LADDER: A LONG-HORIZON ROBOT MANIPULATION BENCHMARK

Anonymous authors
Paper under double-blind review

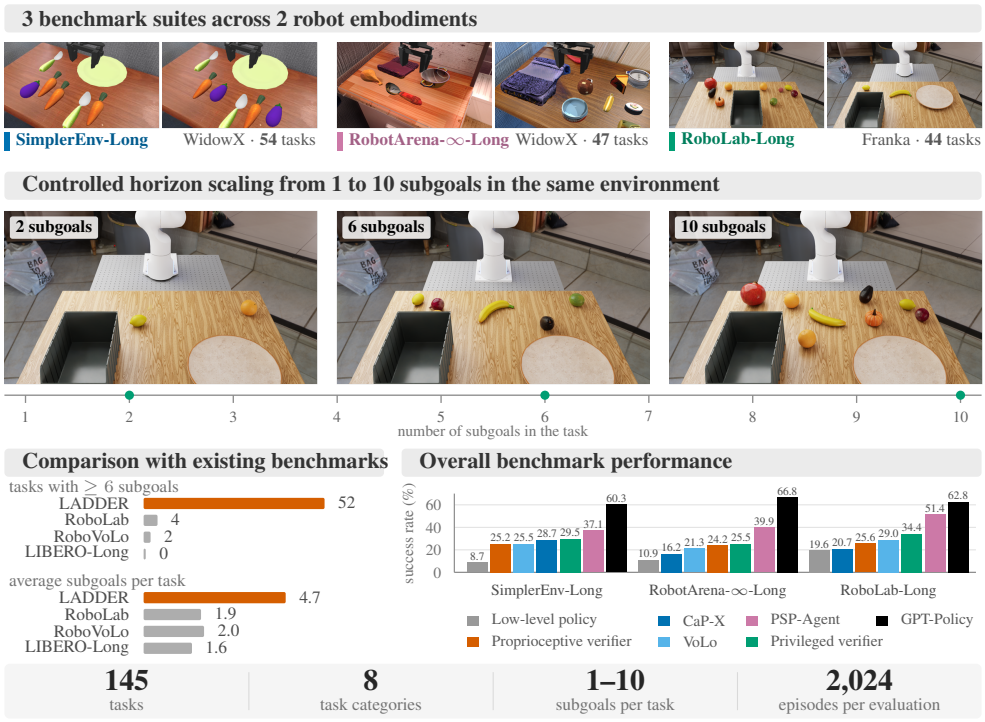

Figure 1: **Overview of our benchmark and key findings.** Across all suites, orchestrating policy execution with sequential subgoal decomposition and visual verification substantially outperforms issuing the full instruction directly to the policy (bottom right). See Tables 1 and 3.

ABSTRACT

General-purpose robot manipulation policies learn short-horizon skills effectively, but degrade rapidly on long-horizon tasks. Existing benchmarks cannot isolate failures because task horizon is either too short or entangled with scene and object variations. We propose LADDER, a long-horizon robot manipulation benchmark scaling from 1 to 10 subgoals within the same environment over three suites (SimplerEnv-Long, RobotArena-∞-Long, and RoboLab-Long), 145 tasks, eight categories, two embodiments, and 2,024 simulated episodes per system. We evaluate two orchestration paradigms: text subgoals for learned low-level policies (VoLo) and tool selection orchestration (CaP-X, GPT-Policy, PSP-Agent, TiPToP). Given the full instruction directly, low-level policies achieve 85.4% on single-subgoal tasks, but drop to near zero on extended tasks. While orchestration remedies this, it remains bounded by physical execution errors of the low-level policy rather than high-level planning and achieves only ∼25%. Tool selection orchestration with GPT-Policy on GPT-6 Astra significantly outperforms learned-policy methods, but incurs significant time and cost budget and succeeds only ∼62%. Real-world experiments across two embodiments confirm the collapse of the low-level policy and the gain from orchestration. Project website: long-horizon-benchmark.github.io

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.

1 INTRODUCTION

General-purpose robot manipulation policies learn diverse skills from large datasets of teleoperated demonstrations (Brohan et al., 2023; Walke et al., 2023; Khazatsky et al., 2024; Black et al., 2025a; NVIDIA, 2026). However, these demonstrations predominantly cover short-horizon, single-subgoal tasks. Because collecting demonstrations for every multi-subgoal combination is infeasible (Mees et al., 2022; Liu et al., 2023), generalist robot models rely on *zero-shot skill composition*: using low-level policies trained on single-subgoal tasks to solve long-horizon tasks without task-specific fine-tuning.

When presented directly with full instructions involving multiple subgoals or conditional constraints, current policies degrade rapidly. Existing benchmarks offer limited ability to isolate why this failure occurs: suites such as CALVIN (Mees et al., 2022) and LIBERO-Long (Liu et al., 2023) evaluate short horizons and often require in-domain training rather than testing zero-shot composition. Recent multi-task evaluations such as RoboLab (Yang et al., 2026) scale task complexity, but conflate task horizon with scene clutter and instruction phrasing. Systematically isolating long-horizon composition requires evaluating low-level policies zero-shot across extended horizons while holding the environment and the object sizes fixed.

In this paper, we introduce LADDER, a benchmark for long-horizon robot manipulation composed of three evaluation suites: SimplerEnv-Long, RobotArena-∞-Long, and RoboLab-Long. Because evaluating thousands of multi-subgoal trials on physical hardware is infeasible at scale, we use physics-based simulation environments that closely match real setups and have demonstrated strong sim-to-real correlation (Li et al., 2024; Jangir et al., 2026). Rather than inflating task horizon through unstructured scene changes, our benchmark structures long-horizon manipulation across **eight spatial and logical task categories** (Figure 1). Across all suites, task horizon scales systematically from 1 to 10 subgoals within the same environment, using explicit programmatic predicates to measure intermediate subgoal progress alongside full task completion.

The benchmark evaluates 145 tasks across two robot embodiments (WidowX and Franka) over 2,024 simulated episodes per system. On single-subgoal control tasks, low-level policies achieve a high success rate: 85.4% in simulation and 66.7% on the real WidowX. However, when given multi-subgoal tasks (2 to 10 subgoals), the low-level policy collapses, falling to 5.2% to 15.8% in simulation (0.5% at 8 to 10 subgoals) and, in real-world experiments, to 1.2% on WidowX and 5.6% on Franka.

To bridge the long-horizon gap, recent frameworks deploy high-level orchestrators across distinct execution backends: decomposing instructions into text subgoals for learned policies (Chen et al., 2026), synthesizing executable code over motion primitives (Fu et al., 2026), or generating symbolic plans for task-and-motion planners (Shen et al., 2026). Although not specifically designed for long-horizon tasks, GPT-Policy (Cheng et al., 2026) uses GPT-6 Astra to invoke robot tools and achieves the strongest performance on LADDER. We therefore systematically study the orchestrator design space, characterizing four modular mechanisms: subgoal planning, execution triggering, visual verification, and failure recovery. Orchestration enables low-level policies to execute tasks up to 10 subgoals without fine-tuning, improving multi-subgoal success to 20.6% to 22.7% in simulation and 31.0% to 58.3% in real-world experiments. Tool selection systems, which call motion primitives instead of a learned policy, go further. In all cases, LADDER remains challenging despite some methods spending impractical budgets. However, decomposition alone is insufficient: low-level policies frequently fail on individual subgoals. Offline, vision-language models decompose instructions accurately (up to 93.1%), but struggle with scene state tracking (at most 70.8%).

Our contributions are as follows:

- A long-horizon manipulation benchmark across three suites, spanning 1 to 10 subgoals and eight structured task categories with programmatic progress predicates (145 tasks, 2,024 simulated episodes per system).
- A systematic study of hierarchical orchestration over frozen low-level policies, characterizing the modular roles of planning, triggering, visual verification, and failure recovery, evaluated alongside representative alternative paradigms that achieve higher success at a higher cost.
- Real-world experiments across two robot platforms (WidowX and Franka) demonstrating that orchestration improves multi-subgoal execution.

2 Related Work

Manipulation benchmarks. Existing robot manipulation benchmarks provide limited means to study how task horizon affects the performance of a system (see Table 1 for an overview). CALVIN (Mees et al., 2022) and LIBERO-Long (Liu et al., 2023) include multi-step tasks, but their standard evaluation protocol for learned policies involves fine-tuning on in-domain demonstrations. Their simulated environments also have limited alignment with real-world robot appearance and dynamics, so success in these benchmarks does not reliably transfer to the real-world. SimplerEnv (Li et al., 2024) and RobotArena-∞ (Jangir et al., 2026) reduce this gap by matching simulated scenes and robot dynamics to real data, while RoboLab (Yang et al., 2026) uses photorealistic simulation. However, the first two focus primarily on single-step tasks, while RoboLab and its long-horizon extension RoboVoLo (Chen et al., 2026) vary task horizon along other axes of complexity, such as scene background and out of distribution objects. In contrast, building on SimplerEnv, RobotArena-∞, and RoboLab, we propose a benchmark that isolates task horizon scaling by systematically increasing the number of subgoals of a task while holding the environment and underlying skill fixed.

Hierarchical orchestrators. Recent advances in the visual and semantic reasoning capabilities of foundation models have motivated hierarchical orchestration as an approach to long-horizon manipulation. Orchestration systems use frontier multi-modal models to break an instruction into subgoals, and delegate their execution to a robot-control backend. Prior work has used code over motion primitives (Liang et al., 2023; Fu et al., 2026), task and motion planning (TAMP) (Shen et al., 2026), and learned skills or policies (Ahn et al., 2022; Huang et al., 2022; Belkhale et al., 2024; Shi et al., 2025; Li et al., 2025; Chen et al., 2026) as backends. However, it remains unclear how different design choices of these systems affect performance as task horizon systematically increases.

Table 1: **Comparison with existing manipulation benchmarks.** *Sim-to-Real* indicates whether policy performance in the simulator transfers to or correlates with physical robots. CALVIN chains five instructions but issues each one separately, so each of its tasks has one subgoal. *Avg Subgoals* is the average number of object placements per task. *Incremental Subgoal Tasks* counts tasks that extend a shorter task of the same scene with more subgoals, or are extended by a longer one, to isolate the effect of task horizon.

Benchmark	Sim-to-Real	Tasks	Avg Subgoals	Max Subgoals	Tasks with $\geq$ 6 subgoals	Incremental Subgoal Tasks
CALVIN (Mees et al.)	×	34	1.0	1	0	0
LIBERO-Long (Liu et al.)	×	10	1.6	2	0	0
SimplerEnv (Li et al.)	✓	4	1.0	1	0	0
RobotArena-∞ (Jangir et al.)	✓	74	1.0	2	0	0
RoboLab (Yang et al.)	✓	120	1.9	11	4	56
RoboVoLo (Chen et al.)	✓	126	2.0	7	2	2
LADDER	✓	**145**	**4.7**	**10**	**52**	**138**

3 Long-Horizon Manipulation Benchmark

We build LADDER on three established platforms: SimplerEnv (Li et al., 2024), RobotArena-∞ (Jangir et al., 2026), and RoboLab (Yang et al., 2026). Each suite—*SimplerEnv-Long*, *RobotArena-∞-Long*, and *RoboLab-Long*—keeps the original assets while adding four key features (Table 1): (1) **controlled horizon scaling**, from 1 to 10 subgoals with 138 tasks that extend a shorter task of the same scene or are extended by a longer one, and 36% of tasks comprising 6 or more subgoals; (2) **diverse task categories**, covering eight spatial and logical task structures (Figure 2); (3) **explicit per-subgoal predicates**, checking every subgoal programmatically; and (4) **multiple embodiments**, evaluating policies across multiple robot platforms.

Problem setting. We define a *long-horizon task* as one that requires completing several subgoals in sequence from a single full instruction. Each *subgoal* is a single pick-and-place action (for example, putting a carrot on a plate), the target location is its *destination*, and the number of subgoals is the task's *horizon*. An *episode* is one execution rollout starting from a randomized scene layout.

The systems we evaluate control the robot with a low-level policy, a task-and-motion planner, or motion primitives. A low-level policy maps visual observations and a language instruction to robot control actions. Generalist low-level policies are trained on demonstration datasets (e.g., BridgeData

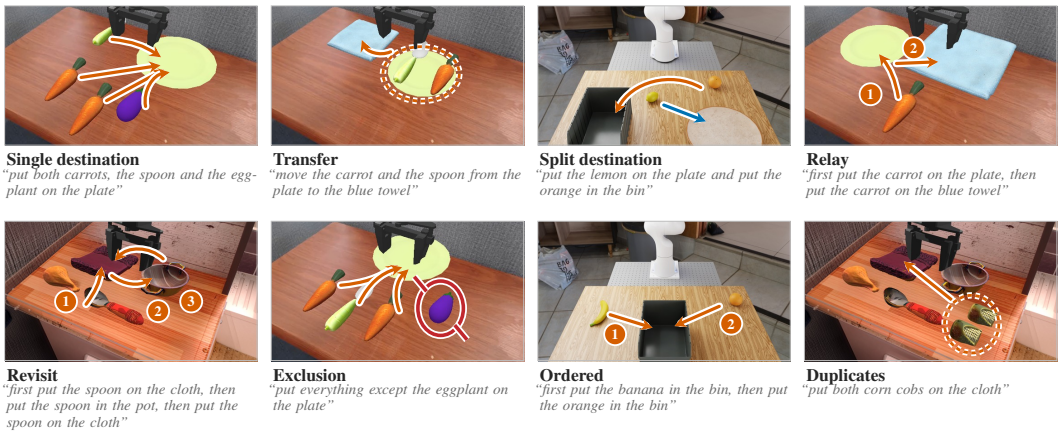

Figure 2: **Manipulation task categories.** Example scenes and natural language instructions across diverse task structures. Visual overlays indicate target placements and task constraints.

and DROID) of short-horizon skills, like placing one object in a container (e.g., a plate, towel, bowl, or bin). We evaluate them without fine-tuning on data of the target tasks and environments.

Task horizon. Task horizon is the number of subgoals an instruction requires the robot to complete. To isolate the effect of horizon, tasks form ladders: within one scene and task category, a longer task keeps all subgoals of a shorter one and adds more (for example, *"put the carrot on the plate"* and *"put the carrot and the spoon on the plate"*). Every task on such a ladder is *incremental*. In total, 138 of our 145 tasks are incremental (Table 1). Except in exclusion tasks, instructions explicitly name each target object rather than using vague collective phrases (like *"put everything in the bin"*), ensuring that we test instruction following rather than random searching. Tasks scale from 1 up to 10 subgoals, which reaches the maximum object density that fits in the workspace without objects blocking basic grasps.

Task categories. Task categories define the spatial and logical structure of each instruction (Figure 2). Across the three suites, we evaluate eight task categories (every task: Appendix D): **single destination**, moving multiple objects to one container (e.g., *"put the spoon and the carrot on the plate"*); **split destinations**, sorting objects into different containers (e.g., *"put the lemon on the plate and the orange in the bin"*); **transfers**, moving objects out of one container into another (e.g., *"move the carrot and the spoon from the plate to the blue towel"*); **relay routing**, moving an object through an intermediate stop (e.g., *"first put the carrot on the plate, then on the towel"*); **revisitations**, returning to a container used earlier (e.g., *"put the spoon on the cloth, then in the pot, then back on the cloth"*); **exclusions**, moving specific objects while leaving distractors alone (e.g., *"put everything except the eggplant on the plate"*); **ordered sequences**, following a strict order (e.g., *"first put the banana in the bin, then the orange"*); and **duplicate instances**, picking specific items among identical visual copies (e.g., *"put both corn cobs on the cloth"*). Evaluating all eight categories allows us to measure how different geometric and logical constraints affect policy success (§5.1).

Scene construction. Each scene is physically stable and within the robot's reachable workspace. A longer task adds more objects and a larger container for them; object sizes stay the same, so grasping difficulty does not grow with the horizon. Every layout is validated by scripted policies that complete all subgoals, so failures are caused by sequential execution rather than reachability issues.

Evaluation metrics. We evaluate every subgoal with programmatic state checks. We report two main metrics: **Task Success Rate (%)**, the percentage of episodes where all subgoals and constraints are satisfied at the end of the episode, and **Subgoal Progress (%)**, the percentage of required subgoals completed, measuring partial progress on unfinished tasks. For every subgoal that is not completed, we also report a **Failure Cause**, such as a subgoal that was completed and later undone (§5.2). The step budget grows with the number of subgoals: 100 steps (20 s at 5 Hz) per subgoal on the WidowX suites, and a per-horizon budget of 900 to 4950 steps (60 s to 330 s at 15 Hz) on RoboLab-Long. The budget counts simulated time only: the simulation pauses while a system computes.

