ROBOBRANCH: BRANCHING EXPLORATION AS A NEW SCALING AXIS FOR ROBOT-USE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

General-purpose AI agents are moving from browsers and terminals to robot bodies. These *robot-use agents* perceive through cameras, act through navigation, perception, and manipulation tools, and improve without any weight updates by learning task guidance from interaction. However, existing agents learn from independent rollouts, which reveal only one outcome at each state they visit: testing another action means replaying a similar trajectory and reaching a state that differs in both context and action. This raises a question: *can a robot-use agent learn from many outcomes of the same state, rather than one outcome of many states?* We answer it with **RoboBranch**, the first framework that makes branching exploration in simulation a new scaling axis for robot-use agents, featuring three designs: (i) **Shared-State Branching**, which executes multiple candidate actions from identical copies of each decision state; (ii) **Outcome-Grounded Guidance Learning**, which revises task guidance from every outcome, selected or not, successful or failed; and (iii) **Width-Scalable Runtime**, which proposes all candidates in one response and executes branches in parallel, so that wider exploration yields more outcomes without a proportional increase in wall-clock time. Across LIBERO-Pro, RoboCasa365, and RoboTwin C2R, RoboBranch improves over Harness VLA built on the same open-source Qwen model by 10.6 points on average and brings the open-source agent into the range of proprietary agents built on Claude Code and Codex. We show that simulation can do more than run additional trajectories: it lets an agent learn more from every state it reaches.

1 INTRODUCTION

General-purpose AI agents that operate browsers, terminals, and computers are beginning to operate robots. Given camera observations and access to navigation, perception, and manipulation tools, such an agent drives a robot body through the same tool-use loop that drives a computer. This emerging class of *robot-use agents* (Isola, 2026) can improve without updating any model weights: it learns *task guidance* from interaction, natural-language guidance that states when an action applies, how to execute it, how to check progress, and how to recover when it fails. The better the guidance, the more capable the same frozen agent becomes.

Recent systems show how much such guidance can improve robot behavior. RoboClaw accumulates experience while unifying robot data collection, policy learning, and deployment (Li et al., 2026); Harness VLA learns task traces, success rules, and failure models to orchestrate a frozen VLA with analytic robot primitives (Zhang et al., 2026b). These agents learn from independent rollouts, which restart from reset and often repeat similar actions. Every visited state yields exactly one outcome, and this one-outcome-per-state structure limits what an agent can learn about any single decision. Consider an agent that asks the VLA to grasp a drawer handle, and the drawer stays closed. Was the grasp alone insufficient, or was the approach wrong? To find out, the agent must replay a comparable trajectory prefix and try again from a state that now differs in both context and action. Every rollout adds experience, yet the evidence about any single decision stays thin.

Humans learn such skills differently. Facing an unfamiliar faucet, a person tries a push, then a twist, then a lift, all from the same spot, and remembers which one works. Simulation gives a robot-use agent an even stronger version of this ability: it can save the exact moment of a decision and return to it as often as needed. This leads us to ask: *can a robot-use agent learn from many outcomes of the same state, rather than one outcome of many states?* Viewed this way, an independent rollout is simply a branch set of size one. Widening the branch set adds comparability across actions, turning every reached state into a controlled comparison of what each action changes.

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

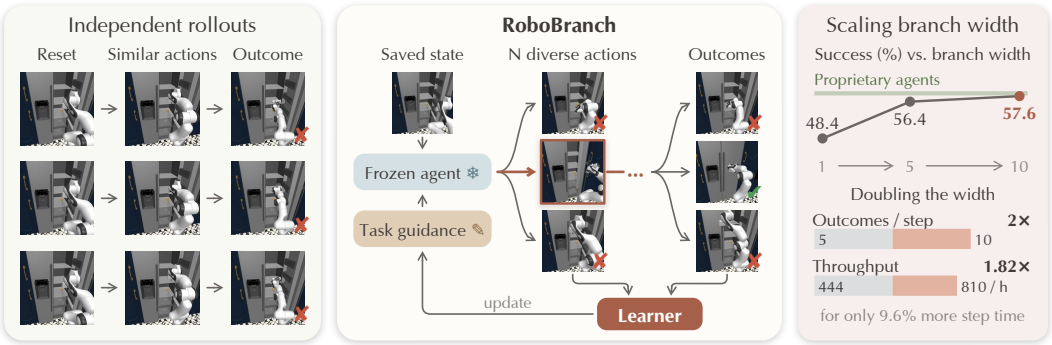

Figure 1: **Learn many outcomes from each state.** Independent rollouts restart from reset, often sample similar actions, and reveal one outcome per attempt. RoboBranch saves a reached state, where the frozen agent, informed by the current task guidance, proposes N distinct actions in one response; the selected branch continues the trajectory and branches again, and the learner turns every outcome, successful or failed, into updated task guidance. Success on RoboTwin C2R rises from 48.4% with one candidate to 57.6% with ten, within one point of proprietary agents, and doubling the branch width from five to ten doubles the outcomes per step and raises outcome throughput on RoboCasa365 by 1.82× without a proportional increase in step time.

We answer this question with RoboBranch, the first framework that makes branching exploration in simulation a new scaling axis for robot-use agents. RoboBranch is built on three designs. **(i) Shared-state branching:** at each decision, the agent proposes diverse candidate actions, and RoboBranch snapshots the simulator, controller, and agent session, then executes every candidate from an identical copy of that state. One resulting state continues the exploration trajectory. For example, from a state with the open gripper at a drawer handle, a grasp-only request leaves the drawer closed, while "pull the drawer open by grasping its handle" opens it. **(ii) Outcome-grounded guidance learning:** a learner receives each branch set intact, including task successes, local progress, actions with no effect, and execution errors, whether or not a branch was selected. It revises task guidance so that every entry cites the outcomes that support it, and the revised guidance steers the next exploration round. **(iii) Width-scalable runtime:** one structured VLM response proposes the full candidate set, and isolated simulator workers execute the branches concurrently while sharing a batched model-serving pool. Doubling the branch width from five to ten yields twice as many outcomes per step without a proportional increase in step time. After exploration, the task guidance is frozen, and the same agent executes each evaluation episode as a single trajectory.

Our experiments answer the question affirmatively. We evaluate RoboBranch on LIBERO-Pro, RoboCasa365, and RoboTwin C2R, with one open-source Qwen model serving as both the agent and the guidance learner. Compared with Harness VLA built on the same Qwen model, RoboBranch improves success on all three benchmarks, by 11.5 points on LIBERO-Pro, 9.6 points on RoboCasa365, and 10.8 points on RoboTwin C2R, or 10.6 points on average. These gains bring the open-source agent into the range of Harness VLA with Claude Code and Harness VLA with Codex on LIBERO-Pro and RoboCasa365, and within one point of both on RoboTwin C2R. Ablations on all three benchmarks show that branching outperforms ten times as many independent rollouts by 3.6 to 5.9 points and improves both where exploration goes and what the learner sees; outcomes short of success alone add 5.2 to 9.0 points. Our contributions are threefold:

- **Branching exploration as a new scaling axis.** We introduce RoboBranch, which branches from saved simulator states so that every reached state yields a controlled comparison of candidate actions instead of a single outcome. A learner turns complete branch sets, selected and unselected, successful and failed, into evidence-cited task guidance.
- **A runtime that turns exploration width into throughput.** One structured response proposes every candidate, and parallel simulator workers share a batched serving pool, so doubling the branch width doubles the outcomes per step without a proportional increase in step time and raises outcome throughput by 1.82×.
- **Open-source agents in the range of proprietary agents.** Across three benchmarks, RoboBranch improves over Harness VLA built on the same open-source Qwen model by 10.6 points on average and brings the open-source agent into the range of proprietary agents, i.e., Claude Code and Codex.

2 Related Work

Robot-Use Agents. Robot-use agents shift the unit of robot intelligence from a single policy to a system: a general-purpose model observes the world, calls robot tools, reads their outcomes, and decides what to do next (Isola, 2026). Early systems establish this interaction pattern by grounding language plans in learned affordances or composing perception and control APIs into executable behavior (Ahn et al., 2022; Liang et al., 2022; Singh et al., 2022; Huang et al., 2022; 2023). Recent work develops persistent agent runtimes around this loop. CaP-X, Thea, RHO, and Act-Observe-Rewrite study coding-agent interfaces, executable tool use, and adaptation from interaction (Fu et al., 2026; Wang et al., 2026; Elmaaroufi et al., 2026; Kumar, 2026). VLAs-as-Tools, VoLo, ETA, and RoboHarness incorporate learned robot policies into agent-controlled execution (Lei et al., 2026; Chen et al., 2026a;b; Huang et al., 2026), while Harness VLA combines a frozen VLA with analytic primitives and memory-guided orchestration (Zhang et al., 2026b). Together, these systems make the organization of tools, feedback, and experience central to robot behavior. RoboBranch advances this direction by branching from states already reached and retaining the outcomes for later execution.

