to neural policies, training infrastructure, and algorithms (Xiao et al., 2026). Our first bridge studies frontier models as autonomous demonstration generators for physical manipulation. Using successful demonstrations and corrective demonstration segments as in-context references, a frontier model executes tasks with visual feedback and produces new training trajectories. We examine generation reliability, track how generation time and cost evolve as successful examples accumulate, and evaluate the resulting demonstrations as a supplementary source for local-policy training.

Language Supervision and Instruction Following. RT-H learns language-motion prediction and action generation within a single VLM, enabling corrections through language (Belkhale et al., 2024). Hu et al. (2026) show that a controller's responsiveness to varied instructions affects hierarchical execution. DeMiAn provides multi-aspect segment annotations and a learned instructor, including evaluation on composite tasks with changing atomic instructions (Kim et al., 2026).

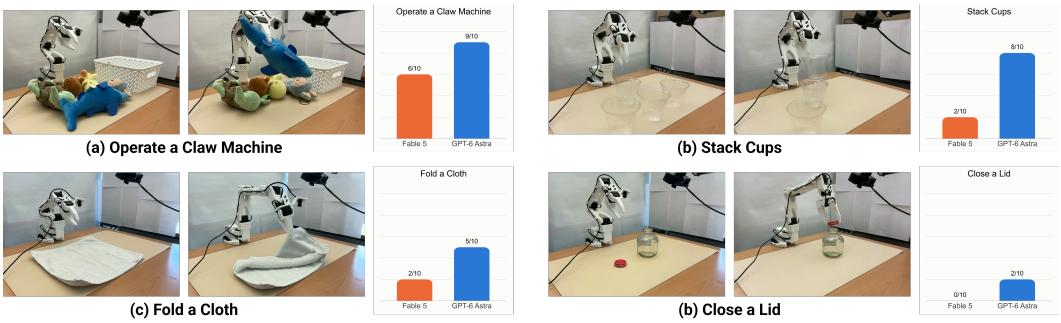


Figure 1: Autonomous manipulation from five successful in-context demonstrations. Task views and success counts are shown for Fable 5 and GPT-6 Astra across four tasks, with ten trials per model and task. Each trial starts from a reset environment and has a one-hour timeout, with no subsequent human intervention.

Building on multi-aspect supervision, our second bridge jointly annotates shared trajectories at three nested granularities—primitive, atomic, and composite—and adds alternative descriptions and rephrasings. We examine how temporal granularity and linguistic diversity affect instruction following under oracle and frontier-model instructors, with additional tests across planners and instruction phrasings.

3 Bridge 1: Distilling Frontier Reasoning into Local Control

Our first bridge uses frontier-generated demonstrations to supplement human demonstrations for training a fast local policy. We first examine autonomous generation reliability, then generation efficiency, and finally deployment through a planner–policy harness.

3.1 Autonomous Rollouts via In-Context Learning

We first evaluate the capacity of frontier models to autonomously control a robot given only a minimal set of human demonstrations. We benchmark two models, Fable 5 and GPT-6 Astra, across four tasks spanning a spectrum of physical complexity: basic pick-and-place (claw machine), spatial alignment (stacking cups), deformable manipulation (folding a cloth), and high-precision contact (closing a lid). Each trial is provided five human demonstrations with zero subsequent human intervention, and the environment is fully reset between attempts. Trials exceeding a one-hour timeout are recorded as failures. This limit applies to the capability evaluation; the subsequent data-collection runs use a separate protocol.

Figure 1 shows that autonomous generation reliability varies across tasks. GPT-6 Astra succeeds in 8/10 Stack Cups trials, while Fable 5 succeeds in 6/10 Claw Machine trials. Precise contact remains challenging: on Close a Lid, Fable 5 succeeds in 0/10 trials and Astra in 2/10. The low success rates on Fold a Cloth and Close a Lid indicate that successful demonstrations alone do not reliably support generation across all four tasks.

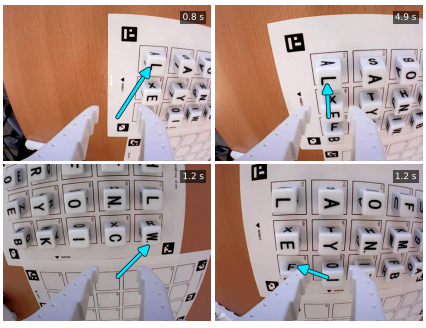


Figure 2: Corrective segments for pickup. Arrows illustrate adjustment directions; they are not recorded actions or model inputs.

Improving Precision through Corrective Demonstration Segments. Minor spatial misalignments during precise manipulation can lead to missed grasps or failed insertions. We augment the in-context examples with **corrective demonstration segments**: short demonstrations focused on moments when execution requires a corrective adjustment. These examples show how to use visual feedback to correct physical errors, providing guidance beyond successful task executions. Figure 2 illustrates

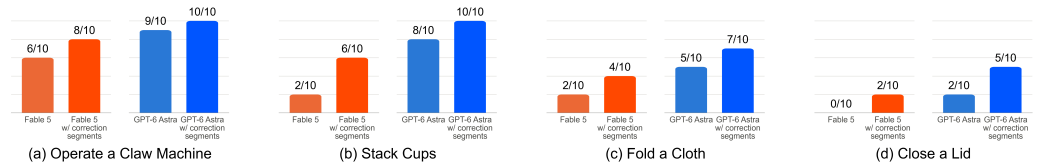


Figure 3: Effect of adding corrective demonstration segments to the in-context examples. Success counts compare the baseline and augmented conditions for Fable 5 and GPT-6 Astra across four manipulation tasks, with ten trials per condition.

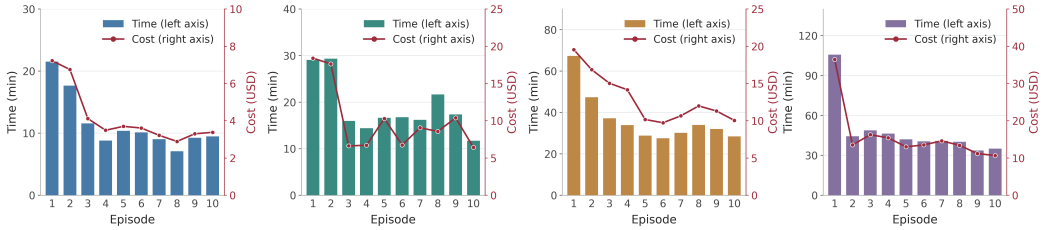


Figure 4: Generation efficiency as successful demonstrations accumulate in context. Bars show time in minutes (left axis), and lines show API cost in USD (right axis) for GPT-6 Astra over ten successful episodes per task. Episode indices count successful rollouts; time and API cost from failed attempts are excluded.

corrective alignment during cube pickup; Appendix G reports per-task segment counts and additional examples (Figure 9).

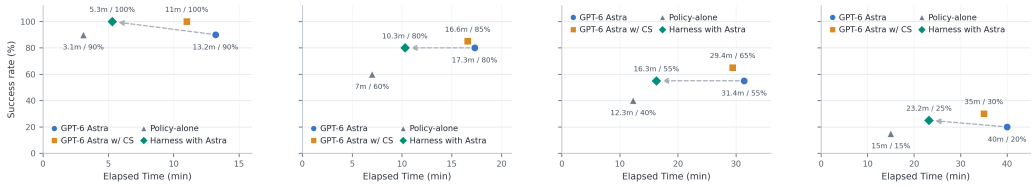
Corrective demonstration segments improve observed success across the evaluated tasks (Figure 3). On Close a Lid, Astra improves from 2/10 to 5/10 successful trials, while Fable 5 improves from 0/10 to 2/10. Astra also succeeds in all ten Stack Cups trials. These results suggest that examples of visual corrections help recover attempts that would otherwise fail, although precise contact remains challenging.

3.2 Data Synthesis via Autonomous Generation

We use GPT-6 Astra to collect successful manipulation trajectories for local-policy training. During synthetic trajectory collection, frontier-controlled episodes can take tens of minutes, with some exceeding 100 minutes. These data-collection runs are distinct from the one-hour capability trials in Section 3.1. We therefore examine whether accumulating successful examples in context reduces generation time and API cost.

Generation Efficiency with Accumulated Examples. We track time and API cost across ten successful episodes per task. Unlike the independent trials with five demonstrations in Section 3.1, these generation sessions retain context across episodes. Useful segments from successful trajectories are added to a separate pool while the original five human demonstrations are retained. The model reuses and adapts these segments when the current situation matches, updating its strategy after each episode. Section G.2 provides the additional instruction used for this experiment. Figure 4 reports time and API cost per successful episode; time and API cost from failed attempts are excluded. These measurements describe successful-rollout efficiency rather than the total cost of collecting the dataset.