4 CHARACTERIZING ORCHESTRATORS

To analyze how different orchestration methods support long-horizon manipulation, we characterize their operation around four design decisions (Figure 3, Table 2): (1) subgoal planning, (2) trigger mechanisms, (3) visual verification, and (4) failure recovery. This characterization informs our experimental comparisons and analysis, helping us identify how decisions in each of these aspects affect task execution and performance. We divide the evaluated systems into two groups. *Learned-policy* systems send text subgoals to a low-level policy. *Tool selection* systems do not use a low-level policy. Instead, they call motion primitives or a task-and-motion planner. In addition to existing systems, we introduce PSP-Agent (Plan-See-Pick Agent), a tool selection system in which a VLM plans text subgoals and executes each subgoal by calling manipulation primitives that verify their own outcome (Appendix B).

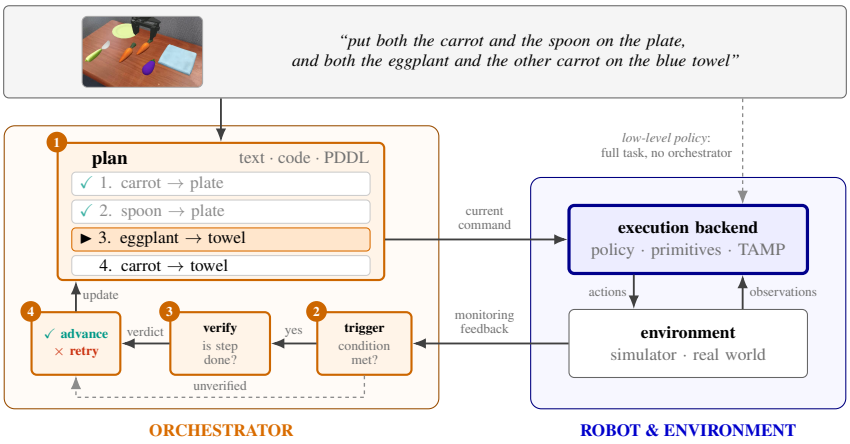

Figure 3: **Overview of the orchestrator architecture.** The top box shows the task input (initial camera view and full instruction). Orange numbered boxes mark the four orchestrator decisions evaluated in our study (Table 2). The blue box denotes the execution backend. Dashed lines show baselines without orchestration (*low-level policy*) or without verification (*unverified*).

1. Subgoal planning. A plan decomposes a full instruction into sequential single-subgoal instructions (for example, decomposing *"put the carrot and the spoon on the plate"* into subgoal 1: *"put the carrot on the plate"* and subgoal 2: *"put the spoon on the plate"*). In our experiments, the planner is always a highly capable frontier Vision-Language Model (VLM). The only exception is the privileged verifier, which takes the ground-truth subgoals from the simulator. Our learned-policy variants and PSP-Agent generate the full list of text subgoals once, at the start of the episode (*Text subgoals*). VoLo (Chen et al., 2026), which we run with the low-level policy as its only execution tool, also generates a list of text subgoals from the first frame and replans the remaining ones during the episode. The other external systems use different plan representations: CaP-X (Fu et al., 2026) generates a Python program, TiPToP (Shen et al., 2026) generates symbolic goal predicates for its planner, and GPT-Policy (Cheng et al., 2026) selects only the next tool call. *Low-level policy* is the no-orchestrator baseline and passes the full instruction to the low-level policy without generating a subgoal plan.

2. Trigger mechanisms. Because the low-level policy runs continuously without knowing when to stop, the orchestrator needs a trigger condition to decide when to switch or advance to the next subgoal. We evaluate three subgoal-switching strategies: (1) *Periodic trigger*: advances after a fixed number of steps; (2) *Proprioceptive trigger*: advances to the next subgoal when internal robot state indicates an attempt has finished, specifically when the gripper opens after a sustained close (Appendix B); (3) *Privileged trigger*: a simulation-only diagnostic that checks ground-truth contacts at every step, advancing to the next subgoal the instant the current goal is reached; it is used by the *Privileged verifier*. Tool selection systems execute blocking commands and make the next decision when a command completes, either after each primitive call (*Primitive completion*) or after the full program (*Program completion*). TiPToP executes its plan open-loop, without a trigger.

3. Visual verification. When a trigger fires, a verifier determines whether the subgoal succeeded. Across our controlled variants and the privileged verifier, we use three verification settings: no verifier

(*None*), which advances unconditionally; visual verification (*Visual (VLM)*), which queries a vision-language model using the latest camera observation and subgoal description; and simulator-state verification (*Simulator state*), used by the privileged verifier to check the task predicate directly. PSP-Agent verifies each primitive geometrically from the depth image (*Geometric state*), for example by checking that a grasped object has left its initial position. CaP-X and GPT-Policy query the VLM with the new camera image.

4. Failure recovery. After verification, the recovery mechanism determines whether to advance, retry, or replan. In our retry-based variants, a completed subgoal is removed from the queue and execution advances; an incomplete subgoal is retried in place. When a plan contains duplicate subgoals for interchangeable objects, an incomplete subgoal moves to the end of the queue so one difficult object does not consume the episode budget. The privileged verifier retries immediately. Without a verifier, the orchestrator always advances to the next subgoal (*Advance*). To replan (*Replan*), the VLM generates a new plan from the current camera image: VoLo replans the remaining subgoals, CaP-X generates a new program, and GPT-Policy selects a new tool call. PSP-Agent combines both mechanisms: it retries failed subgoals and replans after all subgoals have been attempted (Appendix B).

Table 2: **Overview of the evaluated systems according to the proposed characterization.** Each configuration maps to the orchestrator decisions in Figure 3. Top: systems using a low-level policy backend. Bottom: systems using alternative execution backends.

System	Backend	Plan	Trigger	Verification	Recovery
Low-level policy	Learned policy	None	None	None	None
Periodic trigger	Learned policy	Text subgoals	Periodic	None	Advance
Proprioceptive trigger	Learned policy	Text subgoals	Proprioceptive	None	Advance
Proprioceptive verifier	Learned policy	Text subgoals	Proprioceptive	Visual (VLM)	Retry
VoLo (Chen et al., 2026)	Learned policy	Text subgoals	Periodic	Visual (VLM)	Replan
Privileged verifier	Learned policy	Ground-truth subgoals	Privileged	Simulator state	Retry
CaP-X (Fu et al., 2026)	Motion primitives	Python program	Program completion	Visual (VLM)	Replan
TiPToP (Shen et al., 2026)	Symbolic TAMP	Goal predicates	None	None	None
PSP-Agent	Motion primitives	Text subgoals	Primitive completion	Geometric state	Retry, replan
GPT-Policy (Cheng et al., 2026)	Motion primitives	Next tool call	Primitive completion	Visual (VLM)	Replan

5 EXPERIMENTS

We first study how task success scales with task horizon (§5.1). Then we investigate the dominant failures in different systems (§5.2) and whether simulation predicts real-world performance (§5.3).

Experimental setup and low-level policies. On *SimplerEnv-Long* and *RobotArena-∞-Long* we evaluate a π_0-like (Black et al., 2025b) low-level policy that uses Qwen3-VL (Qwen Team, Alibaba, 2025) as backbone and is trained on BridgeData v2 (Walke et al., 2023). On *RoboLab-Long* we use the public $\pi_{0.5}$ (Black et al., 2025a) fine-tuned on DROID (Khazatsky et al., 2024) (Appendix A). For periodic-trigger variants, we use 27 steps (5.4 s at 5 Hz) on the WidowX suites and 250 steps (16.7 s at 15 Hz) on RoboLab-Long. In both cases, the interval is set to the median time the privileged verifier takes to complete a subgoal. Unless noted otherwise, visual verifiers use Gemini 3.7 Flash. GPT-Policy uses GPT-6 Astra. In simulation, the tool selection systems use the ground-truth depth image and camera calibration from the simulator. In real-world experiments, depth is estimated with MoGe-2 (Wang et al., 2025) from calibrated cameras. For the simulation results we report 95% Wilson confidence intervals over episodes, and test differences between systems paired over tasks (Appendix F).

5.1 TASK HORIZON SCALING

On single-subgoal tasks, the low-level policy reaches a similar success rate as the *Proprioceptive verifier* (85.4% vs. 80.2%). As task horizon grows, the low-level policy rapidly collapses to 26.9% at 2 subgoals and 0.0% at 5 subgoals. In contrast, orchestrator-based methods maintain higher success rates with increasing task horizon with *Proprioceptive verifier* achieving 58.4% at 2 subgoals, 8.3% at 5 subgoals and retaining partial progress up to 10 subgoals (Figure 4, Table 3). The Privileged verifier serves as a privileged reference for the specific low-level policy being evaluated. The best learned-policy orchestration methods, VoLo and the *Proprioceptive verifier*, decrease the gap of the

low-level policy to this reference from 18.6 to 4.2–4.5 percentage points. Tool selection systems, which use ground-truth depth in simulation, are not limited by the low-level policy. GPT-Policy achieves the highest success rate on every suite (62.0% overall), and PSP-Agent is second (40.1%) while using the same VLM as the learned-policy orchestrators. Both exceed the privileged reference of the learned-policy systems, and on 5-subgoal tasks PSP-Agent still succeeds in 27.3% of episodes, against 11.1% for the privileged verifier. CaP-X performs on par with the best learned-policy methods (25.0%), while TiPToP (9.4%) does not improve over the low-level policy.

Table 3: **Task success rate (%) of every system.** Top: per suite, over all tasks (1 to 10 subgoals). Middle: per task category. Bottom: all tasks. Middle and bottom combine the episodes of the three suites. ± covers the 95% Wilson interval. Green marks the best learned-policy method, blue the best tool selection method (CaP-X, TiPToP, PSP-Agent or GPT-Policy). Privileged verifier is a privileged reference for learning methods using this specific low-level policy.

Tasks	Low-level policy	Periodic trigger	Proprioceptive trigger	Proprioceptive verifier	VoLo	Privileged verifier	CaP-X	TiPToP	PSP-Agent	GPT-Policy
SimplerEnv-Long	8.7 ±1.7	17.4 ±2.2	21.5 ±2.3	25.2 ±2.4	25.5 ±2.4	29.5 ±2.5	28.7 ±2.5	6.7 ±1.5	37.1 ±2.7	**60.3** ±2.7
RobotArena-∞-Long	10.9 ±3.6	15.2 ±4.0	16.2 ±4.1	24.2 ±4.6	21.3 ±4.4	25.5 ±4.6	16.2 ±4.1	11.2 ±3.6	39.9 ±5.0	**66.8** ±4.9
RoboLab-Long	19.6 ±4.5	25.6 ±4.8	28.1 ±4.9	25.6 ±4.8	29.0 ±4.9	34.4 ±5.1	20.7 ±4.5	17.3 ±4.3	51.4 ±5.2	**62.8** ±5.2
Single destination	14.9 ±2.9	23.5 ±3.4	25.0 ±3.4	31.0 ±3.6	31.0 ±3.6	33.8 ±3.7	23.5 ±3.4	12.9 ±2.8	44.0 ±3.8	**61.8** ±3.7
Transfer	16.7 ±6.9	14.6 ±6.7	14.6 ±6.7	15.3 ±6.8	27.1 ±7.8	26.4 ±7.7	43.8 ±8.2	10.4 ±6.1	36.1 ±8.1	**68.8** ±8.0
Split destination	7.1 ±3.5	11.5 ±4.1	17.6 ±4.7	17.2 ±4.7	20.6 ±5.0	24.3 ±5.2	16.2 ±4.6	12.2 ±4.2	50.7 ±5.7	**60.5** ±5.7
Relay	2.3 ±2.4	10.9 ±4.0	15.5 ±4.5	21.4 ±5.0	16.1 ±4.6	27.3 ±5.3	32.2 ±5.4	0.3 ±1.5	24.3 ±5.1	**71.7** ±5.3
Revisit	0.0 ±2.2	1.2 ±3.0	1.2 ±3.0	3.0 ±3.8	3.6 ±4.0	4.8 ±4.4	8.9 ±5.3	0.0 ±2.2	6.0 ±4.7	**43.5** ±7.6
Exclusion	0.6 ±2.7	31.0 ±7.4	39.9 ±7.6	39.3 ±7.5	32.7 ±7.4	42.3 ±7.6	31.0 ±7.4	10.7 ±5.6	48.8 ±7.5	**60.1** ±7.6
Ordered	22.2 ±7.5	28.5 ±7.9	29.2 ±7.9	35.4 ±8.1	38.2 ±8.1	39.6 ±8.2	27.1 ±7.8	12.5 ±6.4	56.2 ±8.2	**65.3** ±8.1
Duplicates	29.7 ±8.4	24.2 ±8.1	31.2 ±8.5	30.5 ±8.4	31.2 ±8.5	33.6 ±8.6	25.8 ±8.2	11.7 ±6.7	52.3 ±8.6	**58.6** ±8.7
Overall	11.0 ±1.4	18.4 ±1.7	21.7 ±1.8	25.0 ±1.9	25.3 ±1.9	29.6 ±2.0	25.0 ±1.9	9.4 ±1.3	40.1 ±2.2	**62.0** ±2.1

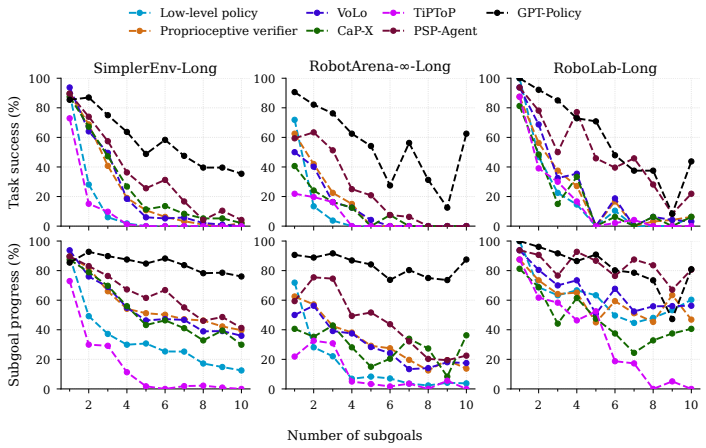

Figure 4: **Performance scaling across task horizon.** Task success rate (top) and subgoal progress (bottom) as the horizon increases from 1 to 10 subgoals across all three benchmark suites (all ten systems: Tables 11 and 13).

Trends also differ across task categories. On *Exclusion* and *Relay* tasks, the low-level policy nearly always fails while the Proprioceptive verifier approaches the privileged verifier. On *Single destination*, *Split destination*, *Transfer* and *Ordered* the gain of VoLo is also steady, but smaller. On *Duplicates* the low-level policy already approaches the privileged verifier and orchestration provides only tiny improvement. *Revisit*, whose tasks mostly have 6 or more subgoals, resists both: it needs intermediate transfers where execution errors compound, and reaches only 2.3% on average over the learned-policy systems (Table 3).

To assess the efficiency we analyze the *elapsed time* and the *compute* used by each method. We calculate *elapsed time* as the percentage of the simulated time budget needed to complete a task and *compute* as the number of VLM calls per episode (Figure 5). We do not compare wall-clock time, because it depends on the hardware of each run and on queueing at shared VLM endpoints (Appendix G). Most progress happens early: the *Proprioceptive verifier* completes 42.1% of all

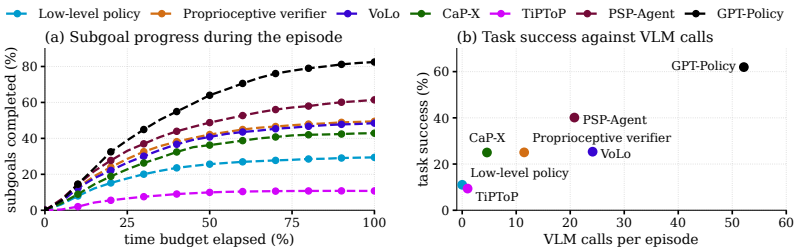

Figure 5: **Robot time and compute.** (a) Share of all subgoals completed as the simulated time budget elapses. (b) Task success against VLM calls per episode. Both combine all suites (per suite: Table 15; tokens and API cost: Appendix G).

subgoals in the first half of the budget and 49.5% by the end, indicating that a longer budget would help little. These results suggest that the end-to-end low-level policy limits the returns to test-time scaling: VoLo makes more VLM calls than the *Proprioceptive verifier* but achieves similar success, while PSP-Agent and GPT-Policy pair higher call counts with higher success using motion primitives.