Embodied Foundation Models. The physical capabilities of a robot-use agent depend on the models behind its tools. Generalist robot policies connect language and visual observations to actions across tasks and embodiments (Brohan et al., 2022; 2023; Octo Model Team et al., 2024; Kim et al., 2024; Black et al., 2024; Physical Intelligence et al., 2025). Agentic systems increasingly use these policies as callable components inside a larger decision loop: VLAs-as-Tools develops policies for bounded agent invocations (Lei et al., 2026), while Harness VLA exposes a frozen VLA as a contact-rich primitive alongside analytic operations (Zhang et al., 2026b). RoboBranch keeps the VLA backends frozen and expands exploration at the agent layer by comparing multiple tool outcomes from the same simulator state. Test-time scaling samples many candidate actions from a VLA and selects one with a learned verifier at every step (Kwok et al., 2025), whereas RoboBranch branches only during exploration and executes a single trajectory at evaluation.

Self-Improving Agents. Once the agent and its tools are fixed, further improvement must come from what the agent retains from experience. Digital agents use reflection, self-critique, iterative refinement, episodic memory, and evolving external context (Shinn et al., 2023; Gou et al., 2023; Madaan et al., 2023; Park et al., 2023; Zhao et al., 2023; Zhang et al., 2025); AutoGuide extracts context-aware guidelines by contrasting successful and failed trajectories (Fu et al., 2024). Voyager and MUSE-Autoskill organize experience into reusable skills and managed skill lifecycles (Wang et al., 2023; Lin et al., 2026). Embodied agents ground this loop in action outcomes. Harness VLA records successful command traces and failure models (Zhang et al., 2026b); HELM and HyMeS couple memory with execution-time verification or steering (Zeng et al., 2026; Zhao et al., 2026); RoboClaw couples autonomous data collection with iterative policy learning (Li et al., 2026); ASPIRE validates agent-written repairs (Lu et al., 2026); and Zetta evolves runtime critics and recovery skills from rollout failures (Ding et al., 2026). These systems expand experience through additional trajectories, search rounds, or policy updates. RoboBranch instead expands each decision state: it restores the same saved state for multiple candidate actions, compares their outcomes under identical starting conditions, and uses the complete outcome set to revise task guidance. Branching therefore learns from multiple outcomes at states that an independent rollout would use only once.

Exploration from Restored States. Returning to previously reached states is a powerful exploration primitive. Go-Explore returns to promising states before exploring further and solves hard-exploration games (Ecoffet et al., 2021), and language agents expand trees of candidate actions to choose better actions at inference time (Zhou et al., 2024; Koh et al., 2024). Web agents turn such search trees into training signal through preference optimization or reflection (Putta et al., 2024; Yu et al., 2025b), and reinforcement learning for language models branches rollouts from intermediate states to assign credit to individual steps (Kazemnejad et al., 2025; Hou et al., 2025). RoboBranch brings state restoration to robot-use agents as a way to improve task guidance. Branch sets from restored simulator states become evidence for guidance that the agent carries into single-trajectory execution, so the value of every branch persists beyond the episode that produced it.

3 RoboBranch

3.1 Overview

RoboBranch is built on one observation: an independent rollout is a branch set of size one. Every reached state can yield many outcomes instead of one, provided that the simulator can return to it. RoboBranch turns this observation into a learning loop that alternates between branching exploration and guidance updates (Figure 2; Algorithm 1).

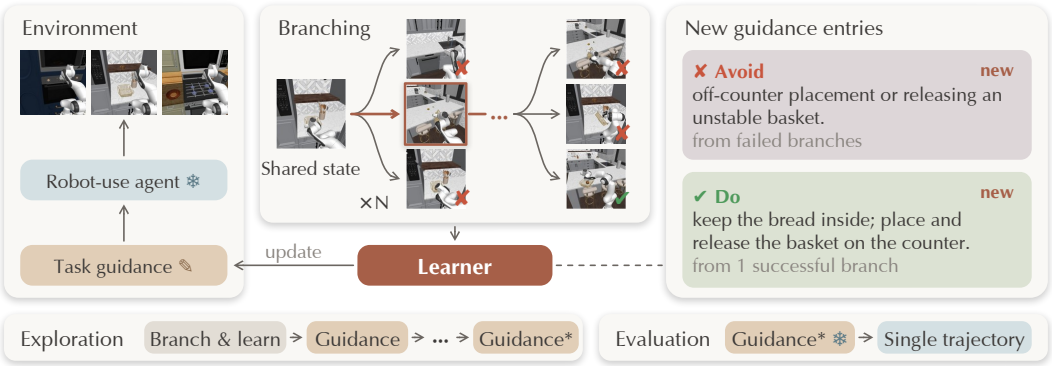

Figure 2: **Overview of RoboBranch.** During exploration, the frozen robot-use agent acts with the current task guidance, and each decision state branches into N candidate actions, one of which is selected to continue the trajectory. The learner compares successful and failed branches from the same state and writes new guidance entries that state what to do and what to avoid. Exploration alternates branching and learning; evaluation executes a single trajectory with the frozen guidance.

Task guidance $\mathcal{G}$ is a natural-language document maintained for each task. It states when an action applies, how to execute it, how to recognize progress or completion, and how to respond when an action fails (Appendix A). A frozen agent reads $\mathcal{G}$ at the start of each episode and grounds it against live observations at every decision. Each run begins from $\mathcal{G}_0$, an empty guidance document. In round m, the agent explores with $\mathcal{G}_m$: at each decision, candidate actions execute from copies of a saved simulator state, and one resulting state continues the trajectory (Section 3.2). The learner then revises $\mathcal{G}_m$ from every observed outcome, and the revision becomes the next guidance version $\mathcal{G}_{m+1}$ (Section 3.3). The agent, VLA, interface, and tools stay fixed; only the task guidance improves.

3.2 Shared-State Branching

Consider a decision state s_t reached after a potentially long trajectory prefix. The agent proposes candidate actions $a_t^{(1)}, \ldots, a_t^{(N)}$ that cover distinct tools, instructions, or action arguments. RoboBranch snapshots s_t before any candidate executes, including the simulator, controller, and agent-session state needed to reproduce the decision point, and executes each candidate from an isolated copy of that snapshot. Each executed candidate defines one branch:

$$b_t^{(i)} = \text{Execute}\left(s_t, a_t^{(i)}\right), \qquad \mathcal{B}_t = \left\{b_t^{(i)}\right\}_{i=1}^{N}. \tag{1}$$

Each branch ends when its action returns, reaches its control horizon, or completes the task. Each $b_t^{(i)}$ records the executed action, its terminal observation, changes in task progress, the result of the simulator's task-success check, and any execution error.

Independent rollouts observe one outcome at each state they visit; branching observes many actions from the same state. What makes these outcomes comparable is their shared origin. Object poses, action histories, and policy states are identical across the branch set, so the candidates differ only in the action the agent chooses. The trajectory prefix is executed once, and the extra exploration budget is spent entirely on trying different actions from that state.

The learner then picks the branch i^* with the most task progress, preferring task success, to continue the exploration trajectory. The exploration trajectory stays linear, while every outcome in $\mathcal{B}_t$ remains available to the learner. We hold N fixed within a run, and Section 5.3 varies N; Section 4 describes the runtime that executes each branch set efficiently.

3.3 Outcome-Grounded Guidance Learning

The learner decides both where exploration goes next and what the agent should remember. It is the same frozen Qwen model that serves as the agent, prompted to revise task guidance; no model weights change, and no stronger model is involved. The learner receives each $\mathcal{B}_t$ intact, with candidate actions and outcomes grouped by their shared starting state, and uses evidence from every candidate, selected or not. Task success, local progress, no observed effect, and execution errors remain distinct observations, so a single branch set can show what completes a step, what advances it, and what to change; the learner marks the informative candidates as good or bad. For an exploration round containing J decision states, RoboBranch updates its task guidance from the corresponding branch sets:

$$\mathcal{G}_{m+1} = \text{Update}(\mathcal{G}_m, \mathcal{B}_{m,1}, \ldots, \mathcal{B}_{m,J}) \,. \tag{2}$$

The drawer example from the introduction shows how a branch set becomes guidance. Both actions below execute from the same saved state, with the open gripper in front of the drawer handle.