Generation time and API cost decrease as successful examples accumulate (Figure 4). After the initial episodes, Claw Machine generation takes approximately 8–12 minutes and $3–4 per trajectory. Fold a Cloth remains more costly, with later episodes taking approximately 28–34 minutes and $10–12 per trajectory. On Stack Cups, API cost drops by more than 50% after the second successful episode. These observations indicate improved generation efficiency as context accumulates.



(a) Operate a Claw Machine (b) Stack Cups (c) Fold a Cloth (d) Close a Lid

Figure 5: Average elapsed task time and task success across four control configurations: GPT-6 Astra alone, Astra with corrective demonstration segments (CS), the local policy alone, and the harness with Astra. Each panel reports one manipulation task, with $n = 20$ trials per configuration and task. Time is averaged over successful trials only and includes reasoning, planning, tool calls, and API latency, excluding hardware-fix intervals. Success rate uses all 20 trials.

3.3 Harnessing Frontier Models for Local Policies

We incorporate autonomously generated trajectories into local-policy training alongside human demonstrations. For each task, the training corpus contains 40 successful episodes: 20 human-teleoperated demonstrations and 20 trajectories generated by GPT-6 Astra. Appendix G.6 compares this mixed corpus with 40 human demonstrations per task; both achieve 41/80 policy-alone successes across the four tasks (Figure 10). The autonomous trajectories include corrective movements, exposing the local policy to error-recovery behavior during training.

At deployment, the frontier model receives frames from three camera views, the task description, and the goal condition, and provides high-level instructions to the local policy. The local controller is an OpenPI $\pi_{0.5}$ policy (Black et al., 2025) that predicts ten-action chunks of absolute joint targets; its interface and timing limitations are detailed in Section G.4.

To evaluate the integrated harness during deployment, we compare four control configurations: GPT-6 Astra alone, Astra with corrective demonstration segments (CS), the local policy alone (Policy-alone), and the harness with Astra. We evaluate 20 trials per configuration and task, reporting success over all trials and mean elapsed time over successful trials only. Time includes frontier reasoning, planning, tool calls, API latency, and corrective retries, but excludes hardware-fix intervals (Section G.5).

Figure 5 compares execution time and success rate across the four control configurations. Policy-alone execution is fast but achieves an aggregate success rate of 51.25% (41/80 trials) across the four tasks. The harness combines shorter execution times than frontier-only control with higher task success than policy-alone execution, while precise contact remains challenging. These results support delegating motor execution to a local policy under frontier-model guidance. The effectiveness of this separation, however, depends on whether the local policy can follow the planner's instructions, motivating the language-supervision study in Section 4.

4 Bridge 2: Expanding the Policy Language Channel

Within the harness, the frontier model provides high-level instructions while the local policy executes motor actions. The effectiveness of this division depends on whether the policy responds to the intended instruction granularity and wording. We address these requirements through supervision along two axes: **1) Temporal resolution**: Task-level or subtask-level descriptions provide little supervision for individual movements. **2) Linguistic diversity**: A single templated description per segment does not capture the varied instructions produced by a frontier planner. We introduce dense language supervision by annotating the same demonstration episodes at three nested granularities—primitive, atomic, and composite—and pairing each level with multiple instruction formats and rephrasings.

Table 1: Illustrative language supervision for `MakeFruitBowl`, with an apple and a banana in the cabinet and a bowl on the counter. (a) The complete task and subtask sequence, with primitives shown only for apple transfer. (b) Four descriptions of that same apple-transfer subtask. Object names and kinematic descriptions must match the demonstrated episode.

(a) Temporal granularity	
Composite	Open the cabinet. Place the apple and banana from the cabinet into the bowl. Then close the cabinet.
Atomic	Open the cabinet → **transfer the apple into the bowl** → transfer the banana into the bowl → close the cabinet.
Primitive (apple transfer only)	Reach for the apple → grasp the apple → lift the apple → carry it over the bowl → lower the apple → release it into the bowl → retract the arm.
(b) Instruction formats for apple transfer	
Arm/gripper kinematics	Grasp the apple, lift it out of the cabinet, carry it over the bowl, and release it into the bowl.
Spatial destinations	Move the apple from inside the cabinet into the bowl on the counter.
Terse command	Put the apple in the bowl.
Natural phrasing	Take the apple out of the cabinet and pop it into the bowl.

4.1 A Corpus at Three Hierarchical Granularities

We annotate the same demonstration episodes at three nested granularities: (1) a **composite** annotation names the *full multi-stage* task; (2) an **atomic** annotation names one *subtask* within it; and (3) a **primitive** annotation names a *single-skill segment* within a subtask, such as a reach, a grasp, or a carry. The three layers describe nested temporal spans within the same episodes. Table 1(a) illustrates the hierarchy for `MakeFruitBowl`, retaining the complete task and subtask sequence and expanding the apple-transfer subtask into primitive instructions. The other subtasks have their own primitive annotations, omitted from the example for space. The annotations describe the same underlying trajectories at different temporal resolutions. This provides a vocabulary for individual movements as well as complete subtasks.

RoboCasa 365 provides both atomic and composite annotations; we additionally construct primitive-level annotations. We derive primitive segments from proprioceptive signals, using finger travel and end-effector motion to identify physical events, with short boundary buffers and fallback windows. This yields eight primitive kinds over 27,113 segments, drawn from 4,100 episodes across 91 tasks on RoboCasa 365. For BEHAVIOR-1K, we use the primitive annotations and corresponding segments provided by the benchmark.

4.2 Dense Multi-aspect Language Supervision

We adapt the multi-aspect supervision approach from DeMiAn Kim et al. (2026) to provide four instruction formats for each segment: arm/gripper kinematics, spatial destinations, a terse command, and natural phrasing. These formats emphasize how the robot moves, where objects move, minimal wording, and conversational wording, respectively. Table 1(b) illustrates all four for the same apple-transfer subtask. The formats can convey different information about the same action segment; they are not simply paraphrases. Appendix B reports corpus construction and physical consistency checks, and Section B.2 describes VLM screening and human review of object-identity annotations.

4.3 Experimental Setup

We evaluate our architecture on two simulated benchmarks, RoboCasa 365 (Nasiriany et al., 2026; RoboCasa, 2026) and BEHAVIOR-1K (Li et al., 2023). We choose simulation-based benchmarks that provide long-horizon tasks requiring robust high-level planning and enable controlled comparisons of language inputs. This allows us to vary instruction granularity and phrasing while keeping initial conditions and evaluation criteria consistent. 1) **RoboCasa 365**: We evaluate ten composite kitchen tasks, scored via strict success rate over 240 paired episodes per seed (24 episodes per task). 2) **BEHAVIOR-1K**: We evaluate eight repeated rollouts of one instance for each of seven tasks (56

episodes per condition and seed). Due to the extreme difficulty of this benchmark, where complete task successes approach zero, we report mean subgoal progress (mean q), the benchmark's predicate-based partial-success score, rather than binary success rates. Here, subgoals refer to benchmark goal predicates, not planner-generated instruction steps. To ensure fair comparison, all policies are initialized from the same pretrained checkpoint and trained for a matched budget of 120,000 steps, with training and evaluation seeds 42 and 142 for the main language-supervision comparison. Table 8 in Appendix C defines the prompt content for each condition. Benchmark tasks and training configuration details are provided in Appendix C.

Instructor conditions. We evaluate each language supervision recipe under two instructor conditions. Under the **oracle** instructor, the subgoal plan is authored and instruction switching follows simulator-state milestones, providing a privileged reference for policy performance. Under **Fable 5**, the frontier model writes the full subgoal sequence once at the start of the episode from the composite goal text and reset-frame camera views, without access to simulator state. For each evaluation episode, this sequence is generated once and reused across all policy arms, holding instruction content fixed while the policy interface varies. A separate local open-weights VLM, Qwen3.6-27B, monitors execution from dynamic observations and decides when the current subgoal is complete. The harness then advances a cursor through the fixed sequence and supplies the local policy (S1) with one instruction at a time. Appendix D details this protocol and its connection to Probe 2.

4.4 Results

Table 2 shows that subtask instructions improve performance over fixed task-level instructions. Adding task-goal context improves both benchmarks. Hierarchical supervision further improves RoboCasa success over Task + Subtask, but not BEHAVIOR-1K progress; multi-aspect supervision alone improves both benchmarks. The combined recipe performs best under both instructors, reaching oracle scores of 0.517 and 0.174 on RoboCasa and BEHAVIOR-1K, respectively. Under Fable 5, it improves the corresponding scores from 0.381 to 0.485 and from 0.084 to 0.143 over subtask-level supervision.