5.2 Failure Analysis

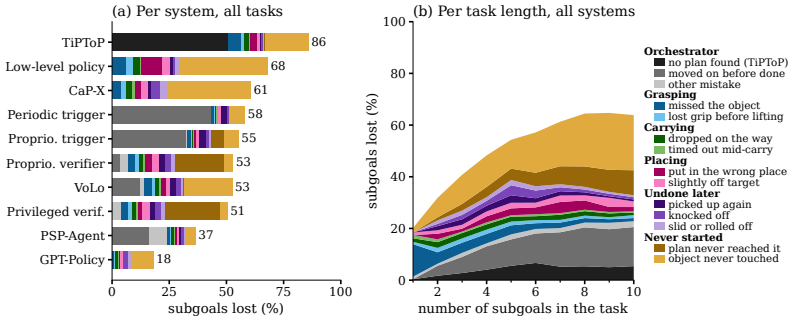

Figure 6: **Why subgoals fail.** Share of subgoals not completed at the end of the episode, by cause (Appendix H). A loss that an orchestrator decision made final is drawn as an orchestrator cause (grey; CaP-X and GPT-Policy log no decisions), every other loss by what happened to the object. (a) One bar per system, over all its episodes. (b) One point per task length, over all ten systems. Every value is computed within each suite and then averaged over the three suites (per suite: Figure 8).

Next, we investigate the dominant failure modes for each method and try to understand whether learned-policy methods are bottlenecked by the orchestrator or the low-level policy.

To evaluate orchestrator reasoning independently from execution noise, we benchmark vision-language models offline on 145 scenes and 562 annotated frames across four capabilities (Table 4): plan decomposition, subgoal verification, scene state tracking, and subgoal switching. Gemini 3.7 Flash and GPT-5.6 Sol lead plan decomposition at 93.1%. Gemini 3.7 Flash also leads state tracking (70.8%), the hardest capability for most models, and serves as our default orchestrator VLM with the highest mean accuracy over the four capabilities (80.8%).

On average 32.6–43.4% of subgoals fail because an orchestrator without verification moves on before the subgoal is completed and never returns to it (Figure 6; Table 18 in the appendix). This failure extends to PSP-Agent, which incorrectly marks 16.2% of its subgoals as completed. With the introduction of verification mechanisms, this failure mode decreases to 3.4% and orchestrator-related failures comprise only 7.1% (13.8% for VoLo). This shifts the primary limitation of learned-policy systems to the low-level policy's capabilities, with failure to manipulate subgoal target objects emerging as the main bottleneck (Low-level policy and VoLo). This is followed by a failure to reach later subgoals in the plan (Proprioceptive verifier and Privileged verifier), because the method keeps retrying earlier ones. We note that for these learned-policy methods, successful episodes on average complete within 50% of the time budget (Figure 5; Appendix G), which indicates that the allocated

Table 4: **Offline evaluation of Vision-Language Models across orchestrator capabilities.** Scores in %; ± covers the 95% Wilson interval, widened for the correlation of frames from the same scene. TP/TN rates measure detecting finished vs. unfinished subgoals. Tokens: generated per query over the four capabilities, reasoning included (Table 16).

	Planning	Subgoal completion		State tracking	Next subgoal	Tokens
Model	exact match (%)	TP rate (%)	TN rate (%)	all correct (%)	valid (%)	per query
Qwen3-VL-2B	75.2 ±7.6	68.2 ±7.0	34.1 ±5.9	9.3 ±4.0	58.9 ±5.3	–
Qwen3-VL-8B	91.0 ±5.8	69.0 ±6.8	79.9 ±5.7	47.9 ±5.5	72.3 ±4.4	–
Qwen3.5-9B	86.9 ±6.5	44.4 ±6.3	96.9 ±2.5	11.7 ±3.7	63.2 ±4.8	5,173
Qwen3.8-27B	91.0 ±5.8	**76.2 ±6.3**	84.8 ±4.3	58.2 ±4.8	79.3 ±4.1	1,904
Gemini 3.1 Pro	89.7 ±6.0	54.8 ±6.6	96.0 ±2.9	68.7 ±4.3	75.2 ±4.5	583
Gemini 3.7 Flash	**93.1 ±5.4**	59.4 ±6.6	96.6 ±2.6	**70.8 ±4.0**	81.4 ±4.2	412
Gemini Robotics-ER 2	90.3 ±5.9	54.0 ±6.5	**97.8 ±2.3**	69.6 ±4.2	76.6 ±4.1	510
GPT-5.6 Sol	**93.1 ±5.4**	72.8 ±6.3	86.4 ±4.3	56.4 ±4.7	82.0 ±4.3	282
GPT-6 Astra	91.7 ±5.7	71.1 ±6.2	86.4 ±4.4	53.2 ±5.1	**85.4 ±4.0**	203
Claude Opus 5	90.3 ±5.9	68.6 ±6.6	87.9 ±4.1	61.0 ±4.7	79.8 ±4.4	400

time budget is sufficient. Instead qualitative inspection reveals that the low-level policy ends up consuming most of the budget moving aimlessly and trying to reach the target object. Long tasks therefore fail because progress stalls: half of the failed episodes complete no subgoal after the first 28% of the budget. Reaching the target object is also the primary failure mode in tool selection systems such as CaP-X and GPT-Policy, while TiPToP fails to find a plan for 50.9% of its subgoals.

5.3 Real-World Experiments

To test whether LADDER is indicative of real-world performance, we repeat the comparison on physical WidowX and Franka robots. Due to the especially high costs and time requirements, we evaluate only a subset of the methods on task horizons ranging from 1 to 5. We observe a similar trend to simulation: the low-level policy's success rate collapses much faster than orchestrator-based methods. Furthermore, we observe preserved rankings of the methods. This is an encouraging result for using LADDER as a reliable rank-measuring tool for long-horizon tasks.

Task category	Subgoals	Low-level policy	Prop. verifier	PSP-Agent
WidowX				
Single destination	1	66.7	75.0	**100.0**
Single destination	2–3	0.0	33.3	**87.5**
Split destination	2–3	0.0	29.2	**83.3**
Exclusion	2	0.0	58.3	**91.7**
Ordered	2	8.3	33.3	**41.7**
Duplicates	2	0.0	0.0	**58.3**
Franka				
Single destination	5	0.0	8.3	**25.0**
Exclusion	2	0.0	83.3	**100.0**
Ordered	3	16.7	83.3	**100.0**

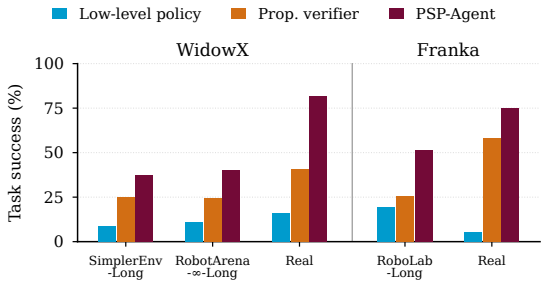

Figure 7: **Real-robot evaluation.** Left: task success (%) on the real robots per task category, 12 episodes per task and system. Right: overall task success in simulation (Table 3) and on the real robots. Both rank the three systems the same.

6 Discussion and Conclusion

Our findings highlight three takeaways: **(1) Task horizon is a primary failure mode.** Low-level policies perform well on single-subgoal tasks, but collapse as horizon grows. High-level orchestration roughly triples overall multi-subgoal success. **(2) Physical execution is the main bottleneck.** VLMs decompose instructions accurately, but tasks often fail during execution even when given ground-truth subgoals. **(3) The two best systems execute with tool-based primitives.** GPT-Policy and PSP-Agent reach 62.0% and 40.1% success overall, against at most 29.6% for the learned-policy systems.

Limitations. High-level VLM inference is slow, limiting reactive adjustments during rapid movements. Our evaluation focuses on tabletop pick-and-place rather than pushing, sliding, or articulated objects. Visual state tracking remains challenging under heavy clutter, and physical robot experiments cover fewer scenes than simulation.

Reproducibility Statement

Section 3 describes how the benchmark is built, and Appendix D lists every task with its instruction, episodes and step budget. Section 4 and Appendix B specify every evaluated system and its settings, Appendix C gives the prompts of our systems, and Appendix A describes the low-level policies. We will release the long-horizon suites, the orchestrator code, the launch configurations and the offline VLM benchmark of Section 5.2. Rollout videos are available on the project website.

AI Use Statement

We used LLM-based coding and writing assistants. They helped implement the benchmark suites, the evaluated systems and the evaluation and analysis code; they helped analyze the results, review the methodology and check the paper's claims against the recorded results; and they helped draft, edit and format the text. The authors reviewed and tested all code, verified every number, claim and reference, and take full responsibility for the paper.

Ethics Statement

This work involves no human subjects, personal data or crowdsourcing. We see no ethical concerns beyond the general safety considerations of deploying learned manipulation policies on physical robots.

References

Michael Ahn et al. Do as I can, not as I say: Grounding language in robotic affordances. In *Conference on Robot Learning (CoRL)*, 2022.

Suneel Belkhale, Tianli Ding, Ted Xiao, Pierre Sermanet, Quon Vuong, Jonathan Tompson, Yevgen Chebotar, Debidatta Dwibedi, and Dorsa Sadigh. RT-H: Action hierarchies using language. In *Robotics: Science and Systems (RSS)*, 2024.

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 *Conference on Robot Learning (CoRL)*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 2025a.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems (RSS)*, 2025b.

Anthony Brohan et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning (CoRL)*, 2023.

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*, 2026. URL `https://arxiv.org/abs/2606.07723`.

Dongzhou Cheng, Taoran Yi, Ye Fang, Xingwu Zhang, Fan Feng, Yixuan Li, Gengxiong Zhuang, Rongze Wang, Shuai Yang, Wei Song, Weizhi Xue, Minyan Wu, Jie Gui, Jiaqi Wang, and Tong Wu. In-context robot learning with VLM agents. *arXiv preprint arXiv:2609.19138*, 2026.

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

Wenlong Huang et al. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning (CoRL)*, 2022.

Yash Jangir, Yidi Zhang, Kashu Yamazaki, Chenyu Zhang, Kuan-Hsun Tu, Tsung-Wei Ke, Lei Ke, Yonatan Bisk, and Katerina Fragkiadaki. RobotArena ∞: Scalable robot benchmarking via real-to-sim translation. In *International Conference on Learning Representations (ICLR)*, 2026.

Alexander Khazatsky, Karl Pertsch, et al. DROID: A large-scale in-the-wild robot manipulation dataset. In *Robotics: Science and Systems (RSS)*, 2024.

Xuanlin Li et al. Evaluating real-world robot manipulation policies in simulation. In *Conference on Robot Learning (CoRL)*, 2024.

Yi Li, Yuquan Deng, Jesse Zhang, Joel Jang, Marius Memmel, Caelan 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.

Jacky Liang et al. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters (RA-L)*, 2022.

NVIDIA. Cosmos 3: Omnimodal world models for physical AI. *arXiv preprint arXiv:2606.02800*, 2026.

Qwen Team, Alibaba. Qwen3-VL. *Technical report*, 2025.

William Shen, Nishanth Kumar, Sahit Chintalapudi, Ryan Lindeborg, Jie Wang, Christopher Watson, Edward Hu, Jing Cao, Dinesh Jayaraman, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. TiPToP: A modular open-vocabulary robot manipulation system that plans. *arXiv preprint arXiv:2603.09971*, 2026.

Lucy Xiaoyang Shi, Brian Ichter, Michael Robert Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, Adrian Li-Bell, Danny Driess, Lachy Groom, Sergey Levine, and Chelsea Finn. Hi robot: Open-ended instruction following with hierarchical vision-language-action models. In *International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 54919–54933. PMLR, 2025.

Homer Walke, Kevin Black, Abraham Lee, et al. BridgeData V2: A dataset for robot learning at scale. In *Conference on Robot Learning (CoRL)*, 2023.

Ruicheng Wang, Sicheng Xu, Yue Dong, Yu Deng, Jianfeng Xiang, et al. MoGe-2: Accurate monocular geometry with metric scale and sharp details. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Xuning Yang, Rishit Dagli, Alex Zook, Hugo Hadfield, Ankit Goyal, Stan Birchfield, Fabio Ramos, and Jonathan Tremblay. RoboLab: A high-fidelity simulation benchmark for analysis of task generalist policies. In *Robotics: Science and Systems (RSS)*, 2026.

Appendix overview. Appendix A describes the low-level policies and compares $\pi_{0.5}$ with Cosmos-3; Appendix B gives the configuration of every orchestrator and external system; Appendix C prints the prompts; Appendix D lists every task, its episodes and step budget; Appendix E gives the real-robot results per task category; Appendix F gives the results per suite and system, their uncertainty over tasks, and the construction of the offline VLM benchmark; Appendix G covers robot time, VLM calls, tokens and cost; and Appendix H analyzes the failures of every system per suite.

A Low-Level Policy Architectures and Training Details

Table 5: **The two DROID-trained low-level policies on RoboLab-Long.** Task success and subgoal progress (%) of $\pi_{0.5}$ (Black et al., 2025a) and Cosmos-3 (NVIDIA, 2026) on the same tasks: with the low-level policy alone by number of subgoals (top), and under each orchestrator over all tasks (bottom); ± covers the 95% Wilson interval, and a dash marks a configuration not run with Cosmos-3. The orchestrators in the main text use $\pi_{0.5}$.

	Task success		Subgoal progress		
Tasks	$\pi_{0.5}$	Cosmos-3	$\pi_{0.5}$	Cosmos-3	Episodes
Low-level policy, by number of subgoals					
1 subgoal	100.0 ±19.4	100.0 ±19.4	100.0 ±19.4	100.0 ±19.4	16
2 subgoals	46.9 ±12.1	12.5 ±10.3	68.8 ±8.5	39.8 ±8.7	64
3 subgoals	22.5 ±15.0	15.0 ±14.1	63.3 ±8.9	55.8 ±8.9	40
4 subgoals	14.6 ±12.6	12.5 ±12.2	66.7 ±6.9	49.5 ±7.0	48
5 subgoals	0.0 ±13.8	0.0 ±13.8	63.3 ±8.9	40.0 ±8.9	24
6 subgoals	10.4 ±11.8	6.2 ±10.6	49.7 ±5.7	44.4 ±5.8	48
7 subgoals	0.0 ±13.8	0.0 ±13.8	44.6 ±7.6	39.9 ±7.6	24
8 subgoals	0.0 ±10.7	0.0 ±10.7	48.0 ±6.1	32.0 ±5.9	32
9 subgoals	0.0 ±13.8	0.0 ±13.8	53.2 ±6.7	35.2 ±6.6	24
10 subgoals	6.2 ±13.9	0.0 ±10.7	60.3 ±5.5	25.6 ±5.1	32
All tasks, by orchestrator					
Low-level policy	19.6 ±4.5	11.1 ±3.7	56.6 ±2.3	39.0 ±2.3	352
Periodic trigger	25.6 ±4.8	–	57.1 ±2.3	–	352
Proprioceptive trigger	28.1 ±4.9	–	58.9 ±2.3	–	352
Proprioceptive verifier	25.6 ±4.8	–	56.1 ±2.3	–	352
VoLo	29.0 ±4.9	25.6 ±4.8	62.0 ±2.3	52.5 ±2.3	352
Privileged verifier[†]	34.4 ±5.1	22.2 ±4.6	63.5 ±2.2	59.3 ±2.3	352

[†] On tasks without a prescribed order, when a subgoal is not completed within a fixed step budget, the privileged verifier moves on to the next subgoal and returns to it later. The budget was set for $\pi_{0.5}$, and Cosmos-3 needs more steps per subgoal than $\pi_{0.5}$.