✗ Avoid
a grasp-only request that closes the gripper without pulling.
from `vla_arm("close the gripper on the brass drawer handle in front of you")`, which reaches its control horizon with the drawer closed

✓ Do
grasp and pull the handle in one request.
from `vla_arm("pull the drawer open by grasping its handle")`, which opens the drawer and completes the task

Both outcomes start from the same state, and the learner writes both into task guidance.

Evidence-cited revisions. After each exploration round, the learner adds guidance for decisions not yet covered or revises the applicability, procedure, success checks, recovery, or limitations of existing guidance. Every proposed entry cites the executed outcomes that support it, which keeps the guidance grounded in observed effects.

3.4 Single-Trajectory Evaluation

After exploration, RoboBranch freezes the final guidance version. Each evaluation episode loads the relevant task guidance once. At every decision, the agent checks the guidance against the current observation, executes one robot action, observes the result, and decides again. Branching happens only during learning: at evaluation, the agent follows a single trajectory, and all improvement comes from the task guidance it carries. Section 5 measures how guidance improves with branch width.

4 Width-Scalable Runtime

RoboBranch learns from a decision state once its candidate actions have been proposed, executed, and returned as a complete branch set. The rate at which branch sets finish therefore sets how fast task guidance improves. A naive implementation turns wider exploration into slower exploration: generating N candidates through independent VLM calls repeats the same context N times, and executing N branches serially places their full cost on the critical path. The RoboBranch runtime removes both bottlenecks (Figure 3).

One response, many candidates. Sampling candidates through separate calls resends the same multimodal context and pays a full round trip for every candidate. RoboBranch instead returns the complete candidate set in one structured response, amortizing proposal latency across the branch set. Wider exploration no longer requires more VLM round trips.

Many simulators, shared models. Robot-use rollouts alternate between environment execution and calls to large perception and action models (Ding et al., 2026). Binding one model replica to every simulator would tie exploration width to GPU memory. RoboBranch decouples the two resource pools: lightweight environment workers expand branch execution (Makoviychuk et al., 2021; Weng et al., 2022; Zakka et al., 2026), while a shared serving pool batches model requests across branches (Kwon et al., 2023). Exploration widens without replicating the frozen models, and simulator stepping overlaps with model inference.

One snapshot, many rollouts. The coordinator advances the exploration trajectory to a decision state, snapshots it once, and dispatches isolated copies to the available simulator workers. Each worker executes one candidate action and returns its outcome. Branches overlap in wall-clock time, so the critical path is set by the slowest branch rather than the sum of all branches. When the branch set finishes, the state reached by the selected branch continues the exploration trajectory and the remaining copies are released.

Shared-state semantics at scale. Each worker receives the same origin and its own controller state and agent session. Branch identities follow requests through simulator execution and VLA serving, so every candidate keeps its own action history. The coordinator merges completed outcomes and continues from exactly one selected branch, preserving the semantics of Section 3.2 at any width.

Together, these designs turn exploration width into throughput: wider branch sets produce more observed outcomes without placing their full cost on the critical path. Section 5.5 measures step time, branch-set time, and outcome throughput as the branch width doubles.

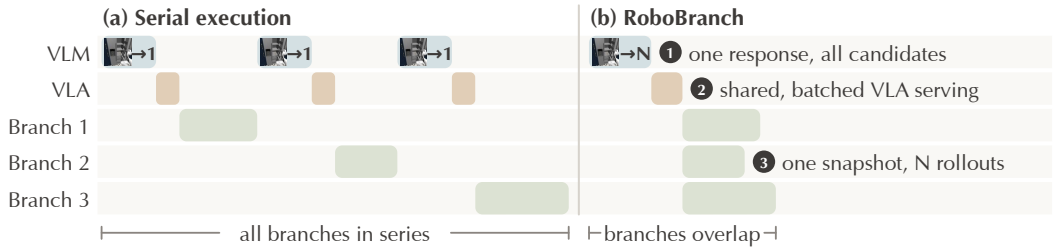

Figure 3: **Turning exploration width into throughput.** (a) Serial execution proposes one candidate per VLM call and executes branches one at a time, so every branch lies on the critical path. (b) RoboBranch (1) proposes all candidates in one response, (2) serves VLA requests from a shared batched pool, and (3) executes every branch from one snapshot in parallel, so branch execution overlaps. Block lengths are schematic; Table 12 (Appendix E) reports the measured times.

5 Experiments

Our experiments answer five questions. **Q1:** How much does RoboBranch improve an open-source agent, and does it enter the range of proprietary agents? (Section 5.2) **Q2:** How does task guidance improve as the branch width grows? (Section 5.3) **Q3:** Which parts of branching make it useful? (Section 5.4) **Q4:** How efficiently does the runtime turn exploration width into throughput? (Section 5.5) **Q5:** What does branching teach the agent? (Section 5.6)

5.1 Experimental Setup

Benchmarks. We evaluate on LIBERO-Pro (Liu et al., 2023; Zhou et al., 2025), RoboCasa365 (Nasiriany et al., 2024; 2026), and RoboTwin C2R (Chen et al., 2025), covering perturbed tabletop manipulation, long-horizon household tasks, and clean-to-randomized bimanual manipulation. For RoboCasa365, the evaluation contains 18 atomic tasks, 16 seen composite tasks, and 16 unseen composite tasks. Atomic tasks use ten seeds and composite tasks use five, for 340 task–seed instances. Following Harness VLA (Zhang et al., 2026b), "unseen" marks task templates outside the pretraining data of the VLA, and RoboTwin C2R guidance is learned in the clean setting and evaluated in the randomized one. RoboBranch learns task guidance for each task and evaluates it on new instances of that task: on LIBERO-Pro and RoboCasa365, exploration uses seeds disjoint from the evaluation seeds, and task guidance is frozen before evaluation.

Models and baselines. RoboBranch uses a frozen Qwen agent and a frozen VLA on each benchmark: π_{RLinf}, the RLinf-released $\pi_{0.5}$-SFT checkpoint (Yu et al., 2025a), on LIBERO-Pro, RLDX-1 (Kim et al., 2026) on RoboCasa365, and LingBot-VLA (Wu et al., 2026) on RoboTwin C2R. It runs in a minimal harness that exposes only the observation and robot-control tools each benchmark requires. The main tables group agentic systems by their underlying model. Harness VLA + Qwen is the open-source baseline and uses the same model as RoboBranch; Harness VLA explores each task until it succeeds and stores the successful trace as task memory. Harness VLA + Claude Code and Harness VLA + Codex build on proprietary models, and their published results (Zhang et al., 2026b) mark the performance level of proprietary agents. The tables also list frozen VLA policies and prior systems. All RoboBranch mechanism ablations keep the agent, VLA, harness, and observations fixed and evaluate on the same task–seed instances.

Implementation details. We use Qwen3.8-Flash-Next, with 125B parameters and 6B activated per token, for both the frozen agent and the learner that revises task guidance (Qiu et al., 2026). All RoboBranch experiments run on a cluster of 32 NVIDIA H100 GPUs. The Harness VLA baselines pair Claude Code with Claude Opus 4.7 and Codex with GPT-5.5 (Zhang et al., 2026b).

Evaluation. Task guidance is learned only during exploration and then frozen; evaluation runs without branching or write-back. Our primary performance metric is task-macro success: each task receives equal weight, independent of its number of seeds. The exploration budget is the number of exploration rounds; each decision executes N candidate branches, and all branch widths use the same number of rounds. Appendix C reports the average budget per task, and Table 12 and Appendix E report the time per branching step.

5.2 Main Results

Tables 1 and 2 answer Q1 with two consistent findings.

RoboBranch substantially improves the open-source agent. With the same Qwen model, RoboBranch outperforms Harness VLA on all three benchmarks, evaluated on new instances of the ex-

Table 1: LIBERO-Pro success rate (%) across four task families under instruction-redirection (T) and position-swap (S) perturbations. Results other than RoboBranch and Harness VLA + Qwen are published results (Zhang et al., 2026b). RoboBranch gains from branching exploration, which gives the learner more outcomes to learn from at each decision without a proportional increase in exploration time (Section 5.5). Proprietary agents build on Claude Opus 4.7 (Claude Code) and GPT-5.5 (Codex). †Overall averages the six reported splits. Shading pairs RoboBranch with Harness VLA + Qwen, which uses the same Qwen model; the darker cell in each column is the better of the two.