Table 2: Effect of language supervision on long-horizon task performance under oracle and frontier-model instructors. Values average seeds 42 and 142: success rate on RoboCasa 365 and mean subgoal progress on BEHAVIOR-1K. Table 9 gives both seed results.

	RoboCasa 365		**BEHAVIOR-1K**	
Language Supervision	Oracle	Fable 5	Oracle	Fable 5
Task-level (fixed)	0.169		0.076	
Subtask-level	0.406	0.381	0.143	0.084
Task + Subtask	0.417	0.402	0.154	0.091
Hierarchical only	0.456	0.440	0.143	0.084
Multi-aspect only	0.425	0.406	0.160	0.103
Hierarchical + Multi-aspect	**0.517**	**0.485**	**0.174**	**0.143**

4.4.1 Probe 1: Multi-Granularity Instruction Following

We evaluate single-seed subtask-level and hierarchical-only checkpoints from the main evaluation on six primitive movement commands across 12 RoboCasa scenes, without additional training. Compliance measures final displacement along the commanded axis against predefined thresholds; Appendix E provides the protocol and results (Table 11). Hierarchical supervision improves compliance across all six commands, including "move the arm down" (0.111 to 0.778) and "move the arm back" (0.028 to 0.444). This probe measures directional movement, not successful manipulation or online correction during complete tasks.

4.4.2 Probe 2: Robustness to Instruction Diversity

A robust local policy must understand the semantic intent of an instruction regardless of the specific syntax or vocabulary used by the planner. We test two forms of instruction variation: changing the source planner among Fable 5, Opus 5, and Gemini-3.1-Pro, and rephrasing the resulting instructions with Fable 5. In the rephrased condition, we prompt Fable 5 to preserve the original task goal while changing how the instructions are expressed. For example, "Open up the cabinet" becomes "Grasp

Table 3: Single-seed (seed 42) execution success rates on RoboCasa 365 by source planner. Rephrased instructions are generated by Fable 5, preserving the task goal while changing the wording. Parentheses report changes between the displayed original and rephrased rates.

Language Supervision	Opus 5		Gemini-3.1-Pro		Fable 5	
	Original	Rephrased	Original	Rephrased	Original	Rephrased
Subtask-level	.346	.333 (-.013)	.325	.329 (+.004)	.392	.350 (-.042)
Hierarchical only	.413	.408 (-.005)	.400	.388 (-.012)	.446	.400 (-.046)
Multi-aspect only	.371	.363 (-.008)	.350	.350 (.000)	.417	.408 (-.009)
Hierarchical + Multi-aspect	**.488**	**.471 (-.017)**	**.467**	**.454 (-.013)**	**.492**	**.483 (-.009)**

the cabinet door handles and swing the doors open." This condition tests goal-preserving variation in wording and descriptive detail; Fable 5 serves as the rephrasing model for the instructions from all three source planners. We first evaluate the quality of the complete high-level plans generated once at the start of each episode by these three frontier models, using the composite goal text and reset-frame camera views. We compare their predicted plans across 30 scenes against the ground-truth oracle plans. Within each planner and phrasing condition, the generated plan for an evaluation episode is reused across all policy arms; observation-based completion monitoring and cursor advancement follow the shared protocol in Appendix D.

Table 10 in Appendix D reports decomposition step-count match and semantic similarity, showing that all three models generate plans highly similar to the oracle reference. These text-based metrics provide a check on plan quality, but do not establish that the plans are functionally equivalent. Execution differences across planners can therefore reflect both wording and differences in plan content or ordering. Table 3 reports single-seed end-to-end execution success across the three planners and their rephrased instruction conditions.

For the subtask-level baseline, changing the planner from Fable 5 to Gemini-3.1-Pro reduces success from 0.392 to 0.325, while using the rephrased versions of Fable 5's instructions reduces success from 0.392 to 0.350. Multi-aspect supervision reduces the Fable 5 rephrasing drop to 0.009, although it does not produce the smallest drop for every planner. Under Opus 5, hierarchical-only supervision has a smaller drop (0.005) than multi-aspect-only supervision (0.008). The joint recipe achieves the highest absolute success in all six conditions, ranging from 0.454 to 0.492, with rephrasing drops of 0.009–0.017. Together, these results support improved execution across the tested planners and phrasings when hierarchical and multi-aspect supervision are combined.

5 Bringing Both Bridges Together

We evaluate both bridges together on the crossword task, which requires forming valid words and placing alphabet cubes on a shared board. Each placement changes the board and remaining letter pool. In our qualitative observations, both frontier models proposed valid words, while physical execution was the main bottleneck.

Setup. The workspace contains an initially empty 6×6 grid and a randomized pool of 15 alphabet cubes. We use GPT-6 Astra to autonomously generate a domain-specific demonstration corpus and annotate the trajectories with dense, multi-aspect language supervision. The two harness configurations use the same demonstrations, policy architecture, training budget, and frontier controller; only language supervision differs: subtask-level instructions versus hierarchical + multi-aspect supervision. During harness deployment, the frontier model reads camera views at times of its choosing and issues language instructions with execution durations; the local policy generates motor actions. Subsequent observations support corrective instructions after failed pickups and revision of planned placements. Unlike the fixed-plan evaluation in Section 4, instructions can be revised during execution. Appendix H provides the policy training configuration and a separate 20-trial pick-and-place evaluation (Table 14). We evaluate performance using two reward components: $+2$ points for each cube successfully placed at its instructed grid coordinate and $+2$ points for each valid word formed.

Results. We report the mean total score over five games, each using a different alphabet set and a one-hour execution budget, for four control configurations: Fable 5 alone, GPT-6 Astra alone, the

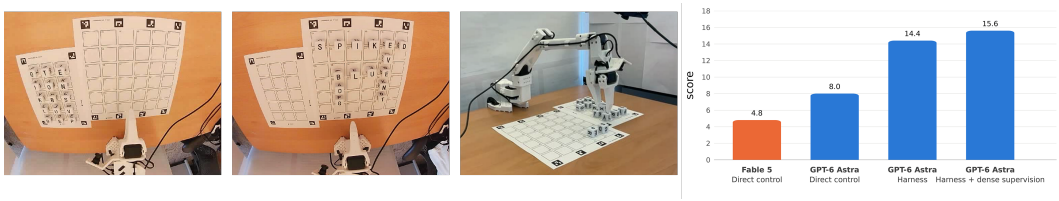


Figure 6: The crossword task: example board states and the robot workspace (left), and mean total score after one hour across five games for four control configurations (right). Per-game scores are reported in Table 13.

GPT-6 Astra harness, and the GPT-6 Astra harness with dense language supervision. Appendix H provides per-game results and an additional comparison prohibiting cell IDs in instructions (Table 13). We omit policy-alone execution because the task requires high-level word planning. As shown in Figure 6, both harness configurations achieve higher scores than either frontier model alone within the fixed time budget. The dense-supervision variant further improves the mean score. Qualitatively, we observed the local policy responding to corrective instructions after failed pickups, but did not separately measure recovery success.

Execution timing. The harness delegates motor execution to a low-latency local policy, but the overall system still pauses for frontier reasoning between execution segments. Figure 7 shows the first 20 minutes. Over the full one-hour runs (Figure 11), cumulative action execution occupies 2 min 40 s under direct control and 9 min 8.55 s with the harness. These are individual runs, not average timing results; more time executing actions does not establish continuous real-time control. Appendix H gives timing details.

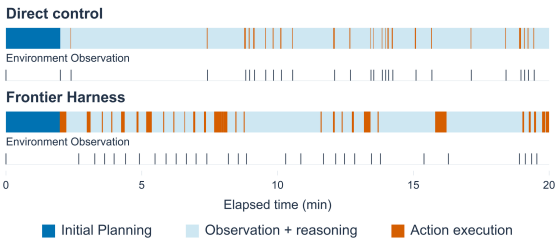


Figure 7: First 20 minutes of GPT-6 Astra crossword execution, one run per configuration. Ticks mark model-requested camera observations. Full 60-minute timelines appear in Figure 11.

6 Conclusion

We study two complementary bridges between frontier reasoning and robot execution: autonomous demonstration generation to supplement local-policy training, and dense language supervision to improve instruction following. Corrective demonstration segments improve generation reliability. Generation time and cost decrease across the observed successful episodes as examples accumulate. Hierarchical and multi-aspect language supervision improves instruction following on long-horizon simulation tasks. Evaluation on the crossword task brings both bridges together, with the harness achieving higher scores than direct frontier control within a fixed time budget. These results suggest that effective delegation requires both capable local execution and a language interface that supports the frontier model's guidance. The overall system remains intermittent because frontier reasoning introduces pauses between execution segments. Extending this approach to broader physical tasks and reducing dependence on human demonstrations remain directions for future work.