We provide architecture, training, and deployment specifications for the base manipulation policies evaluated across our benchmarks:

- **Bridge baseline (WidowX embodiment)**: Trained on the 43,851 trajectories of Bridge-Data v2 (Walke et al., 2023) that carry a language instruction. The model couples a Qwen3-VL-2B vision-language backbone (2.1B parameters) with a conditional flow-matching action expert (0.7B parameters, 2.8B in total). The action expert predicts end-effector deltas (translation and rotation) and a gripper command over an action chunk of 5 steps, of which the first 4 are executed before the next query. Training uses AdamW with a base learning rate of 5e-5, linear warmup for 250 steps, a batch size of 1024, and exponential moving average (EMA) checkpointing with decay 0.99 from step 20,000, trained for 30,000 steps. Control frequency is 5 Hz on SimplerEnv-Long and RobotArena-∞-Long.
- **DROID baseline (Franka embodiment)**: Evaluates the public $\pi_{0.5}$ low-level policy (Black et al., 2025a) fine-tuned on the DROID dataset (Khazatsky et al., 2024) ($\sim$76,000 demonstration trajectories with a 7-DoF Franka Panda robot), in its joint-position version. The model pairs a PaliGemma 3B vision-language backbone ($\sim$2.9B parameters) with a flow-matching action expert (3.35B parameters in total), predicting joint positions and the gripper state over action chunks of 15 steps, of which 8 are executed. Control frequency is 15 Hz on RoboLab-Long.

- **Cosmos-3 baseline (Franka embodiment)**: Evaluates the public Cosmos3-Nano-Policy-DROID checkpoint (NVIDIA, 2026), the Cosmos-3 world model released as a policy for the DROID setup, on RoboLab-Long. Like $\pi_{0.5}$, it predicts joint positions and the gripper state from the camera views, the joint state, and the text instruction.

Table 5 compares the two DROID-trained policies on the same RoboLab-Long tasks. Cosmos-3 completes fewer subgoals than $\pi_{0.5}$ at almost every horizon, so the orchestrators in the main text use $\pi_{0.5}$. Orchestration helps both policies: VoLo raises task success from 11.1% to 25.6% with Cosmos-3 and from 19.6% to 29.0% with $\pi_{0.5}$, and the privileged verifier reaches 22.2% and 34.4%.

B Orchestrator Configurations and External Systems

Periodic trigger. It advances every 27 steps on the WidowX suites and every 250 steps on RoboLab-Long, and stays on the last subgoal.

Proprioceptive trigger. The trigger tracks the normalized gripper aperture ($[0, 1]$ travel). An attempt ends when the gripper opens past 0.5 after a close of at least 3 steps on the WidowX suites, and of at least 21 steps with a lift of 3 cm on RoboLab-Long. A cooldown of 8 steps (WidowX suites) or 57 steps (RoboLab-Long) prevents re-triggering while the jaws settle. On RoboLab-Long an attempt also ends after 300 steps without a release, which produces 68% of its advances, and the plan restarts from its first subgoal after the last. The WidowX suites have no such limit.

Proprioceptive verifier. It detects attempts in the same way, without the cooldown. A verification query is sent 5 steps (WidowX suites) or 35 steps (RoboLab-Long) after a release, and after 45 or 300 steps without one. The verifier asks for an object count rather than a binary verdict, to disambiguate repeated identical objects: a subgoal is complete once the count reaches the number of times it occurs in the plan so far (one on revisit tasks). An incomplete subgoal is retried in place; on RoboLab-Long, an incomplete subgoal with interchangeable duplicates moves to the end of the queue instead.

Privileged verifier. It reads the simulator's success predicate for the current subgoal and advances once the predicate has held for a settle window, 3 steps on SimplerEnv-Long, 10 on RobotArena-∞-Long and 120 on RoboLab-Long (to filter contact chatter on container rims), with the gripper open. On RoboLab-Long, a subgoal not completed within 600 steps is set aside and returned to later, except on ordered tasks; on the WidowX suites it is retried until the budget ends.

PSP-Agent. PSP-Agent gives the VLM seven tools: `pick_and_place`, `pick`, `place`, `push`, `measure`, `grasp_at` and `run_code` (Appendix C). The VLM plans the task from the current image as a list of goals, each naming one object and one destination. The first call for a goal is `pick_and_place` on that object and destination; after a failure, the VLM chooses the next call from the goal, the tool catalog and the reports of the calls so far. Each tool returns whether it succeeded and, on failure, a short description of the cause (for example, that the gripper closed without an object). A goal is given up for the current pass after two failed calls that move the arm. After a pass over all goals, the VLM plans again from the new image and the reported outcomes; an empty plan is checked once more, and a second empty plan ends the episode. To localize an object, a tool fits an upright box to the depth points inside the bounding box the VLM returns for its name. A grasp is checked with depth on the WidowX suites (the object has left its start position) and with the jaw width on RoboLab-Long, and a release is checked with depth over the destination on SimplerEnv-Long and RoboLab-Long.

External systems. We evaluate four external baselines and alternative execution backends:

- **VoLo** (Chen et al., 2026): Evaluated in its default subgoal mode, with the low-level policy as its only execution tool (the full system can also call action primitives). The VLM splits the task into an ordered list of subgoals from the first frame, checks progress every 20 steps on SimplerEnv-Long, 15 on RobotArena-∞-Long and 150 on RoboLab-Long, and replans the remaining task when a subgoal fails, at most three times.

- **CaP-X** (Fu et al., 2026): We evaluate the multi-turn variant of CaP-X with high-level primitives. A VLM generates a Python program over six primitives: `get_object_pose`, `sample_grasp_pose`, `goto_pose`, `open_gripper`, `close_gripper`, and `home_pose`. `get_object_pose` segments the named object with SAM3 in the scene camera, lifts the mask into 3D with the calibrated depth, and fits a box, rather than reading privileged simulator state; grasps are computed from the box. Unlike the release, the grasp is taken from straight above the box center rather than sampled with Contact-GraspNet, the box extends down to the support surface, and on RoboLab-Long the arm ramps to each target within a clamped workspace. After each program, a second VLM query compares the frames before and after execution and decides whether to regenerate the program or finish.
- **TiPToP** (Shen et al., 2026): Performs open-loop task-and-motion planning from an initial scene perception. A VLM extracts 2D object bounding boxes and goal predicates, SAM-2 segments masks, metric depth reconstructs point clouds, M2T2 proposes grasp candidates, RANSAC estimates support planes, and cuTAMP with cuRobo solves for collision-free trajectory plans. Its planner finds a plan for most single-subgoal tasks but fails more often as subgoals are added: no plan is found for 5.2% of its subgoals at one subgoal and 74.0% at five.
- **GPT-Policy** (Cheng et al., 2026): A VLM agent that chooses one tool call at a time. We use its agent loop and tools (`move_to`, `move_eef_chunk`, `set_gripper`, `locate_point`, `done`, and `give_up`), with our description of the robot in its prompt. We answer `locate_point`, for up to 8 pixels per call, with the calibrated depth camera. We disable `check_path`, because the simulators do not provide the robot's inverse kinematics.

VLM settings. In simulation, every hosted VLM query uses Gemini 3.7 Flash (`gemini-3.7-flash`), except those of GPT-Policy, which uses GPT-6 Astra (`gpt-6-astra`); all were queried between 10 and 24 September 2026. The Proprioceptive verifier, PSP-Agent, VoLo and TiPToP decode at temperature 0.0, CaP-X at 1.0, and GPT-Policy at the provider's default. Generation is capped at 2048 tokens for verification, for PSP-Agent's planning and tool choice (1024 for grounding) and for GPT-Policy, at 8192 for VoLo and TiPToP, and at 20,480 for CaP-X. Reasoning effort is the provider's default, except *low* for TiPToP. A verification query generates about 220 tokens, reasoning included.

C Vision-Language Prompts and Tool Catalog

We print the prompts of the two systems we built, as they ran: the verification query of the *Proprioceptive verifier*, and the prompt templates and tool catalog of PSP-Agent. Words in angle brackets are filled in at run time. The external systems use their own prompts: VoLo and TiPToP those of their releases, CaP-X those of its multi-turn variant, and GPT-Policy its agent prompt with our description of the robot.

Completion verification. Sent by the *Proprioceptive verifier* with the current subgoal and frame on all three suites.

> You are checking the CURRENT state of a robot tabletop scene from its camera views. The subtask below moves ONE object to a goal location, and the scene may hold SEVERAL interchangeable objects of that same kind. Count how many DISTINCT objects of the kind the subtask names are in the subtask's END STATE RIGHT NOW – at the goal location the subtask asks for, regardless of how they got there. An object counts as placed when any part of it rests on the goal location, INCLUDING when partially hidden by the robot arm or other objects. An object the gripper is still holding does NOT count, however close to the goal location it is held: the release is not yet a placement and it is the release that most often fails. An object away from the goal location, or lifted well above it, is not counted. Count each physical object at most once, however many views show it. First write one short line listing what you counted, then output STRICT JSON: {"count": <integer>}.

Verification, container suffix. Appended to every verification query.

> When a goal location is a container whose interior you cannot see into, do not look for the object inside it. Instead account for every object of that kind in the scene: if an object of that kind is no longer resting anywhere else in the scene, and the gripper is not holding it, it is in the container and counts as placed. Objects still resting elsewhere do not count.

Verification, multi-view suffix. Appended when more than one camera is sent, which is the case on RoboLab-Long (main and wrist views).

> Several camera views of the SAME scene at the SAME instant are provided, each labelled. They do not show different moments. A view may not contain the goal location at all – judge from whichever view shows it, and treat evidence from a close-up view as authoritative over a distant one where the goal location is small or occluded.

PSP-Agent, planning. Sent with the current frame at the start of the episode and after every pass over the plan. Goals already worked on are listed with the tools' reports in the form below.

> You are planning for a <robot> working on a tabletop, seen from its camera. Task: <task>
> <attempted>Break the task into the smallest goals that each move ONE object to ONE destination. Name every object by its colour and kind as it LOOKS in the image ("the yellow lemon", "the blue bin"), never by a role or a bare noun, because those names are what will be pointed at in the image later and several objects may be of the same kind. Only include goals the task actually asks for. An object the task does not mention must be left where it is, even when it sits among the ones that move. When the task states an order ("first ... then ..."), keep the goals in the order the task gives them, and list a goal TWICE if the task asks for the same object to be moved twice. Otherwise order the goals so that anything placed on an open surface, where a later reach could knock it off, comes after anything dropped into a container. Leave out any goal the image shows is already met, and return an empty list of goals when the whole task is done. Give each goal as a sentence, plus the object it moves and where that object goes, named the same way. Output STRICT JSON and nothing else: {"goals": [{"goal": "...", "object": "...", "destination": "..."}]}

PSP-Agent, attempted goals. Filled into the planning prompt from the second pass on.

> Already worked on, with what the executor reported. The image is the evidence of what actually happened, so ask for one of these again ONLY if the picture shows it did not take – repeating a goal that took sends the arm back over it and knocks it over: <lines>

PSP-Agent, re-check. Appended to the planning prompt, which is sent again, when a plan comes back empty; a second empty plan ends the episode.

> Look again before stopping. The episode is NOT over, and stopping early throws the rest of it away: on a two-goal SimplerEnv unit 57% of the steps were spent standing still after the agent decided it was finished, while a third of the goals were still unmet. Name anything the task asks for that is not done, however small. Where the destination HIDES what goes into it – a bin, a box, a pot you cannot see the bottom of – the executor's report is the evidence and the picture is not. A goal reported done there IS done, and asking for it again sends the hand back in to take the object out. Where the destination is OPEN – a plate, a cloth, a patch of table – the picture is the evidence and the report is not. If the object is not lying on it, that goal is NOT done, whatever was reported: a released object rolls off, and a later reach knocks it off. Return an empty list only when every part of the task is either visible in the picture or reported done into something you cannot see inside.

PSP-Agent, tool choice. Sent with the current frame for every tool call after the first one of a goal; the catalog is listed below.

> You are controlling a <robot>. The image is the scene right now. Goal: <goal> The gripper is <hand> right now.
> Available tools: <catalog>
> <history>Choose ONE tool call that makes progress on this goal. pick_and_place is the cheap path and handles most goals in one call. When it reports something the hand could not do, do not repeat it: the tools below it are yours – measure the thing, then take hold of it where YOU judge the jaws will close, or push it there along the surface when it is too wide to hold at all, and place it yourself. run_code drives the same API when a goal needs something none of the tools expresses. If a previous attempt failed, do not repeat it unchanged: name the OBJECT differently, since that name is what gets pointed at in the image and it may not match what is visible. The destination is what the goal asks for – describe it differently if it

was not found, but never put the object somewhere else. Output STRICT JSON and nothing else: {“tool”: “<name>”, “args”: {...}}

PSP-Agent, grounding. Sent with the current frame whenever a tool locates an object or a destination by name.

Point out <name> in this image. Draw it TIGHTLY around <name> alone. Anything lying beside or touching it must be OUTSIDE the box, even where that means cutting the box in close. Then add one box for EVERY other object lying near it, each with its own label, so <name> can be told apart from them. A name can say WHICH one it means by where it lies or what it is beside. Box the one it points at, and box the others of the same kind as well, so they can be told apart. Never reply with an empty list because the wording is not an exact name. Reply with ONLY a JSON list of boxes, each [ymin, xmin, ymax, xmax] normalised to 0-1000: [{“box_2d”: [y0, x0, y1, x1], “label”: “...”}] Return an empty list [] only if nothing of that kind is in the picture at all.

PSP-Agent, tool catalog. The *catalog* of the tool-choice prompt: each tool’s signature and description.

pick_and_place(object_name: str, destination_name: str): Move one object to one destination. This is the whole of most goals; prefer it over calling pick and place separately.
pick(object_name: str): Grasp an object and lift it clear of the surface. Name it the way it looks, with its colour (“the yellow lemon”), so the right one is grounded when several are similar. The jaws close on the widest part of it they can span, since that is the part a slightly wide approach still catches; use `grasp_at` to take hold somewhere else.
place(destination_name: str): Put the object already in the gripper onto or into a destination, named the same way. This needs a pick to have succeeded first; to move something still on the table, use pick_and_place. Releases above the destination’s own top surface and on a clear part of it, so it works for a plate, for a cloth, and for the inside of a bin.
push(object_name: str, destination_name: str): Slide something along the surface to a destination with the closed hand, without lifting it. This is how to move what the jaws cannot take hold of: a pot or a bowl too wide to span, something too heavy, something flat with nothing to grip. It cannot put one thing INTO another or stack it on top, and it drags whatever stands in the way, so lift when you can and push when you cannot.
measure(object_name: str, surface: bool = False): What the cameras and depth say about one thing, so you can decide where to take hold of it yourself. Reports where it is, how big it is, how wide and how thick it is along its length section by section, and what the hand can actually close on. The jaws hold a prism, not a scoop, so the widest part of a thing is not always the part to take it by. Pass surface=True for something to put things ON, like a plate or a cloth, which is too flat to ground otherwise.
grasp_at(object_name: str, x_m: float, y_m: float, z_m: float, yaw_deg: Optional[float] = None): Take hold of an object at a point YOU choose, then lift it. Use it when `pick` reports it cannot take something – a pot too wide to span has a handle, a bowl has a rim, a spoon has a stem – and `measure` to find where those are. `yaw_deg` turns the hand about the vertical; leave it out to keep the object’s own angle. Everything else is as `pick`: the jaws open on the way, close on the point, and the object is checked and lifted.
run_code(code: str): Run Python against this same API when a goal needs something the other tools do not cover. `pick`, `place`, `pick_and_place`, `measure`, `grasp_at`, `go_home`, `locate(name, surface=False)` (returns an object with `center_m`, `extent_m`, `top_m`, `yaw_rad` and `cross_section_m()`, the width and height along its length; pass surface=True for something to put things ON, like a plate or a cloth, which is too flat to ground otherwise), `move_to(position_m, yaw_rad)`, `open_gripper`, `close_gripper` and `np` are already defined. Every skill returns a report with `ok`, True when it finished, and `text`. Print what you learn.

D BENCHMARK SUITES AND COMPLETE TASK CATALOGS

We provide complete task specifications for all 145 evaluation tasks across the three benchmark suites: 54 tasks in SimplerEnv-Long (Table 7), 47 tasks in RobotArena-∞-Long (Table 8), and 44 tasks in RoboLab-Long (Table 9); one RobotArena-∞-Long task, `scene45_all_3`, repeats another (Table 8). Table 6 summarizes the eight task categories with representative instructions.

Table 6: **The eight kinds of task**, with an example instruction for each. The kind of task decides what one instruction has to express, and so what an orchestrator can add. *Tasks* counts the tasks of that kind across the three suites; Figure 2 shows one frame per kind, with what its instruction asks for drawn on the scene.