Method	Spat-T	Spat-S	Obj-T	Obj-S	Goal-T	Goal-S	L10-T	L10-S	Overall
VLA policies and prior systems									
$\pi_{0.5}$	1.0	20.0	1.0	17.0	2.0	38.0	1.0	8.0	11.0
CaP-X (Fu et al., 2026)†	14.0	12.0	18.0	22.0	17.0	26.0	–	–	18.2
RATs (Zhang et al., 2026a)†	31.0	29.0	63.0	61.0	36.0	43.0	–	–	43.8
π_{RLinf} (Yu et al., 2025a)	42.0	59.0	71.0	78.0	45.0	42.0	49.0	14.0	50.0
Proprietary agents									
Harness VLA + Claude Code	94.0	80.0	88.0	90.0	87.0	87.0	71.0	62.0	82.4
Harness VLA + Codex	81.0	69.0	94.0	91.0	75.0	66.0	52.0	49.0	72.1
Open-source agents									
Harness VLA + Qwen	62.0	66.0	76.0	70.0	65.0	61.0	60.0	35.0	61.9
RoboBranch + Qwen	73.0	89.0	88.0	94.0	78.0	69.0	58.0	38.0	73.4

Table 2: Success rates (%) on RoboCasa365 (left: Atomic-Seen, Composite-Seen, Composite-Unseen, and task-macro Overall over 50 tasks) and RoboTwin C2R (right: clean-to-randomized, 50 tasks, five seeds each). Results other than RoboBranch and Harness VLA + Qwen are published results (Zhang et al., 2026b). Agents build on the models listed in Table 1; CC: Claude Code. Shading as in Table 1.

Method	AS	CS	CU	Overall
VLA policies				
RLDX-1 (Kim et al., 2026)	60.0	21.3	5.0	30.0
WorldDreamer (World Agents, 2026)	66.3	26.7	9.0	35.3
$\pi_{0.5}$	39.6	7.1	1.2	16.9
π_0	34.6	6.1	1.1	14.8
Proprietary agents				
Harness VLA + CC	79.4	47.5	15.0	48.6
Harness VLA + Codex	92.0	61.0	13.8	57.1
Open-source agents				
Harness VLA + Qwen	71.7	27.5	15.0	39.4
RoboBranch + Qwen	89.4	35.0	17.5	49.0

Method	C2R
VLA policies	
$\pi_{0.5}$	47.9
GR00T-N1.7 (Bjorck et al., 2025)	20.7
StarVLA (StarVLA Community, 2026)	10.6
LingBot-VLA (Wu et al., 2026)	50.4
Proprietary agents	
Harness VLA + CC	58.4
Harness VLA + Codex	58.0
Open-source agents	
Harness VLA + Qwen	46.8
RoboBranch + Qwen	57.6

plored tasks, by 10.6 points on average. On LIBERO-Pro, success rises from 61.9% to 73.4%, with gains on seven of the eight splits; the largest gains occur on Object-Swap and Spatial-Swap, where success increases by 24 and 23 points, as guidance learned from branch outcomes ties each action to the observed state and carries over when objects move. The instruction-redirection splits of Spatial, Object, and Goal also gain 11 to 13 points. On RoboCasa365, task-macro success rises from 39.4% to 49.0%, with gains on Atomic-Seen, Composite-Seen, and Composite-Unseen tasks. Atomic tasks rise from 71.7% to 89.4%, above Harness VLA + Claude Code and within three points of Codex; on these tasks, one branch set often reveals the instruction that makes the VLA succeed, as in the drawer example of Figure 5. On RoboTwin C2R, RoboBranch raises success from 46.8% to 57.6%, a gain of 10.8 points. The gains come from the learned task guidance: with the same agent and harness but no task guidance, success is 55.0%, 30.0%, and 42.0%, below Harness VLA + Qwen on every benchmark (Table 3). The agent needs task guidance to use the VLA well: on RoboTwin C2R, the agent without guidance falls below LingBot-VLA alone (42.0% vs. 50.4%), and branching exploration lifts the same agent to 57.6%.

RoboBranch brings the open-source agent into the range of proprietary agents. The two proprietary agents trade wins across benchmarks. On LIBERO-Pro and RoboCasa365, RoboBranch reaches the range of Harness VLA + Claude Code and Harness VLA + Codex, with 73.4% and 49.0% overall success; on RoboTwin C2R, it reaches 57.6%, within one point of both. RoboBranch also sets the best result of any method on LIBERO-Pro Spatial-Swap (89.0%) and Object-Swap (94.0%).

5.3 Scaling with Branching Exploration

We vary the number of candidates at each decision state on RoboTwin C2R, keeping the agent, VLA, harness, and evaluation protocol fixed (Figure 4). With a single candidate, the same learning

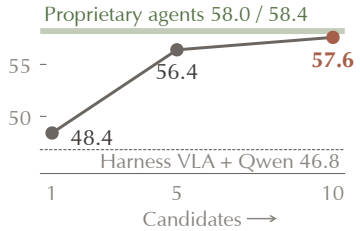

Figure 4: **Scaling branch width.** RoboTwin C2R success of task guidance learned with 1, 5, and 10 candidates per decision state. The band marks the two proprietary agents built on the same frozen VLA (58.0% and 58.4%); the dashed line marks Harness VLA + Qwen (46.8%).

Table 3: Ablations of RoboBranch (overall success, %). "Selected outcomes only" and "Successful outcomes only" change which outcomes the learner receives; "5 candidates" halves the branch width; "Independent rollouts" replaces branching with ten times as many independent rollouts; "No task guidance" evaluates the agent with $\mathcal{G}_0$. Parentheses give the gain of RoboBranch over each variant.

Variant	LIBERO-Pro	RoboCasa365	RoboTwin
RoboBranch	**73.4**	**49.0**	**57.6**
Selected outcomes only	69.5 (+3.9)	46.4 (+2.6)	54.0 (+3.6)
Successful outcomes only	64.4 (+9.0)	43.0 (+6.0)	52.4 (+5.2)
5 candidates	70.1 (+3.3)	46.8 (+2.2)	56.4 (+1.2)
Independent rollouts	67.5 (+5.9)	44.4 (+4.6)	54.0 (+3.6)
No task guidance	55.0 (+18.4)	30.0 (+19.0)	42.0 (+15.6)

loop runs without branching and reaches 48.4% success (121/250). Five candidates raise success to 56.4%, and ten candidates reach 57.6%, within one point of both proprietary agents built on the same frozen VLA (58.0% and 58.4%). Branching thus adds 9.2 points over single-candidate exploration. Five candidates already add 8.0 points on RoboTwin C2R, and on LIBERO-Pro and RoboCasa365 doubling the width from five to ten still adds 3.3 and 2.2 points (Table 3). Ten candidates return ten outcomes from each agent decision, twice as many as five candidates, without a proportional increase in step time on RoboCasa365 (Section 5.5).

5.4 What Makes Branching Useful?

We ablate three parts of RoboBranch and compare branching with independent rollouts on all three benchmarks, while keeping the frozen agent, VLA, harness, and evaluation protocol unchanged (Table 3). The full method gives the learner every branch outcome. *Selected outcomes only* lets the learner learn only from the branch that continues exploration at each decision state, *Successful outcomes only* lets it learn only from branches whose action completes the task, *5 candidates* halves the branch width from ten to five, and *Independent rollouts* runs single-candidate exploration with ten times as many rollouts.

Branching improves both where exploration goes and what the learner sees. The learner sees N outcomes at every decision state and continues exploration from the best one, so it learns from stronger trajectories; it also receives every outcome of the branch set. On RoboTwin C2R, branching with the learner restricted to selected outcomes raises success from 48.4% to 54.0%, and giving the learner every outcome raises it further to 57.6%. With the same exploration data, continuations, and update schedule, but with every outcome given as an independent transition without its shared origin, success drops to 53.6%, so the further gain comes from comparing outcomes of the same state. Ten times as many independent rollouts reach 67.5%, 44.4%, and 54.0% on LIBERO-Pro, RoboCasa365, and RoboTwin C2R, no higher than branching with selected outcomes alone on any benchmark.

Outcomes short of success carry the most guidance. Failures, partial progress, and actions with no effect together add 9.0, 6.0, and 5.2 points on LIBERO-Pro, RoboCasa365, and RoboTwin C2R, the largest contribution of any component. A failure next to a success from the same state isolates the difference that decides the outcome, as in the drawer example of Section 3.3.

Unselected outcomes carry guidance. Learning from the outcomes of unselected branches improves success by 3.9, 2.6, and 3.6 points. The candidates that exploration leaves behind still show which alternatives work and which to avoid.

Wider branch sets yield better guidance. Doubling the branch width from five to ten candidates improves success by 3.3, 2.2, and 1.2 points, giving the learner more evidence per decision.