AI Use Statement

AI models are used as experimental components for autonomous robot demonstration generation, language annotation and rephrasing, high-level instruction generation, progress monitoring, and semantic-similarity assessment, as described in the methods and appendices. We also used AI assistants to assist with literature searches and LaTeX editing, figure-preparation code, and experimental

methodology. Annotation checks and the limitations of model-based assessments are described in Appendix B and Appendix A.

Reproducibility Statement

We document annotation sources, primitive segmentation, and annotation-quality checks in Appendix B; training/evaluation splits, policy configurations, language supervision recipes, and per-seed results in Appendix C; and frontier planning and execution monitoring in Appendix D. Appendix E specifies the primitive-compliance protocol. Appendix G provides the autonomous-generation prompts, corrective-segment counts, physical deployment interface, timing definitions, and demonstration-corpus comparison. Appendix H provides crossword training settings, scoring details, per-game results, and execution timelines. Remaining documentation gaps and limits on reproducibility are stated in Appendix A.

References

Suneel Belkhale, Tianli Ding, Ted Xiao, Pierre Sermanet, Quan Vuong, Jonathan Tompson, Yevgen Chebotar, Debidatta Dwibedi, and Dorsa Sadigh. RT-H: Action hierarchies using language. In *Proceedings of Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.049. URL https://www.roboticsproceedings.org/rss20/p049.html.

Shmuel Berman, Michael Ilie, Jia Deng, and Daniel Freeman. Claude plays robotics. https://www.anthropic.com/research/claude-plays-robotics, July 2026. Anthropic Frontier Red Team research post, July 9, 2026.

Kevin Black, Mitsuhiko Nakamoto, Pranav Atreya, Homer Walke, Chelsea Finn, Aviral Kumar, and Sergey Levine. Zero-shot robotic manipulation with pretrained image-editing diffusion models. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.10639.

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 *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 2025. URL https://proceedings.mlr.press/v305/black25a.html.

Siyi Chen, Hugo Hadfield, Alex Zook, Mikaela Angelina Uy, Chan Hee Song, Erwin Coumans, Xuning Yang, Faisal Ladhak, Qing Qu, Stan Birchfield, Jonathan Tremblay, and Valts Blukis. VoLo: A physical orchestrator for open-vocabulary long-horizon manipulation. *arXiv preprint arXiv:2606.07723*, 2026a.

Yanzhe Chen, Zechen Bai, Zhijun Cao, Wenzheng Zeng, Kevin Qinghong Lin, Yiqi Lin, Guoqiang Liang, Kevin Yuchen Ma, Qiming Huang, and Mike Zheng Shou. Show-Harness: Just a VLM agent can play robots. *arXiv preprint arXiv:2609.10522*, 2026b. URL https://arxiv.org/abs/2609.10522.

An-Chieh Cheng, Yandong Ji, Zhaojing Yang, Zaitian Gongye, Xueyan Zou, Jan Kautz, Erdem Bıyık, Hongxu Yin, Sifei Liu, and Xiaolong Wang. NaVILA: Legged robot vision-language-action model for navigation. In *Proceedings of Robotics: Science and Systems*, 2025. doi: 10.15607/RSS.2025.XXI.018. URL https://www.roboticsproceedings.org/rss21/p018.html.

Jiafei Duan, Wentao Yuan, Wilbert Pumacay, Yi Ru Wang, Kiana Ehsani, Dieter Fox, and Ranjay Krishna. Manipulate-Anything: Automating real-world robots using vision-language models. *arXiv preprint arXiv:2406.18915*, 2024. URL https://arxiv.org/abs/2406.18915.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, Guanya Shi, Jiajun Wu, Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi Fan. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2603.22435.

Jiaheng Hu, Mohit Shridhar, Caden Lu, Dhruv Shah, Hao-Tien Lewis Chiang, Jie Tan, and Annie Xie. What matters in orchestrating robot policies: A systematic study of hierarchical VLA agents. *arXiv preprint arXiv:2606.10267*, 2026.

Bosung Kim, Ruiyi Wang, David Acuna, Jaehun Jung, Alexander Trevithick, Brandon Cui, Yejin Choi, and Prithviraj Ammanabrolu. How to instruct your robot: Dense language annotations power robot policy learning. *arXiv preprint arXiv:2605.17077*, 2026. doi: 10.48550/arXiv.2605.17077. URL https://arxiv.org/abs/2605.17077.

Seungjae Lee, Yoonkyo Jung, Jusuk Lee, Jonghun Shin, Amir Hossein Shahidzadeh, Yao-Chih Lee, H. Jin Kim, Jia-Bin Huang, and Furong Huang. μ_0: A scalable 3D interaction-trace world model. *arXiv preprint arXiv:2606.13769*, 2026.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, Mona Anvari, Minjune Hwang, Manasi Sharma, Arman Aydin, Dhruva Bansal, Samuel Hunter, Kyu-Young Kim, Alan Lou, Caleb R Matthews, Ivan Villa-Renteria, Jerry Huayang Tang, Claire Tang, Fei Xia, Silvio Savarese, Hyowon Gweon, Karen Liu, Jiajun Wu, and Li Fei-Fei. BEHAVIOR-1K: A benchmark for embodied AI with 1,000 everyday activities and realistic simulation. In *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 80–93. PMLR, 2023. URL https://proceedings.mlr.press/v205/li23a.html.

Yi Li, Yuquan Deng, Jesse Zhang, Joel Jang, Marius Memmel, Raymond Yu, Caelan Reed Garrett, Fabio Ramos, Dieter Fox, Anqi Li, Abhishek Gupta, and Ankit Goyal. HAMSTER: Hierarchical action models for open-world robot manipulation. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://hamster-robot.github.io/.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022.

Runyu Lu, Yubo Wu, Ethan Kou, Letian Fu, Wenli Xiao, Ajay Mandlekar, Yinzhen Xu, Guanya Shi, Ken Goldberg, Ang Chen, Mosharaf Chowdhury, Yuke Zhu, Linxi Fan, and Guanzhi Wang. ASPIRE: Agentic skills discovery for robotics. *arXiv preprint arXiv:2607.00272*, 2026.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. MimicGen: A data generation system for scalable robot learning using human demonstrations. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 1820–1864. PMLR, 2023. URL https://proceedings.mlr.press/v229/mandlekar23a.html.

Achu Menon, Sravanthi Machcha, Sabrina Zou, Tzu Kit Chan, and Jay Chooi. GPT-6 Astra on robotic manipulation. https://openai.robocurve.org/gpt-6-astra/, September 2026. Robocurve, September 4, 2026. Accessed 2026-09-18.

Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *International Conference on Learning Representations (ICLR)*, 2026.

NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

RoboCasa. RoboCasa365. https://robocasa.ai/, 2026. Benchmark v1.0 released 2026-02-18; per-frame subtask annotations released 2026-07-07.

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 *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 54919–54933. PMLR, 2025. URL https://proceedings.mlr.press/v267/shi25d.html.

Mohit Shridhar, Yat Long Lo, and Stephen James. Generative image as action models. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2429–2455. PMLR, 2025. URL https://proceedings.mlr.press/v270/shridhar25a.html.

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.

Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian Fu, Haoru Xue, Jalen Lu, Yi Yang, Cunxi Dai, Zi Wang, Jimmy Wu, Guanzhi Wang, S. Shankar Sastry, Ken Goldberg, Linxi Fan, Yuke Zhu, and Guanya Shi. ENPIRE: Agentic robot policy self-improvement in the real world. *arXiv preprint arXiv:2606.19980*, 2026.

A Limitations and Future Work

Experimental scope. The main language-supervision comparison uses two training/evaluation seed pairs, providing limited replication without separating training and evaluation variability. Other experiments do not quantify variation across independent training runs. Evaluation tasks are represented in training: RoboCasa 365 tests held-out layouts, styles, and sampled assets, whereas BEHAVIOR-1K uses one held-out initial configuration per task within existing scenes. The latter does not establish generalization across unseen houses or layouts. Physical evaluations cover four manipulation tasks and five crossword games; broader task coverage and additional independent runs are needed to assess generality.

Autonomous demonstration collection. Policies trained on 20 human and 20 generated demonstrations per task attain the same aggregate success as those trained on 40 human demonstrations (Appendix G.6). This result does not establish statistical equivalence or isolate the incremental benefit over 20 human demonstrations alone, which we do not evaluate. We are collecting additional autonomous demonstrations to evaluate policies trained entirely on generated trajectories.