Task category	Requirement	Example instruction	Tasks
Single destination	every object goes to the same destination	*"put the carrot and the spoon on the plate"*	44
Transfer	every object starts on one destination and must be moved to another	*"move the carrot and the spoon from the plate to the blue towel"*	8
Split destination	different objects go to different destinations	*"put the carrot on the plate and put the spoon on the blue towel"*	25
Relay	objects pass through an intermediate destination	*"first put the carrot on the plate, then put the carrot on the blue towel"*	20
Revisit	objects return to a destination they have already left	*"first put the spoon on the cloth, then put the spoon in the pot, then put the spoon on the cloth"*	13
Exclusion	some objects must be left where they are	*"put everything except the eggplant on the plate"*	13
Ordered	the placements must occur in a stated order	*"first put the banana in the bin, then put the orange in the bin"*	14
Duplicates	identical objects cannot be told apart in language	*"put two spoons on the plate and put one spoon on the blue towel"*	8

Episodes and step budgets. Each episode of a task starts from its own initial layout: SimplerEnv-Long draws 24 seeded layouts per task, RobotArena-∞-Long shifts the objects of one authored layout by 8 fixed offsets of up to 3 cm, and RoboLab-Long jitters every object's position and yaw at reset. The step budget is 100 steps per subgoal on the WidowX suites. On RoboLab-Long it is 900, 1350, 2700, 3600, 4050, 4500 and 4950 steps for 1, 2, 3 to 6, 7, 8, 9 and 10 subgoals. In relay and revisit tasks a later leg of an object counts only once its earlier legs are done, and an ordered task fails when its objects are placed out of order. The primitives and plans of the tool selection systems execute as ordinary control steps, so their motions count against the same budget. Each task counts one run of each system, with all its episodes. A run that stopped before recording its results, or in which a check of the RoboLab-Long videos for unrendered scenes flagged an episode, was repeated; where a task was nevertheless measured twice, only one of the runs is counted. This happened for 20 tasks of a system, 11 of them VoLo's on RoboLab-Long. Summed over each system's repeated tasks, the counted runs never succeed more often than the others, and only for VoLo does the choice move a suite's success by more than half a point: 29.0% on RoboLab-Long with its later runs counted, and 31.0% with the earlier ones. In total, the paper reports 21,296 simulated episodes: 2,024 for each of the ten systems of Table 3 and 1,056 for the three Cosmos-3 configurations of Table 5.

Table 7: **Every SimplerEnv-Long task**, with the instruction the policy is given, grouped by kind of task and ordered by subgoal count.

Task	Subgoals	Instruction
Single destination		
`all1_plate`	1	"put the carrot on the plate."
`all1_towel`	1	"put the spoon on the blue towel."
`all2_plate`	2	"put the carrot and the spoon on the plate"
`all2_towel`	2	"put the carrot and the spoon on the blue towel"
`all3_plate`	3	"put the carrot, the spoon and the eggplant on the plate"

SimplerEnv-Long, continued

Task	Subgoals	Instruction
all3_towel	3	"put the carrot, the spoon and the eggplant on the blue towel"
all4_plate	4	"put both carrots, the spoon and the eggplant on the plate"
all4_towel	4	"put both carrots, the spoon and the eggplant on the blue towel"
all5_plate	5	"put both carrots, both spoons and the eggplant on the plate"
all5_towel	5	"put both carrots, both spoons and the eggplant on the blue towel"
all6_plate	6	"put both carrots, both spoons and both eggplants on the plate"
all6_towel	6	"put both carrots, both spoons and both eggplants on the blue towel"
all7_plate	7	"put the three carrots, both spoons and both eggplants on the plate"
all7_towel	7	"put the three carrots, both spoons and both eggplants on the blue towel"
all8_plate	8	"put the three carrots, the three spoons and both eggplants on the plate"
all8_towel	8	"put the three carrots, the three spoons and both eggplants on the blue towel"
all9_plate	9	"put the three carrots, the three spoons and the three eggplants on the plate"
all9_towel	9	"put the three carrots, the three spoons and the three eggplants on the blue towel"
all10_plate	10	"put the four carrots, the three spoons and the three eggplants on the plate"
all10_towel	10	"put the four carrots, the three spoons and the three eggplants on the blue towel"
Transfer		
move_plate_to_towel_2	2	"move the carrot and the spoon from the plate to the blue towel"
move_plate_to_towel_3	3	"move the carrot, the spoon and the eggplant from the plate to the blue towel"
move_towel_to_plate_3	3	"move the carrot, the spoon and the eggplant from the blue towel to the plate"
move_plate_to_towel_4	4	"move both carrots, the spoon and the eggplant from the plate to the blue towel"
move_plate_to_towel_5	5	"move both carrots, both spoons and the eggplant from the plate to the blue towel"
Split destination		
split2	2	"put the carrot on the plate and put the spoon on the blue towel"
split3	3	"put the carrot and the spoon on the plate and put the eggplant on the blue towel"
split4	4	"put one carrot and the spoon on the plate and put the eggplant and one carrot on the blue towel"
split5	5	"put one carrot, one spoon and the eggplant on the plate and put one carrot and one spoon on the blue towel"
split6	6	"put one carrot, one spoon and one eggplant on the plate and put one carrot, one spoon and one eggplant on the blue towel"
split7	7	"put two carrots, one spoon and one eggplant on the plate and put one spoon, one eggplant and one carrot on the blue towel"
Relay		
relay2e	2	"first put the carrot on the plate, then put the carrot on the blue towel"
relay3e	3	"first put the carrot on the plate, then put the carrot on the blue towel, then put the spoon on the plate"
relay4e	4	"first put the carrot on the plate, then put the carrot on the blue towel, then put the spoon on the plate, then put the spoon on the blue towel"
relay5e	5	"first put the carrot on the plate, then put the carrot on the blue towel, then put the spoon on the plate, then put the spoon on the blue towel, then put the eggplant on the plate"
relay6e	6	"first put the carrot on the plate, then put the carrot on the blue towel, then put the spoon on the plate, then put the spoon on the blue towel, then put the eggplant on the plate, then put the eggplant on the blue towel"
relay7e	7	"first put the carrot on the plate, then put the carrot on the blue towel, then put the spoon on the plate, then put the spoon on the blue towel, then put the eggplant on the plate, then put the eggplant on the blue towel, then put the carrot on the plate"
relay8e	8	"first put the carrot on the plate, then put the carrot on the blue towel, then put the spoon on the plate, then put the spoon on the blue towel, then put the eggplant on the plate, then put the eggplant on the blue towel, then put the carrot on the plate, then put the carrot on the blue towel"
relay9e	9	"first put the carrot on the plate, then put the carrot on the blue towel, then put the spoon on the plate, then put the spoon on the blue towel, then put the eggplant on the plate, then put the eggplant on the blue towel, then put the carrot on the plate, then put the carrot on the blue towel, then put the spoon on the plate"
relay10e	10	"first put the carrot on the plate, then put the carrot on the blue towel, then put the spoon on the plate, then put the spoon on the blue towel, then put the eggplant on the plate, then put the eggplant on the blue towel, then put the carrot on the plate, then put the carrot on the blue towel, then put the spoon on the plate, then put the spoon on the blue towel"
Revisit		
shuttle7	7	"first put the carrot on the plate, then put the spoon on the plate, then put the eggplant on the plate, then put the carrot on the blue towel, then put the spoon on the blue towel, then put the eggplant on the blue towel, then put the carrot on the plate"
shuttle8	8	"first put the carrot on the plate, then put the spoon on the plate, then put the eggplant on the plate, then put the carrot on the blue towel, then put the spoon on the blue towel, then put the eggplant on the blue towel, then put the carrot on the plate, then put the spoon on the plate"
shuttle9	9	"first put the carrot on the plate, then put the spoon on the plate, then put the eggplant on the plate, then put the carrot on the blue towel, then put the spoon on the blue towel, then put the eggplant on the blue towel, then put the carrot on the plate, then put the spoon on the plate, then put the eggplant on the plate"
shuttle10	10	"first put the carrot on the plate, then put the spoon on the plate, then put the eggplant on the plate, then put the carrot on the blue towel, then put the spoon on the blue towel, then put the eggplant on the blue towel, then put the carrot on the plate, then put the spoon on the plate, then put the eggplant on the plate, then put the carrot on the blue towel"

SimplerEnv-Long, continued

Task	Subgoals	Instruction
Exclusion		
except_eggplant_{2,3,4,5}	2–5	"put everything except the eggplant on the plate"
Ordered		
ordered_plate_2	2	"first put the carrot on the plate, then put the spoon on the plate"
ordered_plate_3	3	"first put the carrot on the plate, then put the spoon on the plate, then put the eggplant on the plate"
Duplicates		
spoons2	2	"put one spoon on the plate and put one spoon on the blue towel"
spoons3	3	"put two spoons on the plate and put one spoon on the blue towel"
spoons4	4	"put two spoons on the plate and put two spoons on the blue towel"
spoons5	5	"put three spoons on the plate and put two spoons on the blue towel"

Table 8: **Every RobotArena-∞-Long task**, with the instruction the policy is given, grouped by kind of task and ordered by subgoal count. The scene45_all_N tasks give the instruction of scene45_Ngoal, word for word for $N = 2, 3$ and with two objects named in the opposite order for $N = 4$ to 6. For $N = 3$ to 6 they also share its scene, so scene45_all_3 repeats scene45_3goal; scene45_all_2 has a larger cloth and moved objects. In scene54_transfer_2 both objects start on the table rather than on the cloth.

Task	Subgoals	Instruction
Single destination		
scene36_1goal	1	"put the fork in the pan."
scene45_1goal	1	"put the blue bowl on the purple cloth."
scene54_1goal	1	"put the spoon on the cloth."
scene76_1goal	1	"put the metal pot on the yellow cloth."
scene36_2goal	2	"put the fork and the corn cob in the pan"
scene45_2goal, scene45_all_2	2	"put the blue bowl and the sushi toy on the purple cloth"
scene54_2goal	2	"put the spoon and the chicken drumstick on the cloth"
scene76_2goal	2	"put the metal pot and the mushroom toy on the yellow cloth"
scene45_3goal, scene45_all_3	3	"put the blue bowl, the sushi toy and the banana on the purple cloth"
scene50_3goal	3	"put the stuffed animal, the blue block and the cheese in the pot"
scene54_3goal	3	"put the spoon, the chicken drumstick and the pot on the cloth"
scene45_4goal	4	"put the blue bowl, the sushi toy, the cheese and the banana on the purple cloth"
scene45_all_4	4	"put the blue bowl, the sushi toy, the banana and the cheese on the purple cloth"
scene45_5goal	5	"put the blue bowl, the sushi toy, the cheese, the banana and the mushroom on the purple cloth"
scene45_all_5	5	"put the blue bowl, the sushi toy, the banana, the cheese and the mushroom on the purple cloth"
scene45_6goal	6	"put the blue bowl, the sushi toy, the cheese, the banana, the mushroom and the corn cob on the purple cloth"
scene45_all_6	6	"put the blue bowl, the sushi toy, the banana, the cheese, the mushroom and the corn cob on the purple cloth"
Transfer		
scene54_transfer_2	2	"move the spoon and the chicken drumstick from the cloth to the pot"
Split destination		
scene54_split_2	2	"put the spoon on the cloth and put the chicken drumstick in the pot"
scene54_split_3	3	"put the spoon and the corn cob on the cloth and put the chicken drumstick in the pot"
Relay		
scene54_relay_2	2	"first put the spoon on the cloth, then put the spoon in the pot"
scene54_route_2	2	"first put the spoon in the pot, then put the spoon on the cloth"
scene54_relay_4	4	"first put the spoon on the cloth, then put the spoon in the pot, then put the chicken drumstick on the cloth, then put the chicken drumstick in the pot"
scene54_route_4	4	"first put the spoon in the pot, then put the spoon on the cloth, then put the chicken drumstick in the pot, then put the chicken drumstick on the cloth"
scene54_route_6	6	"first put the spoon in the pot, then put the spoon on the cloth, then put the chicken drumstick in the pot, then put the chicken drumstick on the cloth, then put the corn cob in the pot, then put the corn cob on the cloth"
scene54_route_7	7	"first put the spoon in the pot, then put the spoon on the cloth, then put the chicken drumstick in the pot, then put the chicken drumstick on the cloth, then put the corn cob in the pot, then put the corn cob on the cloth, then put the cheese on the cloth"
scene54_route_8	8	"first put the spoon in the pot, then put the spoon on the cloth, then put the chicken drumstick in the pot, then put the chicken drumstick on the cloth, then put the corn cob in the pot, then put the corn cob on the cloth, then put the cheese in the pot, then put the cheese on the cloth"
Revisit		
scene54_revisit_3	3	"first put the spoon on the cloth, then put the spoon in the pot, then put the spoon on the cloth"
scene54_revisit_6	6	"first put the spoon on the cloth, then put the spoon in the pot, then put the spoon on the cloth, then put the chicken drumstick on the cloth, then put the chicken drumstick in the pot, then put the chicken drumstick on the cloth"

RobotArena-∞-Long, continued

Task	Subgoals	Instruction
scene54_revisit_7	7	"first put the spoon on the cloth, then put the spoon in the pot, then put the spoon on the cloth, then put the spoon in the pot, then put the spoon on the cloth, then put the spoon in the pot, then put the spoon on the cloth"
scene54_revisit_8	8	"first put the spoon on the cloth, then put the spoon in the pot, then put the spoon on the cloth, then put the spoon in the pot, then put the spoon on the cloth, then put the spoon in the pot, then put the spoon on the cloth, then put the spoon in the pot"
scene54_revisit_9	9	"first put the spoon on the cloth, then put the spoon in the pot, then put the spoon on the cloth, then put the chicken drumstick on the cloth, then put the chicken drumstick in the pot, then put the chicken drumstick on the cloth, then put the corn cob on the cloth, then put the corn cob in the pot, then put the corn cob on the cloth"
scene54_revisit_10	10	"first put the spoon on the cloth, then put the spoon in the pot, then put the spoon on the cloth, then put the spoon in the pot, then put the spoon on the cloth, then put the spoon in the pot, then put the spoon on the cloth, then put the spoon in the pot, then put the spoon on the cloth, then put the spoon in the pot"
Exclusion		
scene45_except_{2,3,4,5,6}	2–6	"put everything except the small can on the purple cloth"
scene54_except, scene54_except_3	2, 3	"put everything except the pot on the cloth"
Ordered		
scene54_2goal_ordered	2	"first put the chicken drumstick on the cloth, then put the spoon on the cloth"
scene76_2goal_ordered	2	"first put the metal pot on the yellow cloth, then put the mushroom toy on the yellow cloth"
scene54_3goal_ordered	3	"first put the corn cob on the cloth, then put the spoon on the cloth, then put the chicken drumstick on the cloth"
Duplicates		
scene54_duplicates_2	2	"put both corn cobs on the cloth"
scene54_duplicates_3	3	"put the three corn cobs on the cloth"

Table 9: **Every RoboLab-Long task**, with the instruction the policy is given, grouped by kind of task and ordered by subgoal count. The suffix of comp_relay_N and comp_revisit_N counts objects, so these tasks have $2N$ and $3N$ subgoals.