5.5 System Efficiency

We profile branching steps of the full system, covering candidate proposal and branch execution, on RoboCasa365 with five and ten candidates per decision state, excluding one-time initialization, and compare them with serial execution, which proposes one candidate per VLM call and executes one branch at a time (Table 12). Each candidate produces one observed branch outcome, so doubling the branch count doubles the outcomes collected at each decision state.

Doubling the branch count increases step time by 9.6%, from 40.55 to 44.46 seconds, while serial execution doubles from 135.02 to 271.66 seconds. Rollout time stays nearly constant, at 34.22 ver-

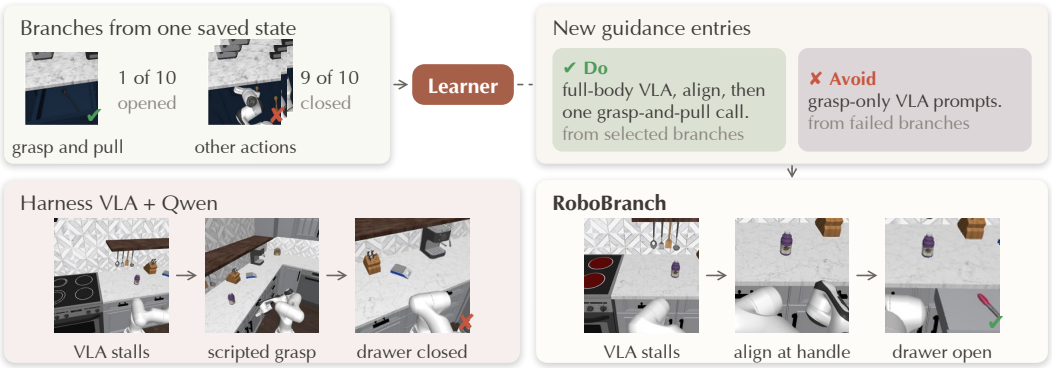

Figure 5: **From branch outcomes to evaluation on OpenDrawer.** From one saved state with the gripper at the drawer handle, one of ten branches opens the drawer with a single grasp-and-pull arm VLA call, while the other nine, including a grasp-only prompt, leave it closed. The learner writes both outcomes into task guidance. At evaluation in a different kitchen, the full-body VLA stalls for both agents; Harness VLA + Qwen falls back on scripted grasps and fails, while RoboBranch aligns the gripper and opens the drawer with the learned call.

sus 36.23 seconds, because the branches execute in parallel, and VLM time grows only from 5.80 to 7.79 seconds because one response proposes every candidate. Since each step returns twice as many outcomes, outcome throughput rises from 444 to 810 outcomes per hour, a 1.82× improvement. Wider branching thus increases decision-level exploration without placing the additional branch execution on the serial critical path.

Against serial execution, the runtime gains grow with the branch width. With ten candidates, RoboBranch completes a branching step in 44.5 seconds instead of 271.7 seconds, a 6.11× speedup that removes 83.6% of the wall-clock time (Table 12); with five candidates, the speedup is 3.33×. Proposing all candidates in one response cuts VLM generation from 40.8 to 7.8 seconds, and parallel branch execution cuts rollout time from 226.2 to 36.2 seconds. RoboBranch thus combines the highest success of all exploration settings with exploration time that grows far slower than the width.

5.6 What Branching Teaches

Figure 5 follows one branch set from exploration to evaluation on the RoboCasa365 task OpenDrawer. The branch set starts from a saved state with an open gripper at the drawer handle. One of ten branches opens the drawer by grasping and pulling the handle in one arm VLA call; the other nine leave it closed. The learner writes this contrast into guidance: start with the full-body VLA, align the open gripper at the handle, grasp and pull in one arm call, and avoid grasp-only prompts at the handle. On an evaluation instance where the full-body VLA stalls, Harness VLA + Qwen spends 86 commands on repeated VLA calls, navigation, and scripted grasps and ends with the drawer closed. RoboBranch follows the guidance and opens the drawer in three commands.

6 Conclusion

We introduced RoboBranch, the first framework that makes branching exploration in simulation a new scaling axis for robot-use agents. Executing candidate actions from identical copies of each reached state turns every decision into a controlled comparison, learning from every outcome turns those comparisons into task guidance, and a width-scalable runtime executes the branches in parallel. With the same open-source Qwen model, RoboBranch improves over Harness VLA by 10.6 points on average, outperforms ten times as many independent rollouts, and brings the open-source agent into the range of proprietary agents. We show that simulation is more than a source of additional trajectories: it lets an agent learn more from every state it reaches.

Limitations and future work. RoboBranch branches from simulators that restore complete decision states, and its learner reads the task-progress and success signals that simulation exposes. Extending branching to real robots through digital twins and learned world models, with progress estimated from observations, is a natural next step. Task guidance is learned per task and steers the skills of a frozen VLA; pairing branching exploration with policy learning would let those skills improve as well. As task guidance accumulates across many tasks and embodiments, organizing, retrieving, and sharing it across many agents opens a further direction for scaling robot-use agents.

REPRODUCIBILITY STATEMENT

We will release the code of RoboBranch, including the branching coordinator, the width-scalable runtime, the prompts for the agent and the learner, the evaluation scripts, and the learned task guidance for every benchmark. Section 3 specifies shared-state branching and the guidance update, Section 4 describes the runtime, Section 5 and Appendix C list the benchmarks, task and seed counts, models, and hardware, and Appendix A gives the structure of task guidance. All benchmarks are public, and the agent and the learner use the open-weight Qwen3.8-Flash-Next model, so every RoboBranch result can be reproduced without proprietary models.

ETHICS STATEMENT

All experiments run in simulation, and no human subjects or personal data are involved. RoboBranch learns natural-language task guidance that people can read and audit before it is used. Deploying guidance learned in simulation on physical robots would require the usual safety measures of real-world robotics, such as workspace limits, collision checks, and human supervision.

AI USE STATEMENT

Large language models are part of the method studied in this paper: a frozen Qwen model serves as the robot-use agent and as the learner, as described in Sections 3 and 5. Separately, we used generative AI assistants to polish the writing. The authors designed the method and the experiments, ran all experiments, and wrote and verified the content of the paper; all AI-assisted text was reviewed and edited by the authors, who take full responsibility for the paper.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, et al. Do as I can, not as I say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022. URL https://arxiv.org/abs/2204.01691.

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

Kevin Black, Noah Brown, Danny Driess, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024. URL https://arxiv.org/abs/2410.24164.

Anthony Brohan, Noah Brown, Justice Carbajal, et al. RT-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022. URL https://arxiv.org/abs/2212.06817.

Anthony Brohan, Noah Brown, Justice Carbajal, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023. URL https://arxiv.org/abs/2307.15818.

Siyi Chen, Hugo Hadfield, Alex Zook, et al. VoLo: A physical orchestrator for open-vocabulary long-horizon manipulation. *arXiv preprint arXiv:2606.07723*, 2026a. URL https://arxiv.org/abs/2606.07723.

Tianxing Chen, Zanxin Chen, Baijun Chen, et al. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025. URL https://arxiv.org/abs/2506.18088.

Yitong Chen, Zezheng Huai, Sixian Li, et al. ETA: A new agentic paradigm for embodied tasks. *arXiv preprint arXiv:2608.03924*, 2026b. URL https://arxiv.org/abs/2608.03924.

Xin Ding, Liang Mi, Mingzhe Huang, et al. Zetta ζ: An efficient closed-loop embodied harness for self-evolving physical intelligence. *arXiv preprint arXiv:2608.16590*, 2026. URL https://arxiv.org/abs/2608.16590.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, 2021.

Karim Elmaaroufi, Justin Svegliato, Sarunas Kalade, Graham Schelle, Sanjit A. Seshia, and Matei Zaharia. RHO: Your coding agent is secretly a roboticist. *arXiv preprint arXiv:2606.16458*, 2026. URL https://arxiv.org/abs/2606.16458.

Letian Fu, Justin Yu, Karim El-Refai, et al. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. *arXiv preprint arXiv:2603.22435*, 2026. URL https://arxiv.org/abs/2603.22435.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. AutoGuide: Automated generation and selection of context-aware guidelines for large language model agents. In *Advances in Neural Information Processing Systems*, 2024.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023. URL https://arxiv.org/abs/2305.11738.

Zhenyu Hou, Ziniu Hu, Yujiang Li, Rui Lu, Jie Tang, and Yuxiao Dong. TreeRL: LLM reinforcement learning with on-policy tree search. In *Annual Meeting of the Association for Computational Linguistics*, 2025.