Timing and deployment. Frontier reasoning introduces pauses between execution segments, so low-latency local control does not make the complete system real-time. The crossword timelines illustrate one run per configuration, not an average speedup. Generation-efficiency measurements exclude failed attempts and describe observational trends across successful episodes, rather than total collection cost or a causal effect of accumulating examples. Physical deployment times are conditional on success and exclude hardware-fix intervals. Achieved control rates and instruction-switching behavior remain incompletely characterized (Appendix G). More responsive coordination between frontier reasoning and local execution is an important direction for future work.

Annotations and reproducibility. Human review covers object identity in 50 sampled segments, rather than all aspects of annotation quality. The exact executed annotation-generation prompt, per-segment phrasing counts, and complete validation/fallback settings are not fully documented; denominators for some physical-validation checks are also unavailable. These gaps limit exact reproduction of the annotation pipeline and motivate more complete documentation and broader annotation audits.

B Corpus Construction and Annotation Audit

B.1 Corpus Construction and Physical Validation

Annotation sources and segmentation. Both RoboCasa 365 and BEHAVIOR-1K provide atomic and composite annotations. BEHAVIOR-1K also provides primitive annotations and their corresponding segments, which we use directly. The primitive segmentation procedure described below applies to RoboCasa 365. For RoboCasa, we additionally construct a primitive corpus containing 27,113 segments from 4,100 episodes across 91 tasks. This is the corpus used for the reported policy training. Segmentation uses recorded action and state streams rather than video or language: finger travel identifies grasp and release, smoothed vertical and horizontal end-effector motion distinguishes lift, lower, and transport, and direction reversals identify shaking. The median segment length is 18 frames, with an interquartile range of 13–35 frames. Table 4 gives the composition. **Segmentation parameters.** We smooth the end-effector trajectory with a five-frame moving average. Motion below 1.5 mm per frame is treated as stationary; otherwise a vertical-to-total motion ratio of at least 0.5 assigns an upward or downward label, with remaining motion assigned to transport. Carry segments require at least eight frames, net displacement of at least 2 cm, and a dominant-axis displacement ratio of at least 0.70. A finger gap below 7 cm identifies candidate held spans. Carry boundaries are trimmed to exclude grasp/release intervals and to limit within-segment finger-gap variation to 3 mm.

Grasp and release windows follow finger travel using a 1 mm per-frame change threshold and a 4 mm net-change check. When that check fails, the implementation falls back to a three-frame window on either side of the event. Windows include two lead-in frames, with trailing buffers of four frames for grasp and eight for release. Reach and retract are bounded by end-effector motion and capped at 80 and 60 frames, respectively; retract begins after release. Shaking merges at least two carry runs when the dominant-axis motion reverses at least three times, the path length is at least 10 cm, net displacement is at most 15 cm, and the net-to-path ratio is at most 0.45.

Table 4: Composition of the RoboCasa primitive annotation corpus.

Primitive	Segments	Physical interpretation
Transport	6,256	Held object translates approximately horizontally.
Grasp	4,086	Gripper closes on an object.
Reach	4,066	Empty arm approaches an object with the gripper open.
Release	3,408	Gripper opens to release an object.
Lower	3,356	Held object descends.
Retract	3,057	Empty arm withdraws after release.
Lift	2,859	Held object rises.
Shake	25	Held object oscillates without sustained transport.
Total	27,113	

Object identity. Object labels are derived from recorded simulator state rather than from nouns in task descriptions. Table 5 summarizes their provenance. For phases with little object displacement, identity can be inherited from transport within the same grasp-to-release span. A consistency check against a directional inheritance rule found one disagreement in 10,314 cases across 222 multi-pick episodes. Unresolved cases retain the decomposition's object name and are explicitly marked; these are concentrated in fixture manipulation, where the relevant part lacks a free joint. Asset mesh categories provide 112 distinct object names.

Language generation. We use Opus 5 for annotation generation. The language supervision uses the four formats described in Section 4: arm/gripper kinematics, spatial destinations, terse commands, and natural phrasing, with additional rephrasings. Captions are conditioned on the segment's primitive kind and object identity and retained only after validation.

Physical consistency. Table 6 reports checks against the motion signals associated with each primitive. Among 230 sampled transport segments, 96.5% contain a single moving object. In eight inspected cases with multiple moving objects, the annotation agrees with the object rigidly held by the gripper. The visual audit is described in Section B.2.

Table 5: Provenance of primitive-segment object identities. Simulator-derived coverage includes inherited identities and is not an annotation-accuracy estimate.

Source	Segments	Share
Object motion within the segment	14,935	55.1%
Inheritance within the grasp-to-release span	10,236	37.8%
Expanded temporal window	1,147	4.2%
Unresolved; decomposition label retained	795	2.9%
Simulator-derived total	26,318	97.1%

Table 6: Physical consistency checks for primitive annotations. These checks measure agreement with motion signals, not downstream task success.

Check	Result
Transport with more than 6 mm of gripper travel	0.0%; median 0.7 mm
Grasp containing gripper closing	96.9%
Release containing gripper opening	93.3%
Retract starting with gripper at least 80% open	74.4%; median openness 0.87
Retract overlapping release	0 / 3,057

Limitations. Object identity inherited across phases is inferred rather than directly observed within each segment, and 2.9% of identities remain unresolved by simulator-based checks. Shaking is rare in the corpus and its translation-reversal detector can miss tilting motions. Visual inspection covers only a sample. These measurements characterize annotation consistency; they do not isolate the causal effect of annotation quality on task success.

B.2 Annotation Audit

VLM screening. We use Opus 5 to inspect manipulated-object identity from RoboCasa's default agent-left, agent-right, and wrist-camera views. The model receives segment imagery without the object label, task name, or annotation slots, and its predicted identity is compared with the simulator-derived annotation. The sample contains 207 segments across 61 tasks, selected round-robin across tasks within each primitive kind: 26 segments for each of seven kinds and all 25 available shake segments. This screening identifies cases for inspection rather than estimating annotation accuracy: the VLM can itself misidentify objects. Each segment is represented by 12 frames.

Human review. The authors manually reviewed 50 randomly sampled segments and agreed with the annotated object identity in 48 cases (96%). This result describes agreement within the reviewed sample, rather than a comprehensive assessment of annotation accuracy across the corpus. Physical consistency checks for primitive labels are reported in Section B.1.

C Evaluation Protocol and Cell Provenance

Training/evaluation split. Evaluation tasks are included in the training task sets for both benchmarks. For RoboCasa 365, training uses pretrain demonstrations, whereas evaluation uses the target environment split with held-out kitchen layouts, visual styles, and sampled object assets. For BEHAVIOR-1K, the seven evaluation tasks belong to the ten-task training subset; evaluation uses a held-out public-test initial configuration, instance 301, for each task. This tests new object and robot initial configurations within the existing scene setup. The seven-task results use repeated rollouts of this single configuration per task and do not establish generalization across unseen houses or layouts.

RoboCasa evaluation. The evaluation set contains ten composite tasks selected to test multi-step execution and instruction switching across increasing task depth (Table 7). It includes cabinet and drawer articulation, repeated object transfers, and rinsing, with three tasks containing two subgoals, three containing three subgoals, and four containing four subgoals. Training demonstrations and milestone checkers are available for all ten tasks, and each task has a complete instruction sequence

including required closing actions. Subgoal counts refer to distinct instruction phases in the execution plan, rather than the number of evaluation predicates. For the main language-supervision comparison, each condition evaluates 24 episodes per task and seed, giving 240 episodes per seed and 480 across seeds 42 and 142, with a 6,000-step limit. Performance is measured by strict task success, including required terminal conditions such as closing a cabinet; intermediate milestone completion alone is insufficient.

Table 7: RoboCasa 365 evaluation tasks by execution-plan depth. Subgoals count distinct instruction phases, not evaluation predicates.

Task	Subgoals
BreadAndCheese	2
ThawInSink	2
StartElectricKettle	2
ClearCuttingBoard	3
GatherCuttingTools	3
RinseBowls	3
CerealAndBowl	4
MakeFruitBowl	4
RestockBowls	4
PlaceFoodInBowls	4

Instructor conditions. Oracle evaluation uses authored subgoal instructions and simulator-state milestones for switching. Frontier-instructor evaluation separates a fixed initial plan from observation-based completion monitoring, as described in Appendix D. The planner identity and the policy checkpoint must be matched to the recorded run when interpreting a table cell.

BEHAVIOR-1K evaluation. For the main language-supervision comparison, we evaluate each condition for eight episodes on each of seven tasks per seed, totaling 56 episodes per seed and 112 across seeds 42 and 142. The seven tasks are Chop an onion, Hiding Easter eggs, Preparing lunch box, Setting mousetraps, Chopping wood, Sorting vegetables, and Putting away Halloween decorations. All rollouts use task instance 301, with each rollout executed in a separate process. The execution limit is 12,000 simulator steps per episode. We compute the mean terminal partial-credit score q within each task, then report the unweighted mean of these seven task means. These are repeated rollouts of one instance per task, not eight distinct evaluation instances; they therefore do not establish generalization across a range of held-out task instances. The score measures benchmark goal-predicate satisfaction, not progress through planner-generated steps.