Task	Subgoals	Instruction
Single destination		
comp_bin_1	1	"put the banana in the bin."
comp_plate_1	1	"put the banana on the plate."
comp_bin_2	2	"put the banana and the orange in the bin"
comp_bin_6	6	"put the banana, the orange, the lemon, the lime, the avocado and the onion in the bin"
comp_bin_10	10	"put the banana, both oranges, the lemon, the lime, the avocado, the onion, the lychee, the pumpkin and the pomegranate in the bin"
Transfer		
comp_transfer_2	2	"move the orange and the lemon from the plate to the bin"
comp_transfer_3	3	"move the orange, the lemon and the lime from the plate to the bin"
Split destination		
comp_route_2	2	"put the lemon on the plate and put the orange in the bin"
comp_route_2_binfirst	2	"put the orange in the bin and put the lemon on the plate"
comp_split_plate_2	2	"put the avocado on the plate and put the orange in the bin"
comp_route_3	3	"put the lemon and the lime on the plate and put the orange in the bin"
comp_split_plate_3	3	"put the avocado and the banana on the plate and put the orange in the bin"
comp_route_4	4	"put the lemon and the lime on the plate and put the orange and the avocado in the bin"
comp_split_plate_4	4	"put the avocado and the banana on the plate and put the orange and the lime in the bin"
comp_route_5	5	"put the lemon, the lime and the banana on the plate and put the orange and the avocado in the bin"
comp_split_plate_5	5	"put the avocado and the banana on the plate and put the orange, the onion and the lychee in the bin"
comp_route_6	6	"put the lemon, the lime and the banana on the plate and put the orange, the avocado and the onion in the bin"
comp_split_plate_6	6	"put the avocado, the banana and the lemon on the plate and put the orange, the lime and the onion in the bin"
comp_route_7	7	"put the banana, the lemon and the lime on the plate and put the orange, the avocado, the onion and the lychee in the bin"
comp_split_plate_7	7	"put the avocado, the banana and the lemon on the plate and put the orange, the lime, the onion and the lychee in the bin"
comp_route_8	8	"put the banana, the lemon and the lime on the plate and put the orange, the avocado, the onion, the lychee and the pumpkin in the bin"
comp_split_plate_8	8	"put the avocado, the banana, the lemon and the lime on the plate and put the orange, the onion, the lychee and the pumpkin in the bin"
comp_route_9	9	"put the banana, the lemon and the lime on the plate and put both oranges, the avocado, the onion, the lychee and the pumpkin in the bin"

RoboLab-Long, continued

Task	Subgoals	Instruction
comp_route_10	10	"put the banana, the lemon and the lime on the plate and put both oranges, the avocado, the onion, the lychee, the pumpkin and the pomegranate in the bin"
Relay		
comp_relay_2	4	"first put the lemon on the plate, then put the lemon in the bin, then put the lime on the plate, then put the lime in the bin"
comp_relay_3	6	"first put the lemon on the plate, then put the lemon in the bin, then put the lime on the plate, then put the lime in the bin, then put the banana on the plate, then put the banana in the bin"
comp_relay_4	8	"first put the lemon on the plate, then put the lemon in the bin, then put the lime on the plate, then put the lime in the bin, then put the banana on the plate, then put the banana in the bin, then put the orange on the plate, then put the orange in the bin"
comp_relay_5	10	"first put the lemon on the plate, then put the lemon in the bin, then put the lime on the plate, then put the lime in the bin, then put the banana on the plate, then put the banana in the bin, then put the orange on the plate, then put the orange in the bin, then put the avocado on the plate, then put the avocado in the bin"
Revisit		
comp_revisit_1	3	"first put the lemon in the bin, then put the lemon on the plate, then put the lemon in the bin"
comp_revisit_2	6	"first put the lemon in the bin, then put the lemon on the plate, then put the lemon in the bin, then put the lime in the bin, then put the lime on the plate, then put the lime in the bin"
comp_revisit_3	9	"first put the lemon in the bin, then put the lemon on the plate, then put the lemon in the bin, then put the lime in the bin, then put the lime on the plate, then put the lime in the bin, then put the banana in the bin, then put the banana on the plate, then put the banana in the bin"
Exclusion		
comp_bin_2_distractor	2	"put everything except the red block in the bin"
comp_bin_4_distractor	4	"put everything except the red block and the blue block in the bin"
Ordered		
comp_ordered_bin_2	2	"first put the banana in the bin, then put the orange in the bin"
comp_ordered_bin_3	3	"first put the banana in the bin, then put the lemon in the bin, then put the orange in the bin"
comp_ordered_bin_4	4	"first put the orange in the bin, then put the lemon in the bin, then put the lime in the bin, then put the banana in the bin"
comp_ordered_bin_5	5	"first put the avocado in the bin, then put the orange in the bin, then put the lemon in the bin, then put the lime in the bin, then put the banana in the bin"
comp_ordered_bin_6	6	"first put the avocado in the bin, then put the orange in the bin, then put the lemon in the bin, then put the lime in the bin, then put the onion in the bin, then put the banana in the bin"
comp_ordered_bin_7	7	"first put the avocado in the bin, then put the orange in the bin, then put the lemon in the bin, then put the lime in the bin, then put the onion in the bin, then put the banana in the bin, then put the lychee in the bin"
comp_ordered_bin_8	8	"first put the orange in the bin, then put the pumpkin in the bin, then put the lychee in the bin, then put the lemon in the bin, then put the lime in the bin, then put the banana in the bin, then put the onion in the bin, then put the avocado in the bin"
comp_ordered_bin_9	9	"first put the onion in the bin, then put the orange in the bin, then put the pumpkin in the bin, then put the lychee in the bin, then put the lemon in the bin, then put the lime in the bin, then put the banana in the bin, then put the avocado in the bin, then put the orange in the bin"
comp_ordered_bin_10	10	"first put the pumpkin in the bin, then put the onion in the bin, then put the lime in the bin, then put the avocado in the bin, then put the lemon in the bin, then put the pomegranate in the bin, then put the lychee in the bin, then put the banana in the bin, then put the orange in the bin, then put the orange in the bin"
Duplicates		
comp_identical_2	2	"put both oranges in the bin"
comp_identical_4	4	"put the four limes in the bin"

E REAL-ROBOT RESULTS PER TASK CATEGORY

Table 10 breaks the real-robot results of Section 5.3 down by task category.

Table 10: **Real-robot evaluation per task category.** Subgoal progress (%) and task success rate (%) on the WidowX (Bridge baseline) and Franka (DROID baseline) physical platforms, 12 episodes per task and system (3 layouts, 4 trials each). PSP-Agent ran a month after the other two systems on the same tasks, with layouts restored from the same reference images. On the Franka single-destination task the low-level policy and the Proprioceptive verifier ran with 150 steps per subgoal (950 steps, with 200 more for the first subgoal) and PSP-Agent with 300 (1,700 steps); every PSP-Agent success there placed its fourth object after step 1,200. PSP-Agent ran that task twice, with no reason for the repeat recorded, and its first run (0/12 successes, 38.3% progress) is not included. On the real robots the Proprioceptive verifier used Gemini Robotics-ER 2 for planning and verification, and PSP-Agent Gemini 3.7 Flash. †Run with PSP-Agent only, and not part of the *All multi-subgoal* row.

			Low-level policy		Proprioceptive verifier		PSP-Agent	
Task category	Subgoals	Episodes	Progress	Success	Progress	Success	Progress	Success
Embodiment A (WidowX, Bridge baseline)								
1 subgoal, control	1	24	66.7	66.7	75.0	75.0	100.0	**100.0**
Single destination	2–3	24	18.3	0.0	56.7	33.3	95.0	**87.5**
Split destination	2–3	24	21.7	0.0	63.3	29.2	86.7	**83.3**
Exclusion	2	12	8.3	0.0	62.5	58.3	95.8	**91.7**
Ordered	2	12	29.2	8.3	50.0	33.3	70.8	**41.7**
Duplicates	2	12	8.3	0.0	37.5	0.0	70.8	**58.3**
All multi-subgoal	2–3	84	18.2	1.2	56.2	31.0	86.5	**76.2**
Embodiment B (Franka, DROID baseline)								
Single destination	5	12	65.0	0.0	65.0	8.3	56.7	**25.0**
Exclusion	2	12	0.0	0.0	91.7	83.3	100.0	**100.0**
Ordered	3	12	38.9	16.7	88.9	83.3	100.0	**100.0**
All multi-subgoal	2–5	36	44.2	5.6	77.5	58.3	78.3	**75.0**
Split destination†	3	12	–	–	–	–	97.2	91.7
Relay†	4	12	–	–	–	–	93.8	91.7

F Results per Suite and System

The main text combines some results over the three suites or draws only some of the systems. This appendix gives every one of them per suite and for all ten systems, over the same episodes. Table 11 and Table 13 are the numbers behind Figure 4, Table 12 splits the task categories of Table 3 by suite, and Table 14 splits the offline VLM benchmark of Table 4 by the suite each scene comes from, without intervals, since each suite holds about a third of the samples. Failure causes per suite are in Appendix H and efficiency per suite in Appendix G.

Uncertainty over tasks. The tasks are the benchmark, so the intervals of the tables cover the randomness of the episodes on these tasks. For a fixed set of tasks the Wilson interval is conservative, because the variance within tasks is smaller than the variance of all episodes taken as one sample. A claim that one system is better than another should also hold across tasks. Since all systems run the same tasks, we test each such difference paired: tasks are resampled within each suite (10,000 bootstrap draws) and both systems are scored on the same resampled tasks, so the difficulty of a task counts for both. The overall differences the main text draws, in points with their 95% intervals, are GPT-Policy over PSP-Agent 21.8 [18.1, 25.8], PSP-Agent over the privileged verifier 10.5 [7.4, 13.7], the privileged verifier over the *Proprioceptive verifier* 4.5 [2.6, 6.6] and over VoLo 4.2 [2.3, 6.3], the *Proprioceptive verifier* over the *Proprioceptive trigger* 3.4 [1.4, 5.3] and over the low-level policy 14.0 [9.8, 18.7]. VoLo and the *Proprioceptive verifier* (0.3 [−1.8, 2.6]), CaP-X and the *Proprioceptive verifier* (0.0 [−3.8, 3.6]), and the low-level policy and TiPToP (1.6 [−1.3, 4.7]) are not separated.

Offline VLM benchmark. Plan decomposition is scored on 145 scenes, the reset frame and instruction of every task; a plan is an exact match when its object and destination pairs are exactly those of the task's ground-truth subgoals, with their multiplicity, in any order. Requiring their order on the 47 ordered, relay and revisit tasks changes no score in Table 4. Verification, state tracking and next-subgoal selection use 562 frames recorded in rollouts of 143 of these tasks, mostly of the privileged verifier, CaP-X and TiPToP, labelled by the simulator's per-subgoal predicates. The TP and TN rates are the shares of finished and unfinished subgoals judged correctly; state tracking asks about every subgoal of a frame and counts the frame as correct only when all answers are; a next subgoal is valid when it names a subgoal that is still unfinished, and on those 47 tasks the first unfinished one. Table 14 splits these scores by suite.

Table 11: **Task success rate (%) per suite, system and number of subgoals** (Figure 4, top).

Tasks	Low-level policy	Periodic trigger	Proprioceptive trigger	Proprioceptive verifier	VoLo	CaP-X	TiPToP	PSP-Agent	GPT-Policy	Privileged verifier	Episodes
SimplerEnv-Long											
1 subgoal	89.6	91.7	91.7	89.6	**93.8**	87.5	72.9	89.6	85.4	85.4	48
2 subgoals	28.1	56.2	62.5	68.8	64.1	67.2	15.1	74.0	**87.0**	74.5	192
3 subgoals	6.0	29.6	36.1	40.7	49.5	47.2	9.7	57.4	**75.0**	50.5	216
4 subgoals	1.8	4.8	11.9	19.0	18.5	26.8	1.2	36.3	**63.7**	22.6	168
5 subgoals	0.0	0.6	7.7	10.7	6.0	11.3	0.0	25.6	**48.8**	11.9	168
6 subgoals	0.0	0.0	2.1	6.2	5.2	13.5	0.0	31.2	**58.3**	17.7	96
7 subgoals	0.0	0.0	0.8	3.3	5.8	8.3	0.0	16.7	**47.5**	3.3	120
8 subgoals	0.0	0.0	0.0	1.0	2.1	5.2	0.0	4.2	**39.6**	6.2	96
9 subgoals	0.0	0.0	1.0	0.0	1.0	5.2	0.0	10.4	**39.6**	3.1	96
10 subgoals	0.0	0.0	0.0	2.1	0.0	2.1	0.0	4.2	**35.4**	1.0	96
All tasks	8.7	17.4	21.5	25.2	25.5	28.7	6.7	37.1	**60.3**	29.5	1,296
RobotArena-∞-Long											
1 subgoal	71.9	62.5	56.2	62.5	50.0	40.6	21.9	59.4	**90.6**	68.8	32
2 subgoals	13.4	25.9	31.2	42.0	40.2	24.1	19.6	63.4	**82.1**	40.2	112
3 subgoals	3.8	8.8	8.8	22.5	16.2	16.2	16.2	51.2	**76.2**	23.8	80
4 subgoals	0.0	2.5	2.5	15.0	12.5	12.5	0.0	25.0	**62.5**	20.0	40
5 subgoals	0.0	0.0	0.0	0.0	4.2	0.0	0.0	20.8	**54.2**	8.3	24
6 subgoals	0.0	0.0	0.0	0.0	0.0	7.5	0.0	7.5	**27.5**	0.0	40
7 subgoals	0.0	0.0	0.0	0.0	0.0	0.0	0.0	6.2	**56.2**	0.0	16
8 subgoals	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**31.2**	0.0	16
9 subgoals	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**12.5**	0.0	8
10 subgoals	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**62.5**	0.0	8
All tasks	10.9	15.2	16.2	24.2	21.3	16.2	11.2	39.9	**66.8**	25.5	376
RoboLab-Long											
1 subgoal	**100.0**	**100.0**	**100.0**	87.5	93.8	81.2	87.5	93.8	**100.0**	87.5	16
2 subgoals	46.9	68.8	73.4	56.2	68.8	48.4	39.1	78.1	**92.2**	78.1	64
3 subgoals	22.5	20.0	30.0	37.5	32.5	15.0	30.0	50.0	**85.0**	40.0	40
4 subgoals	14.6	22.9	29.2	27.1	35.4	33.3	16.7	**77.1**	72.9	43.8	48
5 subgoals	0.0	0.0	12.5	0.0	0.0	0.0	0.0	45.8	**70.8**	8.3	24
6 subgoals	10.4	14.6	14.6	16.7	18.8	6.2	2.1	39.6	**47.9**	18.8	48
7 subgoals	0.0	0.0	0.0	0.0	0.0	0.0	4.2	**45.8**	37.5	0.0	24
8 subgoals	0.0	6.2	0.0	3.1	6.2	6.2	0.0	28.1	**37.5**	15.6	32
9 subgoals	0.0	0.0	0.0	4.2	4.2	0.0	0.0	**8.3**	**8.3**	4.2	24
10 subgoals	6.2	6.2	0.0	6.2	3.1	6.2	0.0	21.9	**43.8**	9.4	32
All tasks	19.6	25.6	28.1	25.6	29.0	20.7	17.3	51.4	**62.8**	34.4	352

Table 12: **Task success rate (%) per suite, system and task category.** Table 3 combines the three suites; here each suite is shown on its own.

Tasks	Low-level policy	Periodic trigger	Proprioceptive trigger	Proprioceptive verifier	VoLo	CaP-X	TiPToP	PSP-Agent	GPT-Policy	Privileged verifier	Episodes
SimplerEnv-Long											
Single destination	9.6	18.8	22.7	26.9	27.1	23.8	11.7	41.2	**54.4**	30.0	480
Transfer	9.2	8.3	7.5	9.2	22.5	44.2	1.7	25.8	**67.5**	21.7	120
Split destination	12.5	15.3	21.5	23.6	25.7	21.5	8.3	41.7	**61.1**	29.2	144
Relay	3.2	13.0	19.0	25.5	18.1	39.8	0.0	23.1	**74.5**	28.2	216
Revisit	0.0	0.0	0.0	0.0	1.0	9.4	0.0	8.3	**39.6**	1.0	96
Exclusion	0.0	35.4	46.9	46.9	44.8	35.4	6.2	55.2	**70.8**	47.9	96
Ordered	39.6	64.6	62.5	72.9	81.2	62.5	22.9	**87.5**	81.2	85.4	48
Duplicates	12.5	10.4	14.6	17.7	15.6	15.6	0.0	40.6	**47.9**	21.9	96
All tasks	8.7	17.4	21.5	25.2	25.5	28.7	6.7	37.1	**60.3**	29.5	1,296
RobotArena-∞-Long											
Single destination	15.8	23.7	19.7	34.2	32.2	13.8	9.2	46.1	**79.6**	36.2	152
Transfer	0.0	0.0	0.0	0.0	0.0	50.0	25.0	**100.0**	62.5	0.0	8
Split destination	0.0	12.5	0.0	18.8	25.0	25.0	31.2	**100.0**	56.2	18.8	16
Relay	0.0	3.6	1.8	5.4	5.4	17.9	1.8	19.6	**55.4**	10.7	56
Revisit	0.0	0.0	0.0	0.0	0.0	12.5	0.0	4.2	**58.3**	2.1	48
Exclusion	1.8	16.1	26.8	30.4	8.9	8.9	10.7	26.8	**32.1**	26.8	56
Ordered	20.8	4.2	20.8	33.3	37.5	29.2	16.7	66.7	**95.8**	33.3	24
Duplicates	68.8	43.8	62.5	50.0	62.5	25.0	62.5	75.0	**100.0**	50.0	16
All tasks	10.9	15.2	16.2	24.2	21.3	16.2	11.2	39.9	**66.8**	25.5	376
RoboLab-Long											
Single destination	75.0	80.0	72.5	67.5	72.5	57.5	42.5	70.0	**82.5**	70.0	40
Transfer	**81.2**	68.8	75.0	68.8	75.0	37.5	68.8	**81.2**	**81.2**	75.0	16
Split destination	2.2	7.4	15.4	10.3	14.7	9.6	14.0	54.4	**60.3**	19.9	136
Relay	0.0	9.4	15.6	21.9	21.9	6.2	0.0	40.6	**81.2**	50.0	32
Revisit	0.0	8.3	8.3	20.8	20.8	0.0	0.0	0.0	**29.2**	25.0	24
Exclusion	0.0	56.2	43.8	25.0	43.8	81.2	37.5	87.5	**93.8**	62.5	16
Ordered	11.1	12.5	9.7	11.1	9.7	2.8	4.2	31.9	**44.4**	11.1	72
Duplicates	93.8	87.5	**100.0**	87.5	93.8	87.5	31.2	**100.0**	81.2	87.5	16
All tasks	19.6	25.6	28.1	25.6	29.0	20.7	17.3	51.4	**62.8**	34.4	352

Table 13: **Subgoal progress (%) per suite, system and number of subgoals** (Figure 4, bottom).