Jinbang Huang, Yuanzhao Hu, Zhiyuan Li, et al. RoboHarness: Memory-driven orchestration of heterogeneous robot policies for long-horizon planning. *arXiv preprint arXiv:2607.18060*, 2026. URL https://arxiv.org/abs/2607.18060.

Wenlong Huang, Fei Xia, Ted Xiao, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022. URL https://arxiv.org/abs/2207.05608.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Fei-Fei Li. VoxPoser: Composable 3D value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023. URL https://arxiv.org/abs/2307.05973.

Phillip Isola. Robot-use agents. Essay, September 2026. URL https://web.mit.edu/phillipi/www/writing/robot-use-agents.html.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. VinePPO: Refining credit assignment in RL training of LLMs. In *International Conference on Machine Learning*, 2025.

Dongyoung Kim, Huiwon Jang, Myungkyu Koo, et al. RLDX-1 technical report. *arXiv preprint arXiv:2605.03269*, 2026. URL https://arxiv.org/abs/2605.03269.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024. URL https://arxiv.org/abs/2406.09246.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*, 2024.

Vaishak Kumar. Act-observe-rewrite: Multimodal coding agents as in-context policy learners for robot manipulation. *arXiv preprint arXiv:2603.04466*, 2026. URL https://arxiv.org/abs/2603.04466.

Jacky Kwok, Christopher Agia, Rohan Sinha, Matt Foutter, Shulu Li, Ion Stoica, Azalia Mirhoseini, and Marco Pavone. RoboMonkey: Scaling test-time sampling and verification for vision-language-action models. *arXiv preprint arXiv:2506.17811*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, et al. Efficient memory management for large language model serving with PagedAttention. *arXiv preprint arXiv:2309.06180*, 2023. URL https://arxiv.org/abs/2309.06180.

Zixing Lei, Changxing Liu, Yichen Xiong, Minhao Xiong, Yuanzhuo Ding, Zhipeng Zhang, Weixin Li, and Siheng Chen. Towards long-horizon embodied agents with tool-aligned vision-language-action models. *arXiv preprint arXiv:2605.13119*, 2026. URL https://arxiv.org/abs/2605.13119.

Ruiying Li, Yunlang Zhou, YuYao Zhu, Kylin Chen, Jingyuan Wang, Sukai Wang, Kongtao Hu, Minhui Yu, Bowen Jiang, Zhan Su, Jiayao Ma, Xin He, Yongjian Shen, Yang Yang, Guanghui Ren, Maoqing Yao, Wenhao Wang, and Yao Mu. RoboClaw: An agentic framework for scalable long-horizon robotic tasks. *arXiv preprint arXiv:2603.11558*, 2026. URL https://arxiv.org/abs/2603.11558.

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

Huawei Lin, Peng Li, Jie Song, Fuxin Jiang, and Tieying Zhang. MUSE-Autoskill: Self-evolving agents via skill creation, memory, management, and evaluation. *arXiv preprint arXiv:2605.27366*, 2026. URL https://arxiv.org/abs/2605.27366.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. *arXiv preprint arXiv:2306.03310*, 2023. URL https://arxiv.org/abs/2306.03310.

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

Aman Madaan, Niket Tandon, Prakhar Gupta, et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023. URL https://arxiv.org/abs/2303.17651.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, et al. Isaac Gym: High performance GPU-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021. URL https://arxiv.org/abs/2108.10470.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. RoboCasa: Large-scale simulation of everyday tasks for generalist robots. *arXiv preprint arXiv:2406.02523*, 2024. URL https://arxiv.org/abs/2406.02523.

Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A large-scale simulation framework for training and benchmarking generalist robots. *arXiv preprint arXiv:2603.04356*, 2026. URL https://arxiv.org/abs/2603.04356.

Octo Model Team, Dibya Ghosh, Homer Walke, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024. URL https://arxiv.org/abs/2405.12213.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*, 2023. URL https://arxiv.org/abs/2304.03442.

Physical Intelligence, Kevin Black, Noah Brown, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025. URL https://arxiv.org/abs/2504.16054.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent Q: Advanced reasoning and learning for autonomous AI agents. *arXiv preprint arXiv:2408.07199*, 2024.

Zihan Qiu, Zekun Wang, Xiao Li, et al. On the design of Qwen3.8-Next architecture: Evaluation, efficiency, and training stability. *arXiv preprint arXiv:2608.30320*, 2026. URL https://arxiv.org/abs/2608.30320.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023. URL https://arxiv.org/abs/2303.11366.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022. URL https://arxiv.org/abs/2209.11302.

StarVLA Community. StarVLA: A lego-like codebase for vision-language-action model developing. *arXiv preprint arXiv:2604.05014*, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Qi Wang, Tianyi Wang, Chengyang Li, Shikun Ban, Yurun Chen, Yizhong Ge, Jason Qin, Chengtai Li, and Wentao Zhu. Towards the harness of embodied agents. *arXiv preprint arXiv:2608.11246*, 2026. URL https://arxiv.org/abs/2608.11246.

Jiayi Weng, Min Lin, Shengyi Huang, et al. EnvPool: A highly parallel reinforcement learning environment execution engine. *arXiv preprint arXiv:2206.10558*, 2022. URL https://arxiv.org/abs/2206.10558.

World Agents. WorldDreamer: RoboCasa365 multi-task leaderboard submission. https://github.com/robocasa-benchmark/leaderboard, 2026.

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

Chao Yu, Yuanqing Wang, Zhen Guo, Hao Lin, Si Xu, Hongzhi Zang, Quanlu Zhang, Yongji Wu, Chunyang Zhu, Junhao Hu, et al. RLinf: Flexible and efficient large-scale reinforcement learning via macro-to-micro flow transformation. *arXiv preprint arXiv:2509.15965*, 2025a.

Xiao Yu, Baolin Peng, Vineeth Vajipey, Hao Cheng, Michel Galley, Jianfeng Gao, and Zhou Yu. ExACT: Teaching AI agents to explore with reflective-MCTS and exploratory learning. In *International Conference on Learning Representations*, 2025b.

Kevin Zakka, Qiayuan Liao, Brent Yi, Louis Le Lay, Koushil Sreenath, and Pieter Abbeel. mjlab: A lightweight framework for gpu-accelerated robot learning. *arXiv preprint arXiv:2601.22074*, 2026. URL https://arxiv.org/abs/2601.22074.

Zijian Zeng, Fei Ding, Huiming Yang, and Xianwei Li. HELM: Harness-enhanced long-horizon memory for vision-language-action manipulation. *arXiv preprint arXiv:2604.18791*, 2026. URL https://arxiv.org/abs/2604.18791.

Junyi Zhang, Jiaxin Ge, Hanjun Yoo, Letian Fu, Zihan Yang, Yaowei Liu, Raj Saravanan, Shaofeng Yin, Justin Yu, Dantong Niu, et al. Playful agentic robot learning. *arXiv preprint arXiv:2606.19419*, 2026a.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025. URL https://arxiv.org/abs/2510.04618.

Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Yi Nie, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, Jiaji Rao, Zhengru Fang, Changxu Wei, Yu Wang, Wenbo Ding, and Chao Yu. Harness VLA: Steering frozen VLAs into reliable manipulation primitives via memory-guided agents. *arXiv preprint arXiv:2607.08448*, 2026b. URL https://arxiv.org/abs/2607.08448v4.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. *arXiv preprint arXiv:2308.10144*, 2023. URL https://arxiv.org/abs/2308.10144.

Yunhao Zhao, Zhenyang Ni, Haoyang Chen, Ruohan Zhang, and Qi Zhu. Skills in weights, memory in code: Hybrid learning for memory-dependent robot manipulation. *arXiv preprint arXiv:2608.09410*, 2026. URL https://arxiv.org/abs/2608.09410.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *International Conference on Machine Learning*, 2024.

Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. LIBERO-PRO: Towards robust and fair evaluation of vision-language-action models beyond memorization. *arXiv preprint arXiv:2510.03827*, 2025.

A TASK GUIDANCE

We describe task guidance through the six components below. A concrete decision-table representation expresses these components through observable conditions, executable actions, progress checks, and conditional routing. The prompt and example show how the contrast between actions from one state becomes a reusable instruction for a later single-trajectory episode.

Table 4: Structure of the task guidance used by RoboBranch.

Section	Content
Applicability	Task and scene conditions under which a procedure is supported.
Procedure (Do)	Supported action order and parameters; scene-dependent entities and geometry are grounded from live observations.
Avoid	Actions or action orders to avoid under stated conditions, supported by observed failures or inferior outcomes.
Success checks	Observations or task checks that distinguish a physical effect from a tool return.
Recovery	Supported responses to observed failures, with conditions rather than universal retry counts.
Limitations	Missing evidence, unresolved ambiguity, or conditions outside the source experience.