Training configuration. The policies in the simulation benchmark comparisons are trained for 120,000 steps. Crossword policy training uses a separate configuration described in Appendix H. The main language-supervision comparison uses two independently trained policies per condition, with training seeds 42 and 142; each is evaluated using the corresponding evaluation seed. The RoboCasa OpenPI configurations use a $\pi_{0.5}$ policy with an action horizon of ten, a maximum text length of 384 tokens, and batch size 256. Optimization uses AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, weight decay 10^{-10}, and gradient-norm clipping at 1.0. The learning rate warms up over 1,000 steps to 5×10^{-5} and remains at that value; exponential moving average decay is 0.999. Because training and evaluation seeds change together, differences between the two runs reflect both sources of variation rather than isolating training-seed variability.

The BEHAVIOR mixed-policy configuration uses $\pi_{0.5}$ with an action horizon of 32, a maximum text length of 256 tokens, and batch size 64. Its learning-rate schedule warms up over 1,000 steps to 2.5×10^{-5} and decays to 2.5×10^{-6}.

Language supervision recipes. Table 8 specifies the prompt construction for each condition. Here, `goal` is the episode-level task instruction and `subgoal` is the current subtask instruction. In the hierarchical conditions, `hierarchical instruction` denotes the instruction corresponding to the sampled segment's hierarchy level, including primitive instructions such as "lift the arm". The prompt pairs this instruction with the original episode-level goal using the fixed connective "Current step:". Multi-aspect supervision samples from the original segment instruction and descriptions in the four formats introduced in Section 4: arm/gripper kinematics, spatial destinations, terse commands, and natural phrasing.

Table 8: Prompt construction for the six language supervision conditions. Both hierarchical conditions pair the episode-level goal with the instruction corresponding to the sampled segment's hierarchy level, using "Current step:" as a fixed connective.

Condition	Annotation levels in prompt	Training format
Task-level	Episode-level goal only	`{goal}`
Subtask-level	Current subgoal	`{subgoal}`
Task + Subtask	Episode-level goal + current subgoal (no primitive supervision)	`{goal}. {subgoal}`
Hierarchical only	Episode-level goal + corresponding hierarchy instruction (e.g., primitive: "lift the arm")	`{goal}. Current step: {hierarchical instruction}`
Multi-aspect only	Current subgoal	`{sampled subgoal}`
Hierarchical + Multi-aspect	Episode-level goal + corresponding hierarchy instruction	`{goal}. Current step: {sampled hierarchical instruction}`

The hierarchical-only and joint conditions use the same goal context and connective; the joint condition additionally varies the segment instruction through multi-aspect rephrasings.

C.1 Language Supervision Results by Seed

Table 9 reports the two seed results underlying the averages in Table 2. Each displayed mean is calculated before rounding. For RoboCasa, the mean is $(s_{42} + s_{142})/480$, where s_{42} and s_{142} are success counts over 240 trials per seed.

Table 9: Language supervision results: mean, followed by (seed 42 / seed 142). Metrics are success rate for RoboCasa 365 and mean subgoal progress for BEHAVIOR-1K. The task-level condition uses a fixed task instruction and is shown once per benchmark.

	RoboCasa 365		BEHAVIOR-1K	
Language Supervision	Oracle	Fable 5	Oracle	Fable 5
Task-level (fixed)	0.169 (0.175 / 0.163)		0.076 (0.050 / 0.102)	
Subtask-level	0.406 (0.408 / 0.404)	0.381 (0.392 / 0.371)	0.143 (0.130 / 0.156)	0.084 (0.070 / 0.098)
Task + Subtask	0.417 (0.425 / 0.408)	0.402 (0.408 / 0.396)	0.154 (0.140 / 0.168)	0.091 (0.080 / 0.102)
Hierarchical only	0.456 (0.463 / 0.450)	0.440 (0.446 / 0.433)	0.143 (0.130 / 0.156)	0.084 (0.070 / 0.098)
Multi-aspect only	0.425 (0.433 / 0.417)	0.406 (0.417 / 0.396)	0.160 (0.150 / 0.170)	0.103 (0.090 / 0.116)
Hierarchical + Multi-aspect	0.517 (0.525 / 0.508)	0.485 (0.492 / 0.479)	0.174 (0.160 / 0.188)	0.143 (0.130 / 0.156)

D Frontier Planning and Observation-Based Instruction Dispatch

This appendix describes the frontier-instructor protocol used in Section 4 and the high-level plan evaluation in Probe 2. The protocol separates initial plan generation, execution monitoring, and instruction dispatch.

D.1 Plan Generation and Reuse Across Policy Arms

At the start of each episode ($t = 0$), a frontier vision-language model receives the composite task description and the reset-frame camera views. It generates the complete ordered subgoal sequence once. For a given evaluation episode and planner/phrasing condition, this sequence is reused across every policy arm. Instruction content is therefore held fixed across arms while the policy interface varies, following the same discipline as the oracle condition.

The frontier model supplies the plan before execution. Subsequent observations determine when the harness advances through that plan; they do not trigger regeneration of the subgoal sequence.

D.2 Local Progress Monitoring

A separate local open-weights vision-language model, Qwen3.6-27B, checks progress every 40 control steps using the active subgoal instruction and three-camera RGB observations from the subgoal's start and the current state. Its decisions use visual observations rather than simulator-state milestones. Monitor queries are synchronous.

The oracle condition instead uses an authored subgoal sequence and simulator-state milestones to determine completion. The frontier-instructor condition uses neither simulator state for initial planning nor simulator state for progress monitoring.

D.3 Instruction Dispatch to the Local Policy

The harness maintains a cursor into the pre-written subgoal list and hands the local policy, S1, one instruction at a time. S1 executes the current instruction between monitor queries. Two consecutive positive judgments trigger advancement to the next instruction after the configured switching buffer. Uncompleted subgoals retain the current instruction without automatic skipping or replanning. Episodes terminate on benchmark success, environment termination, or exhaustion of the 6,000-step budget for RoboCasa 365 or the 12,000-step budget for BEHAVIOR-1K.

Thus, the instruction supplied to S1 changes in response to observed completion, while the content and order of the subgoal list remain fixed. Plans are shared across policy arms, but switching times depend on each arm's observed execution progress.

D.4 Connection to Probe 2: Robustness to Instruction Diversity

The high-level plan evaluation in Section 4, Probe 2, evaluates the initial plan-generation stage described above. Opus 5, Gemini-3.1-Pro, and Fable 5 each generate a full subgoal sequence from the composite goal text and reset-frame camera views. Their plans are compared with oracle reference plans across 30 scenes using decomposition step-count match and semantic similarity, with Fable 5 serving as the semantic-similarity judge. Fable 5 compares individual generated steps with oracle-reference steps and directly assigns a semantic-similarity score in $[0, 1]$ to each comparison. Higher scores indicate closer semantic agreement.

Table 10: Plan quality against oracle references: step-count match and mean semantic similarity from direct 0–1 judgments of individual steps by Fable 5.

Metric	Opus 5	Gemini-3.1-Pro	Fable 5
Step-count match	29/30	25/30	27/30
Mean semantic similarity	.886	.855	.895

The subsequent execution evaluation varies the source planner and whether its instructions are rephrased. Table 3 reports RoboCasa 365 results from one training/evaluation seed (42). Its original-instruction Fable 5 results match the seed-42 entries in Table 9, whereas Table 2 averages seeds 42 and 142. Fable 5 generates the rephrased instructions for all three source planners. It is prompted to preserve the original task goal while changing the wording. For example, "Open up the cabinet" may be rewritten as "Grasp the cabinet door handles and swing the doors open." Such rewrites can make

action details explicit, so this condition tests variation in wording and descriptive detail rather than lexical substitution alone. Within each planner and phrasing condition, the generated sequence is reused across policy arms. All arms use the same local monitoring and cursor-based dispatch protocol. The plan-quality evaluation assesses the initial frontier-generated text; the execution evaluation measures how the policies perform with that text under observation-based instruction switching.

E Primitive Instruction Compliance

Table 11 reports the full primitive-compliance probe, including the instruction wording and movement threshold for each action.

We compare one checkpoint per condition from a single training seed of the subtask-level and hierarchical-only policies used in the main evaluation, without additional training for this probe.