Tasks	Low-level policy	Periodic trigger	Proprioceptive trigger	Proprioceptive verifier	VoLo	CaP-X	TiPToP	PSP-Agent	GPT-Policy	Privileged verifier	Episodes
SimplerEnv-Long											
1 subgoal	89.6	91.7	91.7	89.6	**93.8**	87.5	72.9	89.6	85.4	85.4	48
2 subgoals	49.2	73.7	77.9	81.8	76.0	78.9	29.9	83.1	**92.7**	83.6	192
3 subgoals	37.2	62.0	65.9	66.0	69.0	69.8	29.2	76.4	**89.8**	70.2	216
4 subgoals	29.9	43.3	48.2	54.2	54.9	56.0	11.3	67.3	**87.5**	54.5	168
5 subgoals	30.7	38.2	46.9	51.2	46.3	43.2	1.8	61.5	**84.8**	49.0	168
6 subgoals	25.3	41.0	48.1	50.2	47.2	46.4	0.0	66.8	**88.2**	59.5	96
7 subgoals	25.2	31.4	42.3	47.1	46.7	41.1	1.9	55.1	**83.7**	50.4	120
8 subgoals	17.2	31.1	37.6	46.2	38.9	32.8	2.2	46.1	**78.3**	44.1	96
9 subgoals	14.8	34.5	35.0	42.2	39.1	39.5	0.9	48.6	**78.6**	39.2	96
10 subgoals	12.5	26.8	32.6	39.7	35.9	29.9	0.0	41.2	**76.0**	43.8	96
All tasks	25.3	39.9	45.8	51.0	48.3	45.9	7.1	58.2	**83.3**	52.4	1,296
RobotArena-∞-Long											
1 subgoal	71.9	62.5	56.2	62.5	50.0	40.6	21.9	59.4	**90.6**	68.8	32
2 subgoals	28.1	51.3	51.8	57.1	56.2	35.3	32.6	75.4	**88.8**	53.1	112
3 subgoals	22.1	40.4	44.2	42.5	39.2	42.9	30.8	74.6	**91.7**	47.9	80
4 subgoals	6.9	23.1	25.0	38.1	37.5	28.1	5.0	49.4	**86.9**	40.0	40
5 subgoals	8.3	29.2	30.0	29.2	28.3	15.0	3.3	51.7	**84.2**	30.0	24
6 subgoals	7.1	21.2	22.1	27.5	24.2	20.4	1.7	43.8	**73.8**	18.8	40
7 subgoals	3.6	11.6	11.6	19.6	13.4	33.9	3.6	32.1	**80.4**	17.9	16
8 subgoals	2.3	9.4	9.4	12.5	14.1	27.3	0.0	20.3	**75.0**	15.6	16
9 subgoals	4.2	16.7	8.3	19.4	18.1	8.3	5.6	19.4	**73.6**	16.7	8
10 subgoals	3.8	8.8	8.8	13.8	17.5	36.2	0.0	22.5	**87.5**	8.8	8
All tasks	13.5	28.3	28.9	33.7	31.8	29.5	12.6	50.2	**83.4**	32.7	376
RoboLab-Long											
1 subgoal	**100.0**	**100.0**	**100.0**	87.5	93.8	81.2	87.5	93.8	**100.0**	87.5	16
2 subgoals	68.8	77.3	84.4	73.4	80.5	68.8	61.7	90.6	**96.1**	87.5	64
3 subgoals	63.3	54.2	70.0	64.2	70.0	44.2	58.3	76.7	**91.7**	72.5	40
4 subgoals	66.7	65.6	69.3	65.1	73.4	61.5	46.4	**92.7**	86.5	77.6	48
5 subgoals	63.3	49.2	60.8	45.0	50.8	47.5	52.5	86.7	**90.8**	60.8	24
6 subgoals	49.7	50.0	57.3	59.4	67.7	37.5	18.8	76.4	**80.2**	55.2	48
7 subgoals	44.6	52.4	50.6	51.2	52.4	24.4	17.3	**87.5**	78.6	49.4	24
8 subgoals	48.0	56.2	51.6	45.3	55.9	32.8	0.0	**83.6**	73.4	58.6	32
9 subgoals	53.2	56.0	56.9	63.4	56.0	37.5	5.1	**67.1**	47.2	53.2	24
10 subgoals	60.3	56.2	48.8	46.9	56.2	40.6	0.0	80.6	**80.9**	67.5	32
All tasks	56.6	57.1	58.9	56.1	62.0	42.4	22.4	**81.6**	78.7	63.5	352

Table 14: **Offline evaluation of VLMs per suite.** Bold marks the best model within each suite.

	Planning	Subgoal completion		State tracking	Next subgoal
Model	exact match (%)	TP rate (%)	TN rate (%)	all correct (%)	valid (%)
SimplerEnv-Long scenes					
Qwen3-VL-2B	61.1	52.1	40.4	10.6	39.9
Qwen3-VL-8B	92.6	58.5	86.5	58.6	65.2
Qwen3.5-9B	96.3	45.7	**97.1**	13.6	60.6
Qwen3.8-27B	92.6	**63.8**	92.3	**71.2**	79.3
Gemini 3.1 Pro	96.3	47.9	**97.1**	66.2	69.2
Gemini 3.7 Flash	96.3	52.1	96.2	67.2	77.8
Gemini Robotics-ER 2	96.3	43.6	**97.1**	65.2	71.7
GPT-5.6 Sol	94.4	55.3	89.4	66.7	83.3
GPT-6 Astra	**98.1**	55.3	90.4	64.6	**84.8**
Claude Opus 5	96.3	60.6	90.4	69.7	83.8
RobotArena-∞-Long scenes					
Qwen3-VL-2B	80.9	71.2	40.9	8.5	60.1
Qwen3-VL-8B	85.1	80.8	73.9	44.1	70.7
Qwen3.5-9B	80.9	42.5	94.8	12.2	70.2
Qwen3.8-27B	83.0	**82.2**	77.4	50.0	75.5
Gemini 3.1 Pro	74.5	42.5	92.2	64.9	71.3
Gemini 3.7 Flash	85.1	47.9	95.7	**70.2**	79.3
Gemini Robotics-ER 2	76.6	42.5	**97.4**	68.6	72.9
GPT-5.6 Sol	**87.2**	76.7	78.3	55.3	76.6
GPT-6 Astra	80.9	71.2	82.6	54.3	**80.3**
Claude Opus 5	76.6	80.8	84.3	58.0	71.8
RoboLab-Long scenes					
Qwen3-VL-2B	86.4	86.1	20.2	8.5	79.3
Qwen3-VL-8B	95.5	70.8	79.8	39.8	82.2
Qwen3.5-9B	81.8	44.4	**99.0**	9.1	58.6
Qwen3.8-27B	**97.7**	86.1	85.6	52.3	83.3
Gemini 3.1 Pro	**97.7**	76.4	**99.0**	**75.6**	86.2
Gemini 3.7 Flash	**97.7**	80.6	98.1	**75.6**	87.9
Gemini Robotics-ER 2	**97.7**	79.2	**99.0**	**75.6**	86.2
GPT-5.6 Sol	**97.7**	**91.7**	92.3	46.0	86.2
GPT-6 Astra	95.5	**91.7**	86.5	39.2	**91.4**
Claude Opus 5	**97.7**	66.7	89.4	54.5	83.9

G EFFICIENCY AND TOKEN USAGE

Table 15: **Efficiency of every system, per suite.** Subgoals completed are shares of all subgoals, measured when half of the simulated time budget has elapsed and at its end (the latter is the subgoal progress of Figure 4).

	Low-level policy	Periodic trigger	Proprioceptive trigger	Proprioceptive verifier	VoLo	CaP-X	TiPToP	PSP-Agent	GPT-Policy	Privileged verifier
VLM calls per episode										
SimplerEnv-Long	0.0	0.0	0.0	12.7	24.8	3.8	1.0	18.0	50.1	0.0
RobotArena-∞-Long	0.0	0.0	0.0	9.3	26.4	5.6	1.0	20.2	41.7	0.0
RoboLab-Long	0.0	0.0	0.0	9.3	19.2	6.3	1.0	31.8	70.8	0.0
VLM calls per completed subgoal										
SimplerEnv-Long	0.0	0.0	0.0	4.9	10.1	1.6	2.8	6.1	11.8	0.0
RobotArena-∞-Long	0.0	0.0	0.0	7.3	22.2	5.1	2.1	10.7	13.3	0.0
RoboLab-Long	0.0	0.0	0.0	3.2	6.0	2.9	0.9	7.5	17.4	0.0
Input tokens per episode (thousands)										
SimplerEnv-Long	0.0	0.0	0.0	17.2	70.5	7.5	2.0	27.3	357.8	0.0
RobotArena-∞-Long	0.0	0.0	0.0	12.6	75.0	12.9	2.0	31.2	293.5	0.0
RoboLab-Long	0.0	0.0	0.0	23.6	91.6	15.3	2.1	48.5	508.8	0.0
Generated tokens per episode (thousands)										
SimplerEnv-Long	0.0	0.0	0.0	2.7	7.4	3.1	0.5	10.4	5.7	0.0
RobotArena-∞-Long	0.0	0.0	0.0	1.8	6.6	4.6	0.4	10.1	4.9	0.0
RoboLab-Long	0.0	0.0	0.0	2.5	7.2	5.1	0.4	16.4	8.3	0.0
API cost per episode (USD)										
SimplerEnv-Long	0.000	0.000	0.000	0.046	0.161	0.034	0.007	0.119	3.861	0.000
RobotArena-∞-Long	0.000	0.000	0.000	0.032	0.162	0.054	0.006	0.123	3.177	0.000
RoboLab-Long	0.000	0.000	0.000	0.054	0.191	0.061	0.006	0.196	5.501	0.000
API cost per successful episode (USD)										
SimplerEnv-Long	0.00	0.00	0.00	0.18	0.63	0.12	0.10	0.32	6.40	0.00
RobotArena-∞-Long	0.00	0.00	0.00	0.13	0.76	0.33	0.05	0.31	4.76	0.00
RoboLab-Long	0.00	0.00	0.00	0.21	0.66	0.30	0.04	0.38	8.76	0.00
Subgoals completed by half of the time budget (%)										
SimplerEnv-Long	21.3	37.8	42.3	44.1	40.3	38.4	6.0	42.9	60.9	47.8
RobotArena-∞-Long	10.5	25.4	24.4	24.6	26.7	19.4	12.6	37.9	60.2	24.4
RoboLab-Long	53.1	52.6	50.2	48.4	53.6	41.6	22.3	78.4	78.1	55.9
Subgoals completed by the end of the budget (%)										
SimplerEnv-Long	25.3	39.9	45.8	51.0	48.3	45.9	7.1	58.2	83.3	52.4
RobotArena-∞-Long	13.5	28.3	28.9	33.7	31.8	29.5	12.6	50.2	83.4	32.7
RoboLab-Long	56.6	57.1	58.9	56.1	62.0	42.4	22.4	81.6	78.7	63.3

Section 5.1 compares systems by two costs that do not depend on the hardware a run used.

Robot time. Every episode has a simulated time budget (Section 3). The simulation advances only when a system returns an action, so time spent computing, for example waiting for a VLM, does not use the budget. We record which subgoals are satisfied at every step and report the share of all subgoals completed when a given share of the budget has elapsed. A successful episode ends at success, and its subgoals count as completed for the rest of the budget. At the end of the budget this share is exactly the subgoal progress of Figure 4, which we check for every evaluated unit that recorded its episode log (all but one RoboLab-Long task of the privileged verifier, whose 8 episodes are therefore absent from Tables 15, 17 and 18). Successful episodes of the learned-policy systems end, on average, after 36–45% of their budget.

VLM calls. We count the successful requests each episode sends to the VLM endpoint. Each request is recorded twice, in the evaluation log and in the usage log of the VLM client, and neither record is complete on its own. We match the two records by time and count a request found in either of them once. Requests refused by the endpoint's rate limit and then retried are not counted.

Why not wall-clock time. Wall-clock time is logged, but it does not compare systems fairly. The same $\pi_{0.5}$ policy with no VLM calls takes 51–56 ms per step on an RTX PRO 6000 but 130–177 ms on an A100, several runs share one GPU, and VLM requests wait in queues shared with other jobs. No two systems ran on identical hardware.

Tokens and API cost. Calls of different systems send different amounts of text and images and generate different amounts of text, so we also report the tokens of the counted calls, from each

provider's own usage report, and their API cost at the providers' standard list prices on 25 September 2026: $1.50 per million input tokens and $7.50 per million generated tokens for Gemini 3.7 Flash, and $10 and $50 for GPT-6 Astra (Table 15). Generated tokens include reasoning. The cost covers only the hosted VLM, not the GPU time of the low-level policy, of TiPToP's planner or of the simulators. Over all suites together, an episode costs $0.007 with TiPToP, $0.043 with CaP-X, $0.045 with the *Proprioceptive verifier*, $0.13 with PSP-Agent, $0.17 with VoLo and $4.02 with GPT-Policy. GPT-Policy succeeds most often, but costs $6.49 per successful episode ($8,134 for one full evaluation) against $0.33 for PSP-Agent, and most of its cost comes from input tokens. The usage reports do not say how much input the provider served from its cache, so GPT-Policy's cost is an upper bound.

Table 16: **Tokens per query of the offline benchmark.** Mean input and generated tokens (reasoning included) per scored query, from each API's own usage counts. A query is one scene for plan decomposition and one frame for the other capabilities; state tracking asks about each subgoal of the frame separately. Input counts follow each provider's own tokenizer and image accounting, so they compare within a provider only. The two local Qwen3-VL models report no usage.

	Plan decomposition		Subgoal completion		State tracking		Next subgoal	
Model	input	generated	input	generated	input	generated	input	generated
Qwen3-VL-2B	–	–	–	–	–	–	–	–
Qwen3-VL-8B	–	–	–	–	–	–	–	–
Qwen3.5-9B	655	2,951	875	2,404	3,445	10,325	653	3,339
Qwen3.8-27B	690	943	617	967	2,321	3,790	690	1,189
Gemini 3.1 Pro	1,474	663	1,727	231	6,965	970	1,471	529
Gemini 3.7 Flash	1,475	431	1,727	195	6,960	743	1,471	290
Gemini Robotics-ER 2	1,476	518	1,729	219	6,987	886	1,473	424
GPT-5.6 Sol	682	147	610	151	2,270	591	674	138
GPT-6 Astra	682	91	610	110	2,270	475	675	53
Claude Opus 5	903	119	784	253	2,930	884	885	133

Tokens of the offline benchmark. Table 16 gives the tokens each model uses on the offline benchmark of Section 5.2. The counts come from each API's own usage report, matched to the scored queries of every run (at least 98.6% of the queries of every model and capability). Each model ran the benchmark once. Next-subgoal queries were sampled at temperature 0.3, and Claude Opus 5 and GPT-6 Astra ran at the provider's default temperature, because their endpoint does not accept one.

H Diagnostic Failure Analysis

Section 5.2 summarizes where subgoals are lost. This appendix defines every cause and reports the causes for each benchmark separately.

Subgoal causes. We reconstruct each episode from its logged object poses, per-subgoal success predicates, gripper state, and orchestrator events, and assign every subgoal exactly one outcome. A subgoal that counts toward the subgoal progress is *completed*; every other subgoal receives the first of the following twelve causes that applies, grouped into five stages.