A.1 EXPLORATION PROCEDURE

Algorithm 1 summarizes the exploration loop of Section 3 for one task. The learner selects a continuation at every decision and updates the task guidance once per round. An exploration episode ends at task success or at the step limit of the benchmark.

Algorithm 1 Branching exploration and guidance learning for one task.

Require: frozen agent π, learner L, simulator with snapshot and restore, branch width N, exploration rounds R

$\mathcal{G} \leftarrow \mathcal{G}_0$ {empty guidance document}
for $m = 1, \ldots, R$ **do**
  reset to an exploration instance; $s \leftarrow$ its initial state; $\mathcal{D}_m \leftarrow \emptyset$
  while the episode has not ended **do**
    $a^{(1)}, \ldots, a^{(N)} \leftarrow \pi(s, \mathcal{G})$ {one response proposes all candidates}
    save a snapshot of s
    execute each $a^{(i)}$ from its own copy of the snapshot, in parallel: $\mathcal{B} \leftarrow \{b^{(i)}\}_{i=1}^{N}$
    $\mathcal{D}_m \leftarrow \mathcal{D}_m \cup \{\mathcal{B}\}$
    $i^* \leftarrow L.\textsc{Select}(\mathcal{B})$ {most task progress, preferring task success}
    $s \leftarrow$ terminal state of $b^{(i^*)}$ {continue from the selected branch}
  end while
  $\mathcal{G} \leftarrow L.\textsc{Update}(\mathcal{G}, \mathcal{D}_m)$ {uses every outcome of every branch set}
end for
freeze $\mathcal{G}$; evaluate each episode as a single trajectory without branching

Table 5 lists the signals available to each role. At evaluation, RoboBranch and Harness VLA + Qwen receive the same observations, task-progress signals, and tools, and RoboBranch additionally follows its frozen task guidance. Outcomes of other candidates from the same state reach only the learner, during exploration.

Table 5: Signals available to each role.

Signal	Agent (exploration)	Learner	Agent (evaluation)	Harness VLA + Qwen
Task instruction	✓	✓	✓	✓
Camera images	✓	✓	✓	✓
Robot state	✓	✓	✓	✓
Task-progress and success signals	✓	✓	✓	✓
Read-only perception tools	✓	–	✓	✓
Results of executed actions	✓	✓	✓	✓
Task guidance	current	current, revised	frozen	stored traces
Other candidates from the same state	–	✓	–	–

A.2 GUIDANCE LEARNING PROMPT

The prompt template below expresses the guidance-update operation in Section 3.3.

Guidance-update prompt template

You are the learner for a robot-use agent. Revise the current task guidance using the supplied exploration outcomes. The agent, VLA, and tool interface are fixed; your output updates only the task guidance.

Inputs. Read the current guidance and all branch sets collected in this exploration round. Within each branch set, every candidate starts from the same saved state. Each candidate includes its action, terminal observation, task-progress feedback, task-success result, execution errors, and whether its terminal state was selected to continue exploration.

Compare outcomes. Inspect every candidate, including unselected and unsuccessful candidates. Distinguish task success, local progress, no observed effect, and execution errors. Compare actions within their shared starting state before drawing conclusions about their effects. A tool return, gripper closure, or end-effector motion alone does not establish task success. An action rationale states an intention, not an observed outcome.

Revise the guidance. Preserve supported existing entries. Add instructions for uncovered decisions and revise entries when the new outcomes support a change. For each entry, state its applicability, procedure (Do), actions to avoid (Avoid), success checks, supported recovery, and limitations. Keep alternatives conditional on the observed state.
The selected branches form the executed continuation. Unselected branches are alternatives from their shared origins; do not concatenate them into an executed trajectory. Distinguish a sequence that was actually executed from a proposed sequence that still needs exploration.

Ground each revision. Cite the supporting exploration outcomes for every added or revised entry. Each citation should identify the branch set and candidate and summarize the observed effect supporting the claim. Retain uncertainty when evidence is missing or contradictory. Keep these evidence references separate from the instructions that the evaluation agent follows.
Use only observations and tools available through the supplied interface. Bind scene-dependent objects and geometry from current observations. Do not turn an episode's seed, pixel location, or object coordinates into a task-wide instruction.

Output. Return the revised guidance together with an evidence reference for each changed entry.

A.3 EXAMPLE: OPENDRAWER

The example below summarizes the lesson in Figure 5 using the six guidance components listed above.

OpenDrawer: example task guidance

Applicability. The task is to open the drawer named in the task instruction. Identify the drawer and its handle from the current view; the side and position of the drawer come from the live instruction and observations.

Procedure (Do).
1. Start with the full-body VLA on the task instruction to bring an open gripper to the front of the named drawer. Do not add base motion before this call.
2. If the open fingers are near the handle but not aligned with it, move the open gripper to the front of the visible handle.
3. From the aligned open-gripper pose, call the arm-only VLA with a single instruction that both grasps the handle and pulls the drawer open.

Success checks. The task succeeds only when the success check reports the drawer open. Low reaching error, open fingers, or small drawer motion alone do not establish success. Stop once the drawer is reported open.

Recovery. If a grasp-only instruction or a scripted grasp stalls at the handle, reopen the gripper, realign it in front of the handle, and return to the combined grasp-and-pull instruction.

Avoid. Grasp-only VLA instructions at the handle; repeating a grasp attempt that has already stalled.

Limitations. The guidance assumes the full-body VLA call reaches a pose near the handle. If it does not, inspect the current scene and choose an action applicable to the visible drawer and available access.

Evidence summary. In Figure 5, one of ten branches from the same saved state opens the drawer with a single grasp-and-pull arm VLA call; the other nine, including a grasp-only instruction, leave it closed.

B Trajectory and Branch Statistics

B.1 From Guidance to a Single Trajectory

Figure 6 follows a RoboBranch evaluation episode on OpenDrawer with the frozen guidance learned in Figure 5. Exploration used a different kitchen and asked for the left drawer; this episode asks for the right drawer. Each decision follows one step of the learned procedure (Appendix A.3): the full-body VLA call leaves the drawer closed, the agent aligns the open gripper in front of the handle as the second step prescribes, and the grasp-and-pull arm VLA call opens the drawer, reaching task success after three decisions.

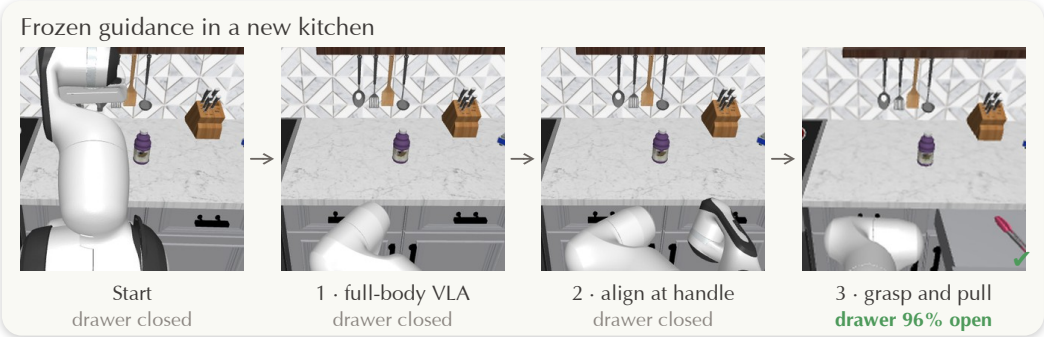

Figure 6: **Executing guidance along one trajectory.** The initial state and the endpoint of each of three consecutive decisions on OpenDrawer. Below each frame is the recorded drawer state. The final frame reaches task success.

B.2 One Branch Set in Full

Table 6 lists the ten candidates of the branch set in Figure 5. All ten execute from the same saved state, with the open gripper in front of the drawer handle. Only the arm VLA call that grasps and pulls the handle opens the drawer and reaches task success; the learner selects it and marks it good. The scripted grasp moves the drawer slightly without securing the handle, and the learner marks it good as partial progress. The grasp-only VLA call stalls and is marked bad. The remaining seven motions leave the drawer closed.

Table 6: The ten candidates of the OpenDrawer branch set in Figure 5. Drawer opening is read from the task-progress signal at the end of each branch.