Each command is evaluated three times in each of the 12 scenes, with a 24-second execution window per repetition. We average the three repetitions within each scene and then average equally across the 12 scenes. Each repetition is scored as compliant when final displacement along the commanded axis exceeds its threshold. Thus, each reported rate aggregates 36 binary trials per command and policy.

Table 11: Single-seed policy compliance rates for primitive kinematic instructions.

Primitive Action	Subtask-level	Hierarchical only	threshold
Object up ("lift the object up")	0.250	0.833	16 cm
Object down ("lower the object down")	0.139	0.750	16 cm
Arm up ("move the arm up")	0.111	0.750	34 cm
Arm down ("move the arm down")	0.111	0.778	34 cm
Arm forward ("move the arm forward")	0.056	0.556	24 cm
Arm back ("move the arm back")	0.028	0.444	24 cm

F Interfaces for Frontier-Guided Robot Control

Figure 8 distinguishes five routes to robot control: a policy without a frontier model (a), frontier-generated conditioning for a trained policy (b), executable programs over a fixed API (c), geometric guides (d), and direct joint commands (e). The distinction is what the frontier model emits and how that output reaches the actuators. Geometric guides may be followed by a policy or converted to joint commands through inverse kinematics, so type (d) is defined by the guide rather than its receiver.

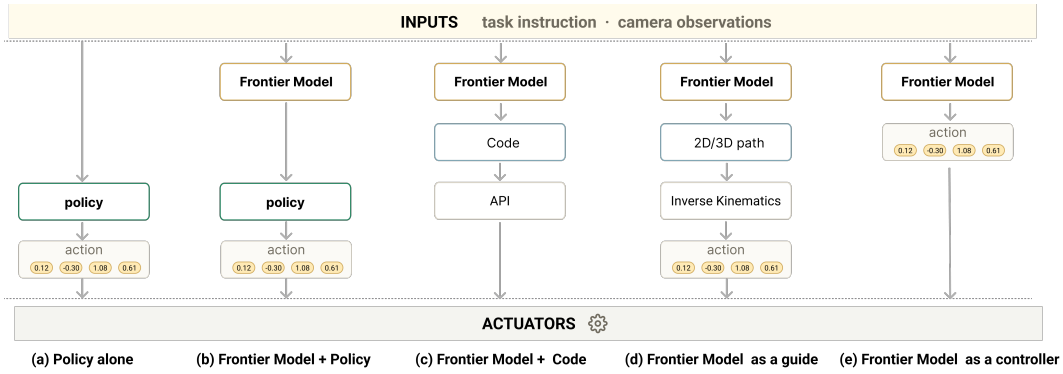


Figure 8: Routes from task instructions to robot actions.

G AUTONOMOUS DEMONSTRATION GENERATION

G.1 AUTONOMOUS ROLLOUT PROMPT

The following prompt is used for autonomous rollouts in Section 3.1. The success-verification clause below is the supplied Operate a Claw Machine example; its shark-and-basket condition is task-specific. Angle-bracketed fields denote the task instruction and data paths.

> Complete "<TASK_INSTRUCTION>" with an SO-101 robot, using **five successful demonstrations** from <DEMONSTRATION_DATASET> as in-context references.
>
> 1. **Learn and act.** Review all five demonstrations and observe the current scene. Infer a suitable strategy, execute with visual feedback, and adapt to the robot's actual response. Preserve calibrated hardware limits and stop when safe execution cannot be established.
> 2. **Verify success.** Confirm that the shark remains supported inside the upright basket after release and gripper withdrawal, with the other toys outside.
> 3. **Record trajectories.** Save every attempt in <OUTPUT_DIRECTORY>, including measured states, executed actions, camera observations, timestamps, source references and outcome. Build the synthetic corpus from verified successes; label failures separately.
> 4. **Report results.** Report the outcome, segment reuse, **episode-only estimated API cost**, **episode elapsed time**, and **robot command-execution time** separately. Exclude preparation and postprocessing; state measurement limitations. Create a review video for successful episodes, omitting camera-free intervals.

G.2 ACCUMULATED-EXPERIENCE PROMPT EXTENSION

For the generation-efficiency experiment in Section 3.2, the following instruction is added to the rollout prompt:

> **Improve through experience.** Add useful successful segments to <SEGMENT_POOL>, separate from the original demonstrations. Reuse and adapt them when the current situation matches. Update the strategy after each episode to improve reliability, time and cost.

The original five demonstrations and the growing segment pool are maintained separately. The model can reuse relevant successful segments in later episodes. The reported generation-efficiency measurements exclude preparation, postprocessing, and failed attempts; they describe successful episodes rather than the total cost of corpus construction. Episode elapsed time and robot command-execution time are recorded separately.

G.3 CORRECTIVE DEMONSTRATION SEGMENTS

Corrective demonstration segments are short examples of adjustments made when execution encounters physical errors. Table 12 reports the number of segments used for each task. These counts refer to corrective segments, rather than complete successful trajectories or evaluation trials.

Table 12: Number of corrective demonstration segments used per task.

Task	Corrective segments
Operate a Claw Machine	8
Stack Cups	8
Fold a Cloth	16
Close a Lid	16
Physical Crossword Puzzle	32

Figure 9 illustrates alignment toward target cubes from different initial gripper offsets in four crossword pickup demonstrations.

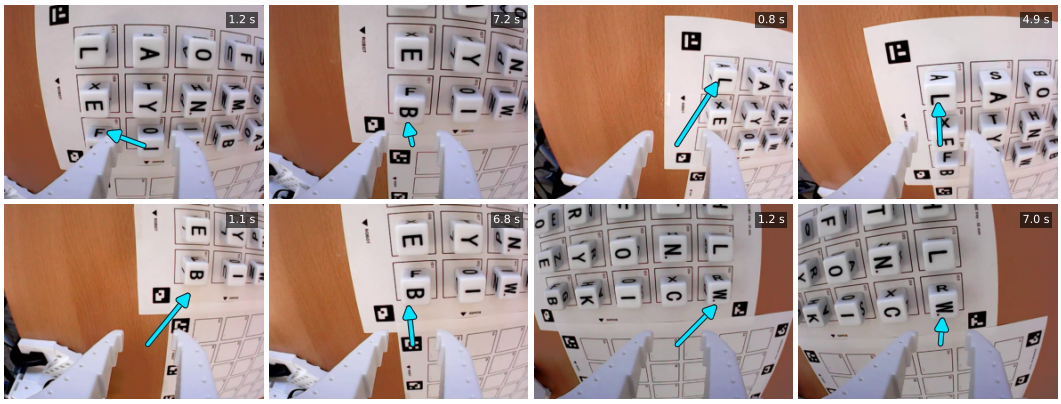


Figure 9: Illustrative corrective demonstrations for crossword cube pickup. Each adjacent pair shows the initial offset (left) and subsequent approach (right) in a recorded wrist-camera sequence. The top row shows B and L cube pickups; the bottom row shows a second B cube pickup and a W cube pickup. Cyan arrows indicate illustrative alignment directions toward the target, manually added for visualization; they are not recorded action vectors or overlays supplied to the model. Timestamps indicate elapsed time within each episode.

G.4 Physical Policy and Deployment Interface

The physical controller is an OpenPI $\pi_{0.5}$ vision-language-action policy with a PaliGemma backbone (`gemma_2b`) and a flow-matching action expert (`gemma_300m`). It controls an SO-101 arm using overhead and wrist RGB images, six measured joint/gripper state values, and a natural-language instruction. The deployment interface uses 224×224 RGB images and duplicates the overhead frame into the policy's two external-camera input slots. This policy input contract is distinct from the frontier model's camera access.

Policy inference runs remotely over WebSocket. Each prediction provides ten absolute joint-target actions; the robot uses six action channels, corresponding to five arm joints and the gripper. The intended action rate is 30 Hz, so ten actions represent approximately 333 ms of motion. The documented scheduling strategy issues actions every 33.3 ms and replaces the unexecuted remainder of a chunk when a new prediction arrives.

The 30 Hz rate is a target rather than a consistently achieved measurement. Early diagnostics reported 3.38 end-to-end cycles per second, while later bench measurements reported control rates of 18.5–19.9 Hz. These are different timing measurements from diagnostic runs and do not establish a uniform achieved rate for the task evaluations. Camera acquisition, preprocessing, communication, and inference all contribute to the realized timing.

The instruction is supplied in the `prompt` field of every inference request. Updating that field can condition the next action chunk without retraining or reloading the policy. The interface alone does not establish a tested mid-execution interruption protocol or switching latency; the documented offline instruction-swap probe compares predictions rather than demonstrating successful physical task switching.