- *Undone later*: the subgoal was done at some step but not at the end. The robot *picked it up again* when the gripper took the object again; the object was *knocked off* when a collision moved it while nothing held it, and *slid or rolled off* otherwise.
- Otherwise, if its object was ever held, the last hold decides. A hold credited to another of the object's subgoals (relay and revisit tasks) belongs to that subgoal and is skipped. The robot *lost grip before lifting* when the object left the gripper before rising 3 cm. The object was *dropped on the way* when it was lifted and released away from every destination, and *timed out mid-carry* when time ran out while the robot still held it. An object set down within its own destination's footprint (plus 2 cm) without being counted is *slightly off target* (*Placing*); one set down on a different destination was *put in the wrong place*.

- Otherwise the object was never held. For orchestrators that follow a fixed plan in order, the *plan never reached* the subgoal when it was never issued. The robot *missed the object* (*Grasping*) when the gripper closed next to it, nearer to it than to any other object, and held nothing. For TiPToP only, *no plan was found* when its planner reported failure. Every remaining subgoal had its *object never touched*. A later leg of an object whose earlier leg was never done counts as never touched, since every hold of that object served the earlier leg.

The completed count reproduces the reported subgoal progress exactly for every evaluated unit with an episode log, so the causes partition exactly the subgoals Figure 4 counts as lost.

How the figures are computed. Every share is a count of subgoals divided by a count of subgoals. Within one suite, a system's bar (Figure 8, top) divides the subgoals it lost to each cause by all subgoals of all its episodes, over every task of the suite. A point at N subgoals (Figure 8, bottom) combines the episodes of all ten systems on the suite's N-subgoal tasks. Figure 6 shows the unweighted mean of the three per-suite values, because SimplerEnv-Long holds more than three times as many subgoals as either other suite and would otherwise dominate. Table 17 lists the per-suite values. Both figures and Table 17 draw every lost subgoal once: as an orchestrator decision when one made the loss final (below), and otherwise by its cause. What grows most with horizon on every suite is orchestrator decisions and subgoals never started (Figure 8, bottom).

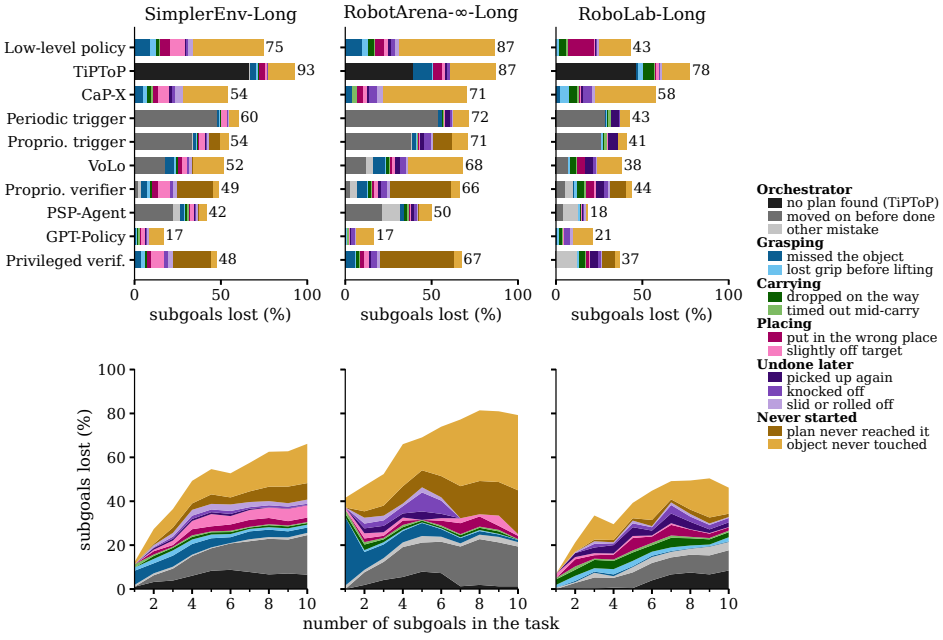

Figure 8: **Why subgoals fail, per benchmark.** Share of each suite's subgoals not completed, split by cause. Top: one bar per system, over all tasks of the suite. Bottom: all ten systems together, by number of subgoals in the task; on RobotArena-∞-Long each length from 7 up rests on one or two tasks. Systems keep the same order in every panel.

Orchestrator decisions. The causes above describe what happened to the object. Table 18 adds the orchestrator decision that made a loss final, from the logged subgoal switches, VLM verdicts, plans and tool calls. The simulator's own success predicate at the moment of each switch tells whether the subgoal left behind was satisfied. VoLo's plan steps and PSP-Agent's goals are free text, which we match to the ground-truth subgoals by their object and destination; the match covers at least 98.6% of the completed subgoals of every suite and system, which a plan must have named. A lost subgoal gets the first decision that applies: its orchestrator's plan never named it, TiPToP's planner found no plan, PSP-Agent had stopped making tool calls, the orchestrator left it unsatisfied after judging it done (*moved on before done*) or after a *not done* verdict or a stall (*deferred*) and never returned,

Table 17: **Where each system loses its subgoals**, as a share of all its subgoals (per suite, over all its tasks), split exactly as Figure 8 draws it. *Lost*: not completed at the end of the episode (one minus the subgoal progress). *Orch.*: lost to an orchestrator mistake (Table 18); every other column counts only the remaining losses. The other columns abbreviate the causes defined above.

			Grasping		Carrying		Placing		Undone later			Never started	
System	Lost	Orch.	Missed	Lost grip	Dropped	Timed out	Wrong place	Off target	Picked up	Knocked off	Slid off	Not reached	Untouched
SimplerEnv-Long													
Low-level policy	74.7	–	9.1	3.3	1.7	0.7	6.1	8.3	0.6	0.9	3.3	0.0	40.6
TiPToP	92.9	66.6	4.0	1.1	0.7	0.0	3.4	1.3	0.1	0.2	0.3	0.0	15.2
CaP-X	54.1	–	5.0	2.7	1.9	1.1	3.2	6.4	1.5	2.0	4.5	0.0	26.0
Periodic trigger	60.1	47.8	1.4	0.5	0.3	0.2	0.2	2.9	0.5	0.3	0.7	0.0	5.2
Proprioceptive trigger	54.2	34.2	1.5	0.7	0.3	0.2	0.5	3.3	0.6	0.8	1.3	5.9	4.7
VoLo	51.7	18.0	4.9	1.6	0.6	0.4	2.0	2.8	0.3	1.0	2.1	0.0	17.9
Proprioceptive verifier	49.0	4.0	3.5	1.8	0.7	0.4	3.2	6.8	0.5	1.4	2.3	21.1	3.5
PSP-Agent	41.8	26.5	2.3	0.7	1.2	0.9	0.8	2.2	0.6	0.4	1.7	0.0	4.5
GPT-Policy	16.7	–	0.7	0.9	0.7	0.8	0.9	2.3	0.4	0.5	1.2	0.0	8.2
Privileged verifier	47.6	0.0	3.8	2.3	1.0	0.3	2.5	7.1	0.4	1.8	3.4	21.7	3.3
RobotArena-∞-Long													
Low-level policy	86.5	–	10.2	2.9	3.7	0.7	4.9	2.6	2.1	1.8	2.4	0.0	55.3
TiPToP	87.4	39.4	11.2	0.0	0.6	0.0	4.7	2.0	1.1	1.6	0.6	0.0	26.3
CaP-X	70.5	–	3.9	0.0	0.5	2.3	3.6	2.2	1.6	4.7	3.5	0.0	48.2
Periodic trigger	71.7	53.8	2.4	0.1	0.5	0.3	0.3	0.6	2.6	1.4	0.6	0.0	9.1
Proprioceptive trigger	71.1	38.6	2.7	0.2	0.2	0.4	0.5	0.6	2.8	3.7	1.1	11.2	9.1
VoLo	68.2	16.4	6.7	0.8	1.8	0.3	1.3	1.8	2.4	4.0	0.9	0.0	31.8
Proprioceptive verifier	66.3	7.0	5.8	0.6	1.3	0.6	1.6	2.3	1.4	3.8	1.7	35.2	4.9
PSP-Agent	49.8	31.8	2.6	0.0	1.6	1.3	0.5	0.9	1.2	2.1	0.8	0.0	7.1
GPT-Policy	16.6	–	0.8	0.1	0.4	1.0	0.4	0.2	0.5	2.5	0.8	0.0	10.0
Privileged verifier	67.3	0.0	5.8	0.6	1.7	0.4	2.4	2.7	1.6	3.6	1.5	42.6	4.5
RoboLab-Long													
Low-level policy	43.4	–	0.2	1.7	4.1	0.8	15.4	0.7	0.2	0.8	1.0	0.0	18.4
Low-level policy (Cosmos-3)	61.0	–	1.4	9.5	7.0	1.6	12.2	0.7	0.1	0.9	1.0	0.0	26.5
TiPToP	77.6	46.7	0.9	2.8	6.5	0.8	0.7	1.4	0.0	0.9	0.8	0.0	16.2
CaP-X	57.6	–	2.3	5.1	5.2	0.7	0.6	0.8	1.2	5.4	1.4	0.0	35.0
Periodic trigger	42.9	28.9	0.0	0.7	1.0	1.2	0.1	0.1	4.8	0.6	0.2	0.0	5.3
Proprioceptive trigger	41.1	26.3	0.1	0.7	1.2	1.3	0.4	0.1	5.5	0.5	0.2	0.0	4.7
VoLo	38.0	7.0	0.1	1.4	3.0	1.0	4.2	0.3	4.7	1.6	0.6	0.0	14.1
Proprioceptive verifier	43.9	10.3	0.1	1.9	4.2	1.2	4.8	0.6	5.1	2.4	0.8	9.2	3.3
PSP-Agent	18.4	13.3	0.0	0.6	0.8	0.1	0.1	0.2	0.9	0.9	0.6	0.0	1.0
GPT-Policy	21.3	–	0.6	1.3	2.0	0.2	0.2	0.4	0.3	3.6	1.2	0.0	11.4
Privileged verifier	36.7	12.4	0.0	1.0	3.3	1.1	1.3	0.4	5.1	1.7	0.5	7.6	2.4

or it was never issued because the orchestrator was still commanding a satisfied subgoal when the budget ran out (*stuck*). A completed subgoal the policy picked up again counts as *re-issued* when the orchestrator was commanding it again at the time, having issued it while it was satisfied or kept it in force for 45 steps after. The success predicate we read at each switch agrees with the one the orchestrator logged at every switch, and every early advance of the *Proprioceptive verifier* follows a VLM *done* verdict that the simulator contradicts. The privileged verifier's deferrals use privileged success and are not VLM errors. Figures 6 and 8 draw the two largest decisions, *no plan found* (TiPToP only) and *moved on before done*, and group the rest as *other mistake*.

Per-benchmark findings. The three suites fail in visibly different ways (Figure 8). Misplacement is one example: on the WidowX suites, 60.4% of misplaced objects lie on their correct destination without credit, while on RoboLab-Long $\pi_{0.5}$ mostly picks the wrong container.

- *SimplerEnv-Long.* TiPToP finds no plan for 66.6% of its subgoals.
- *RobotArena-∞-Long.* The hardest suite: every system except PSP-Agent and GPT-Policy loses at least 66.3% of its subgoals, against 49.8% for PSP-Agent and 16.6% for GPT-Policy. The low-level policy misses the grasp on 10.2% of its subgoals, and the *Proprioceptive verifier* and privileged verifier never command 38.9% because they spend the budget retrying earlier subgoals.
- *RoboLab-Long.* The Franka almost never misses a grasp (at most 2.3% of subgoals for any system), so its physical losses come after the grasp. Without an orchestrator, $\pi_{0.5}$ sets 15.4% of its subgoals on the wrong container, and under the low-level policy alone and every orchestrator together 90.8% of its misplaced objects land in the wrong container, against 13.3% for PSP-Agent, which locates each object with a VLM. Orchestrated configurations undo more of their completed subgoals, and the policy picks 61.8% of those back up, against 10.5% under the low-level policy alone. CaP-X never touches 35.0%, and Cosmos-3 loses its grip five times as often as $\pi_{0.5}$ (9.5% vs. 1.7%). PSP-Agent loses only 18.4%.

Table 18: **Subgoals lost to an orchestrator mistake**, as a share of all subgoals (per suite, over all its tasks), over the episodes of Table 17. Each lost subgoal is attributed to the orchestrator decision that made its loss final; the rest of *Lost* was lost while the robot carried out a correct command. A dash marks a system that logs no orchestrator decisions (low-level policy, CaP-X and GPT-Policy). The columns are the decisions defined above; *Stopped* (PSP-Agent only) means the agent made no tool call in the last 30% of the episode.

System	Lost	Orch.	Not planned	No plan	Stopped	Moved on	Deferred	Stuck	Re-issued
SimplerEnv-Long									
Low-level policy	74.7	–	–	–	–	–	–	–	–
TiPToP	92.9	66.6	0.0	66.6	0.0	0.0	0.0	0.0	0.0
CaP-X	54.1	–	–	–	–	–	–	–	–
Periodic trigger	60.1	47.8	0.0	0.0	0.0	47.8	0.0	0.0	0.0
Proprioceptive trigger	54.2	34.2	0.0	0.0	0.0	33.2	0.0	1.0	0.0
VoLo	51.7	18.0	0.5	0.0	0.0	17.5	0.0	0.0	0.1
Proprioceptive verifier	49.0	4.0	0.0	0.0	0.0	2.2	0.0	1.7	0.0
PSP-Agent	41.8	26.5	0.1	0.0	4.0	22.3	0.0	0.0	0.2
GPT-Policy	16.7	–	–	–	–	–	–	–	–
Privileged verifier	47.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
RobotArena-∞-Long									
Low-level policy	86.5	–	–	–	–	–	–	–	–
TiPToP	87.4	39.4	0.0	39.4	0.0	0.0	0.0	0.0	0.0
CaP-X	70.5	–	–	–	–	–	–	–	–
Periodic trigger	71.7	53.8	0.0	0.0	0.0	53.7	0.0	0.0	0.1
Proprioceptive trigger	71.1	38.6	0.0	0.0	0.0	38.2	0.0	0.4	0.0
VoLo	68.2	16.4	4.3	0.0	0.0	12.1	0.0	0.0	0.0
Proprioceptive verifier	66.3	7.0	0.0	0.0	0.0	2.6	0.1	4.4	0.0
PSP-Agent	49.8	31.8	3.3	0.0	5.8	21.7	0.0	0.0	1.0
GPT-Policy	16.6	–	–	–	–	–	–	–	–
Privileged verifier	67.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
RoboLab-Long									
Low-level policy	43.4	–	–	–	–	–	–	–	–
Low-level policy (Cosmos-3)	61.0	–	–	–	–	–	–	–	–
TiPToP	77.6	46.7	0.0	46.7	0.0	0.0	0.0	0.0	0.0
CaP-X	57.6	–	–	–	–	–	–	–	–
Periodic trigger	42.9	28.9	0.0	0.0	0.0	28.9	0.0	0.0	0.0
Proprioceptive trigger	41.1	26.3	0.0	0.0	0.0	26.3	0.0	0.0	0.0
VoLo	38.0	7.0	0.0	0.0	0.0	7.0	0.0	0.0	0.1
Proprioceptive verifier	43.9	10.3	0.0	0.0	0.0	5.4	3.8	1.0	0.1
PSP-Agent	18.4	13.3	0.0	0.0	8.5	4.6	0.0	0.0	0.2
GPT-Policy	21.3	–	–	–	–	–	–	–	–
Privileged verifier	36.7	12.4	0.0	0.0	0.0	0.0	12.3	0.1	0.0

Measurement precision and caveats. Classifying precise placement versus near-target placement relies on oriented bounding box predicates in the simulator. Validating against logged axis-aligned extents reproduces the simulator's per-placement verdict for 73.0% of 504 sampled subgoals. Disagreements go both ways (11.1% credited without the logged geometry placing the object, 15.9% placed by the geometry without credit). The second kind is what we label slightly off target, so part of those may be the geometry's error rather than the robot's.

Online verification accuracy. Deployed online against simulator ground truth, the visual verifier achieves 88.6% precision and 61.8% recall across 23,145 calls (89.9%/62.4% on SimplerEnv-Long, 88.5%/56.5% on RobotArena-∞-Long and 84.0%/62.7% on RoboLab-Long). A missed completion makes the orchestrator keep commanding a subgoal that is already done, which uses up the budget: on RobotArena-∞-Long, this leaves 4.4% of the verifier's subgoals never issued (Table 18).

Failures with every subgoal completed. The subgoal view cannot see one failure: an episode that completes every subgoal but breaks the task's ordering or exclusion rule. This is 2.5% of the 14,824 failed episodes of the ten systems of Table 3, and 32.3% of the failed episodes of ordered tasks.