#	Candidate action	Drawer opening	Outcome	Learner annotation
0	Close the gripper	0.0%	closed	–
1	Small arm motion, gripper open	0.0%	closed	–
2	Scripted grasp at the handle	6.8%	closed	good (partial progress)
3	Arm VLA: close on the handle	0.0%	closed	bad
4	Small arm motion	0.0%	closed	–
5	Rotate the gripper	0.0%	closed	–
6	Small arm motion, gripper open	0.0%	closed	–
7	Small arm motion, gripper closed	0.0%	closed	–
8	Arm VLA: grasp and pull the handle	97.5%	**success**	selected, good
9	Move the open gripper lower	0.0%	closed	–

B.3 Evidence Beyond the Selected Branch

We count the learner's informative-candidate annotations over the PreSoakPan exploration episodes: 63 branch sets and 630 executed candidates. The learner marks useful progress or preparation as good and concrete failures or inferior alternatives as bad; these labels describe exploration evidence,

rather than episode success. Table 7 separates candidates selected for continuation from the other candidates.

Table 7: Informative candidate annotations in PreSoakPan exploration.

Annotation	Selected	Unselected	Total
Good	49	41	90
Bad	0	385	385
Total	49	426	475

Unselected candidates contribute 41/90 (45.6%) of the good annotations and all 385 bad annotations. Overall, 426/475 (89.7%) of informative annotations come from unselected candidates; all 63 branch sets contain at least one such candidate. These records show the additional evidence available beyond the branch used to continue the trajectory.

C Benchmark Composition and Success Metrics

Table 8 lists the task–seed composition of the evaluation suites. For LIBERO-Pro, a task is counted within its perturbation condition: the four families are evaluated under instruction-redirection (T) and position-swap (S), giving eight splits.

Table 8: Evaluation suite composition.

Benchmark / split	Tasks	Seeds per task	Episodes
LIBERO-Pro (eight splits)	8×10	10	800
RoboCasa365: Atomic-Seen	18	10	180
RoboCasa365: Composite-Seen	16	5	80
RoboCasa365: Composite-Unseen	16	5	80
RoboTwin C2R	50	5	250

Exploration budget. Table 9 reports the exploration budget of RoboBranch per task, averaged over the tasks of each benchmark. Each decision executes ten candidate branches, so the number of executed branches is ten times the number of decisions.

Table 9: Average exploration budget per task with ten candidates per decision. Model tokens count both the agent and the learner.

Benchmark	Rounds	Decisions	Executed branches	Model tokens
LIBERO-Pro	8	96	960	6.07M
RoboCasa365	8	62.5	625	4.85M
RoboTwin C2R	8	32	320	2.39M

Cost per decision. Table 10 compares the cost of one decision of the exploration trajectory across the settings of Sections 5.3 and 5.4, averaged over the three benchmarks. For independent rollouts, a decision counts all ten rollouts. Ten independent rollouts execute as many branches and about as many simulator steps as branching with ten candidates, while using about ten times the agent tokens and 3.6 times the learner tokens.

Task-macro success. Let K be the number of tasks, n_k the number of evaluation seeds for task k, and $y_{k,j} \in \{0, 1\}$ the task-success outcome for its jth seed. The task-macro success rate is

$$S_{\text{macro}} = \frac{100}{K} \sum_{k=1}^{K} \frac{1}{n_k} \sum_{j=1}^{n_k} y_{k,j}. \tag{3}$$

Every task has equal weight. When all tasks have the same number of seeds, this equals the episode-level success rate. Thus the overall rates for LIBERO-Pro and RoboTwin C2R reduce to $100s/800$ and $100s/250$, respectively, where s is the total number of successful episodes.

Table 10: Average cost per decision of the exploration trajectory, over the three benchmarks.

Setting	Branches	Simulator steps	Agent tokens	Learner tokens
RoboBranch (10 candidates)	10	4,607	22,316	47,708
5 candidates	5	2,632	22,347	30,813
1 candidate	1	482	21,393	17,441
Independent rollouts ($\times$10)	10	4,920	213,233	174,060
Selected outcomes only	10	4,728	22,728	26,189
Successful outcomes only	10	4,656	22,245	11,857

RoboCasa365 weighting. Let s_{AS}, s_{CS}, and s_{CU} denote the successful episode counts in the three splits. Because atomic tasks have twice as many seeds as composite tasks, task-macro success is

$$S_{\mathrm{macro}} = 100 \left(\frac{18}{50} \frac{s_{\mathrm{AS}}}{180} + \frac{16}{50} \frac{s_{\mathrm{CS}}}{80} + \frac{16}{50} \frac{s_{\mathrm{CU}}}{80} \right) \tag{4}$$

$$= 100 \, \frac{s_{\mathrm{AS}} + 2s_{\mathrm{CS}} + 2s_{\mathrm{CU}}}{500}. \tag{5}$$

The denominator 500 is a weighting normalization; the suite contains 340 episodes. Episode-level success instead uses $100(s_{\mathrm{AS}} + s_{\mathrm{CS}} + s_{\mathrm{CU}})/340$. We compute overall rates from counts before rounding the displayed split percentages.

D Branch Evidence

The branch record in Section 3.2 links an action to its origin and observed outcome. Table 11 summarizes the information used to compare candidates. Selection determines the continuation of exploration; the selected flag is separate from the observed success or progress of a branch.

Table 11: Information associated with a candidate branch.

Information	Role
Shared origin	Identifies the decision state from which the candidates execute.
Candidate action	Records the tool, instruction, and action arguments.
Terminal observation	Shows the state reached after the candidate executes.
Task-progress feedback	Identifies changes in intermediate task conditions.
Task success	Records whether the environment's task-success check is satisfied.
Execution status	Records errors and whether the action reached its control horizon.
Selected continuation	Identifies the terminal state used for the next exploration decision.

Task success establishes task completion. Local progress describes an observed advance toward a subgoal without establishing completion. No observed effect means the available observations do not show such an advance. Execution errors describe problems carrying out an action. The record retains these signals separately, allowing the learner to compare useful intermediate effects as well as completed tasks.

E Runtime Measurement Details

Table 12: Time per branching step on RoboCasa365 (s) for a serial implementation (one VLM call per candidate, one branch at a time) and RoboBranch with five and ten candidates, excluding one-time initialization.

N	Execution	VLM	Rollout	Other	Total	Outcomes/h
5	Serial	20.94	111.74	2.33	135.02	133
	RoboBranch	5.80	34.22	0.53	40.55	444
10	Serial	40.80	226.22	4.64	271.66	133
	RoboBranch	7.79	36.23	0.44	44.46	810

The proposal-plus-rollout measurements in Table 12 run on RoboCasa365 with branch widths $N \in \{5, 10\}$. Serial and parallel execution are measured in pairs from the same simulator snapshot, in alternating order, after a warm-up pair.

Proposal and execution. Both conditions restore the same initial simulator snapshot within each measurement pair. Serial execution makes N independent live VLM proposals and executes each

proposed action before requesting the next. Parallel execution makes one live VLM proposal containing N candidates and dispatches the candidates concurrently. Proposal conversations are reset, and earlier branch feedback is not supplied to later proposals.

Timing boundaries. Total wall-clock time includes visual observation, VLM requests and their read-only tool calls, validation, rollout execution, cleanup, and per-call evidence serialization. Rollout time includes VLA inference, simulation, rendering, and communication. One-time service, episode, and model-client initialization is excluded. The *Other* column is the residual after subtracting the measured VLM and rollout intervals from total wall-clock time.

Aggregation. For each condition, we average total elapsed time over the measured branch sets. Speedup is the ratio of serial and parallel mean times:

$$\text{Speedup} = \frac{\overline{T}_{\text{serial}}}{\overline{T}_{\text{parallel}}}, \qquad \text{Outcomes/hour} = \frac{3600N}{\overline{T}}. \tag{6}$$

These rates measure the full proposal-and-execution workflow. At $N = 10$, the means of 271.66 s and 44.46 s give a $6.11\times$ speedup and approximately 133 and 810 outcomes per hour, respectively.

E.1 Variation in Branch-Set Latency

Table 13 reports the spread of the measured proposal-plus-rollout times per condition under the protocol above. Each row shows the arithmetic mean, sample standard deviation, and observed range, excluding warm-up pairs.

Table 13: Variation in total time per branch set (seconds).

N	Execution	Mean $\pm$ SD	Range
5	Serial	135.02 ± 15.40	118.99–163.00
5	RoboBranch	40.55 ± 10.02	29.90–52.08
10	Serial	271.66 ± 27.47	243.60–312.42
10	RoboBranch	44.46 ± 12.89	32.38–64.69

For both widths, the largest measured parallel time is below the smallest measured serial time. The reduction in elapsed time is therefore visible throughout these measurements, as well as in their means.