G.5 Elapsed-Time Accounting and Limitations

For the physical deployment comparison, elapsed time is averaged over successful trials only; failed and timed-out trials contribute to success rates but not to the time average. Elapsed time includes frontier reasoning and planning, camera observation and verification, tool calls, API and communication latency, physical action execution, and task-level corrective retries. Hardware-fix intervals are excluded. Consequently, these means measure time conditional on success rather than expected time per attempt or total time to obtain a success.

Even with the same prompt, the frontier agent can choose different sequences of observation, verification, and preparatory actions across trials. Service and communication issues introduce addi-

tional variability. The prompt does not enforce an identical operational sequence, and excluding hardware-fix intervals limits what the reported times establish about total deployment effort. We do not attribute this variability to unobserved internal model mechanisms. The horizontal coordinates in Figure 5 report elapsed task time averaged over successful trials. The crossword activity timeline in Figure 11 instead illustrates one run per configuration and is not an average across games.

G.6 Human-Only and Mixed Demonstration Corpora

We compare policy success after training on 40 human demonstrations per task with training on a mixed corpus of 20 human and 20 autonomously generated demonstrations per task (Figure 10). Each condition is evaluated over 20 trials per task. The mixed and human-only corpora yield success rates of 90% versus 90% on Operate a Claw Machine, 60% versus 55% on Stack Cups, 40% versus 50% on Fold a Cloth, and 15% versus 10% on Close a Lid. Both conditions achieve 41 successes across 80 trials (51.25%).

At the same total demonstration count, the mixed corpus uses half as many human demonstrations and attains the same aggregate success in this evaluation, with task-specific differences. This result supports the feasibility of using generated trajectories in place of part of the human training corpus; it does not establish statistical equivalence or consistent improvement across tasks. We do not evaluate a 20-human-only condition, so this comparison does not isolate the incremental benefit of adding generated trajectories to those 20 human demonstrations.

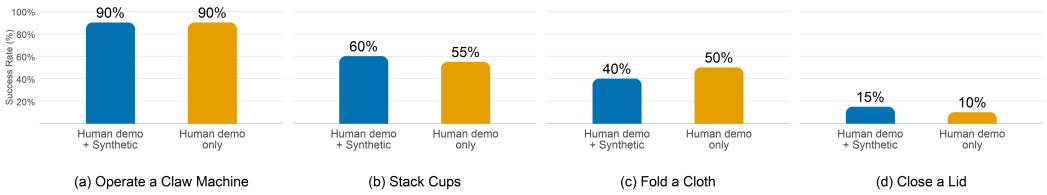


Figure 10: Policy success rates with mixed and human-only training corpora. "Human demo + Synthetic" uses 20 human and 20 autonomously generated demonstrations per task; "Human demo only" uses 40 human demonstrations per task. Each bar reports success over 20 evaluation trials. Both conditions achieve 41/80 successes across the four tasks.

H Crossword Evaluation Results

Table 13 reports individual game scores and completed-word counts underlying the means in Section 5. Each game uses a different alphabet set and a one-hour execution budget, with scoring as defined in the main text. The harness baseline uses subtask-level supervision, while the dense-supervision harness uses the hierarchical + multi-aspect recipe. Within each instruction condition, both use the same demonstrations, policy architecture, training budget, and GPT-6 Astra frontier controller; the comparison varies language supervision only. Word validity is judged by the authors based on familiar English usage. This is human adjudication rather than an automated dictionary-based metric. Repositioning an already credited cube or repeating an already credited word earns no additional points.

Policy training configuration. Both crossword policies are initialized from pretrained $\pi_{0.5}$ (Black et al., 2025) and trained with an action horizon of 10, batch size 32, and a budget of 8,000 steps. The peak learning rate is 2.5×10^{-5}. These settings are shared by the subtask-level and hierarchical + multi-aspect supervision conditions.

Instructions without cell IDs. In an additional comparison, we explicitly instruct the frontier model not to use board cell IDs such as "1A", "3B", or "6F" in its directives to the local policy. It must instead describe destinations spatially, for example, "put the cube on the left of the B cube" rather than "put the cube on 1A". Table 13 reports five games under this restriction. Mean scores fall from 14.4 to 2.8 without dense supervision and from 15.6 to 5.6 with dense supervision. Under the restriction, dense supervision yields higher scores in four games and ties in one. These results support improved performance with spatially described targets, while both policies remain substantially less effective

Table 13: Crossword results after one hour per game. Each game cell reports total score / completed words. Scores award +2 per correctly placed cube and +2 per completed valid word; the final column reports mean total score. The final two rows prohibit cell IDs in frontier-generated instructions.

Configuration	Game 1	Game 2	Game 3	Game 4	Game 5	Mean score
Fable 5 (direct control)	6 / 0	4 / 0	4 / 0	6 / 0	4 / 0	4.8
GPT-6 Astra (direct control)	8 / 0	4 / 0	10 / 1	10 / 1	8 / 0	8.0
GPT-6 Astra harness	14 / 2	16 / 2	14 / 1	14 / 1	14 / 1	14.4
GPT-6 Astra harness + dense supervision	16 / 2	16 / 2	14 / 1	16 / 1	16 / 2	15.6
Cell IDs prohibited in instructions						
GPT-6 Astra harness	4 / 0	4 / 0	2 / 0	2 / 0	2 / 0	2.8
GPT-6 Astra harness + dense supervision	6 / 1	4 / 0	8 / 1	6 / 1	4 / 0	5.6

when cell IDs are unavailable in instructions. Qualitatively, placement was more difficult because spatial descriptions could leave the intended target ambiguous; placement and recovery success were not separately quantified.

Pick-and-place evaluation. Table 14 reports accuracy over 20 trials per configuration, excluding retries. This measures first-attempt execution separately from the one-hour game scores above. The local policy can be evaluated on individual pick-and-place trials even though it cannot independently perform the full crossword task.

Table 14: Crossword pick-and-place accuracy over 20 trials per configuration, excluding retries.

Configuration	Successful trials	Accuracy
Fable 5 (direct control)	5/20	25%
GPT-6 Astra (direct control)	13/20	65%
Local policy only	10/20	50%
GPT-6 Astra harness	10/20	50%

H.1 Observation and Execution Timing

The frontier model autonomously decides when to read the available camera views and how long to execute each action segment. For example, it can select 2–3 s segments for transport and 0.12 s segments for fine adjustments. These durations describe physical execution rather than frontier-model inference latency.

Figure 11 compares one-hour activity timelines for GPT-6 Astra under direct control and frontier harness execution. In the illustrated direct-control run, GPT-6 Astra repeatedly reasons about successive raw actions, with 2 min 40 s spent on physical execution. In the illustrated harness run, action generation is delegated to the local policy, and physical execution occupies 9 min 8.55 s. This example is consistent with the harness allocating more time to robot motion while retaining observation-based guidance and correction. These durations correspond to 4.4% and 15.2% of the one-hour budget, respectively, or 3.43 times as much physical execution with the harness. This ratio measures time spent executing actions, not movement speed; the comparison uses one illustrated run per configuration and does not estimate an average timing advantage across games.

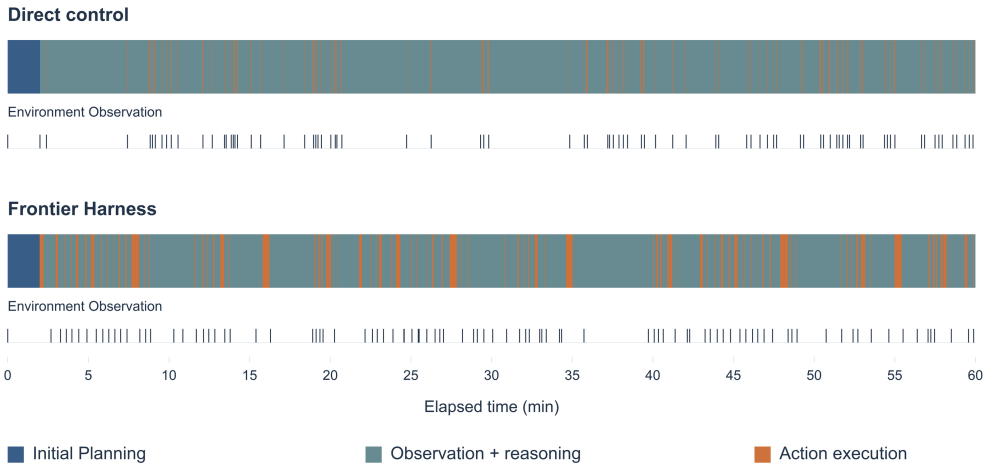


Figure 11: Full 60-minute crossword activity timelines for GPT-6 Astra, one run per configuration. Ticks mark camera observations requested by the model. Cumulative action-execution time is 2 min 40 s under direct control and 9 min 8.55 s with the harness; these are individual runs, not averages across games.